1.

**from** nonlinearEquation **import** \*  
**import** openpyxl **as** op  
**import** matplotlib.pyplot **as** plt  
**from** mpl\_toolkits.mplot3d **import** Axes3D  
**import** time  
  
# Constants  
a1 = 20  
a2 = 10  
b1 = 15  
b2 = 20  
theta = 30  
delta = 0.001  
# P\_START = (0.000000, 50000.000000, 5000.000000)  
# P\_TARGET = (100000.000000, 59652.343380, 5022.001164)  
P\_START = (0.000000, 50000.000000, 5000.000000)  
P\_TARGET = (100000.000000, 74860.5488999781, 5499.61109489643)  
l\_len = math.sqrt(math.pow((P\_TARGET[0]-P\_START[0]), 2)+math.pow((P\_TARGET[1]-P\_START[1]), 2)+math.pow((P\_TARGET[2]-P\_START[2]), 2))  
eAB = ((P\_TARGET[0]-P\_START[0])/l\_len, (P\_TARGET[1]-P\_START[1])/l\_len, (P\_TARGET[2]-P\_START[2])/l\_len)  
count = 0  
qiedian = [P\_START]  
  
# set-decorate  
# Control the use of greedy Rules  
# Closure should be learned  
  
  
**class DataProcessor**(object):  
 **def** \_\_init\_\_(self, path):  
 self.path = path  
 **return** # Get Data from Excel  
 # Could think about Transformation  
 **def get\_source**(self):  
 df = pd.read\_excel(self.path)  
 **return** df[['X坐标（单位: m）', 'Y坐标（单位:m）', 'Z坐标（单位: m）', '校正点类型', '第三问点标记']]  
  
 @staticmethod  
 # Find available area of solution  
 **def find\_avaliable**(p\_data, greedObj, p\_last, flag=1):  
 t = 0  
 **global** qiedian  
 narrow\_data\_0 = p\_data[((p\_data['X坐标（单位: m）'] < (greedObj.p\_cur[0]+1\*greedObj.d\_max)) &  
 (p\_data['X坐标（单位: m）'] > greedObj.p\_cur[0]) &  
 (p\_data['Y坐标（单位:m）'] < (greedObj.p\_cur[1]+1\*greedObj.d\_max)) &  
 (p\_data['Y坐标（单位:m）'] > (greedObj.p\_cur[1]-1\*greedObj.d\_max)) &  
 (p\_data['Z坐标（单位: m）'] < (greedObj.p\_cur[2]+1\*greedObj.d\_max)) &  
 (p\_data['Z坐标（单位: m）'] > (greedObj.p\_cur[2]-1\*greedObj.d\_max)) &  
 ((p\_data['校正点类型'] == greedObj.cate) |  
 (p\_data['校正点类型'] == 'A 点') |  
 (p\_data['校正点类型'] == 'B点'))  
 )][['X坐标（单位: m）',  
 'Y坐标（单位:m）',  
 'Z坐标（单位: m）',  
 '第三问点标记']]  
 # print(narrow\_data\_0)  
 list\_0 = narrow\_data\_0.index.tolist()  
 narrow\_data\_0['dist\_line'] = **None** # 612, 326  
 **if** 326 **in** narrow\_data\_0.index.tolist():  
 t = 1  
 min\_l = (30 - max(greedObj.tor[0], greedObj.tor[1]))/delta  
 coord = narrow\_data\_0.loc[326, :]  
 square = math.pow((coord[0]-greedObj.p\_cur[0]), 2) + \  
 math.pow((coord[1]-greedObj.p\_cur[1]), 2) + \  
 math.pow((coord[2]-greedObj.p\_cur[2]), 2)  
 dist\_calc = math.sqrt(square)  
 **if** dist\_calc < min\_l:  
 **return** narrow\_data\_0, t  
 **else**:  
 print('直飞也到达不了')  
 t = 0  
 **for** index **in** list\_0:  
 p\_temp = narrow\_data\_0.loc[index, :]  
 dist\_calc = 0  
 **if** flag == 1:  
 square = math.pow((p\_temp[0]-greedObj.p\_cur[0]), 2) + \  
 math.pow((p\_temp[1]-greedObj.p\_cur[1]), 2) + \  
 math.pow((p\_temp[2]-greedObj.p\_cur[2]), 2)  
 dist\_calc = math.sqrt(square)  
 **elif** flag == 2:  
 p\_O, e\_vertical = calc\_O\_coord(p\_last, greedObj.p\_cur, p\_temp)  
 **if** p\_O **is not None**:  
 **if** p\_O == 1:  
 square = math.pow((p\_temp[0]-greedObj.p\_cur[0]), 2) + \  
 math.pow((p\_temp[1]-greedObj.p\_cur[1]), 2) + \  
 math.pow((p\_temp[2]-greedObj.p\_cur[2]), 2)  
 dist\_calc = math.sqrt(square)  
 **else**:  
 p\_D = fsolve(calc\_D, [greedObj.p\_cur[0], greedObj.p\_cur[1], greedObj.p\_cur[2]], args=(p\_last, greedObj.p\_cur, p\_temp, p\_O, e\_vertical))  
 # Assert p\_D to check  
 eOD = (p\_D[0]-p\_O[0], p\_D[1]-p\_O[1], p\_D[2]-p\_O[2])  
 mm\_OD = math.sqrt(math.pow((p\_D[0]-p\_O[0]), 2) + \  
 math.pow((p\_D[1]-p\_O[1]), 2) + \  
 math.pow((p\_D[2]-p\_O[2]), 2))  
 eOB = (greedObj.p\_cur[0]-p\_O[0], greedObj.p\_cur[1]-p\_O[1], greedObj.p\_cur[2]-p\_O[2])  
 beta = math.acos(np.dot(eOD, eOB)/(200\*mm\_OD))  
 **if** beta > math.pi/2:  
 print('Solution Should be checked')  
 # dist\_calc = 1e6  
 dist\_calc = 200\*beta + math.sqrt(math.pow((p\_temp[0]-p\_O[0]), 2) + \  
 math.pow((p\_temp[1]-p\_O[1]), 2) + \  
 math.pow((p\_temp[2]-p\_O[2]), 2))  
 **else**:  
 dist\_calc = 200\*beta + math.sqrt(math.pow((p\_temp[0]-p\_O[0]), 2) + \  
 math.pow((p\_temp[1]-p\_O[1]), 2) + \  
 math.pow((p\_temp[2]-p\_O[2]), 2))  
 narrow\_data\_0.loc[index, 'dist\_line'] = dist\_calc  
 **if** dist\_calc >= 1\*greedObj.d\_max:#-400\*math.pi:  
 narrow\_data\_0 = narrow\_data\_0.drop([index])  
 **if** narrow\_data\_0.shape[0] < 1:  
 **raise** ValueError('No Available choice')  
 **return** narrow\_data\_0, t  
  
  
**class GreedyObject**(object):  
 **def** \_\_init\_\_(self, rule\_flag, vi=0, hi=0):  
 self.rule\_ctrl = rule\_flag  
 self.tor = (vi, hi)  
 self.cate = 1  
 self.p\_cur = (0.0, 0.0, 0.0)  
 self.d\_max = 0  
 self.result\_line = []  
 self.result\_length = 0  
 self.coord\_line = []  
 self.coord\_line.append(P\_START)  
 **return  
  
 def category**(self):  
 # v\_flag = 0  
 # h\_flag = 0  
 temp\_a = (a1-self.tor[0], a2-self.tor[1])  
 temp\_b = (b1-self.tor[0], b2-self.tor[1])  
 **if** temp\_a[0] <= temp\_a[1]:  
 min\_a = temp\_a[0]  
 **else**:  
 min\_a = temp\_a[1]  
 **if** temp\_b[0] <= temp\_b[1]:  
 min\_b = temp\_b[0]  
 **else**:  
 min\_b = temp\_b[1]  
 **if** min\_a <= min\_b:  
 # v\_flag = 1  
 # First Time Choice  
 **if** self.tor[0] == 0 **and** self.tor[1] == 0:  
 self.cate = 1  
 self.d\_max = min\_a/delta  
 **return  
 if** self.tor[0] **is not** 0:  
 self.cate = 1  
 self.d\_max = min\_a/delta  
 **else**:  
 self.cate = 0  
 self.d\_max = min\_b/delta  
 **else**:  
 # h\_flag = 1  
 **if** self.tor[1] **is not** 0:  
 self.cate = 0  
 self.d\_max = min\_b/delta  
 **else**:  
 self.cate = 1  
 self.d\_max = min\_a/delta  
 **return  
  
 def choose**(self, avaliable\_data):  
 list\_index = avaliable\_data.index.tolist()  
 comp\_list = []  
 du\_list = []  
 **for** i **in** list\_index:  
 p\_temp = avaliable\_data.loc[i, :]  
 t\_projection = calc\_projection(p\_temp, self.p\_cur, self.d\_max, self.rule\_ctrl)  
 # Compare To Choose With angle and line-length  
 comp\_list.append((str(i), t\_projection, avaliable\_data.loc[i, ['dist\_line']]))  
 result\_list = sorted(comp\_list, key=**lambda** x: x[1], reverse=**True**)  
 # 轮盘赌  
 cum = 0  
 **for** item **in** result\_list:  
 du\_list.append(item[1])  
 cum += item[1]  
 du\_cumsum = np.cumsum(du\_list)  
 rand\_t = np.random.uniform(0, 1, size=[1, 1])[0][0]  
 **if** rand\_t < du\_cumsum[0]/cum:  
 **return** result\_list[0]  
 **for** i **in** range(1, len(result\_list)):  
 **if** (rand\_t >= du\_cumsum[i-1]/cum) **and** (rand\_t < du\_cumsum[i]/cum):  
 **return** result\_list[i]  
  
 **def after\_choose**(self, avaiable\_data, best\_choice):  
 b\_index = int(best\_choice[0])  
 b\_series = avaiable\_data.loc[b\_index, :]  
 dist\_cost = b\_series['dist\_line']  
 self.p\_cur = (b\_series['X坐标（单位: m）'], b\_series['Y坐标（单位:m）'], b\_series['Z坐标（单位: m）'])  
 is\_cls = b\_series['第三问点标记']  
 # Update vi,hi  
 self.tor = (self.tor[0]+dist\_cost\*delta, self.tor[1]+dist\_cost\*delta)  
 print(self.tor)  
 **if** self.rule\_ctrl == 1 **or** self.rule\_ctrl == 2:  
 **if** self.cate:  
 self.tor = (0, self.tor[1])  
 **else**:  
 self.tor = (self.tor[0], 0)  
 **elif** self.rule\_ctrl == 3:  
 **if** is\_cls == 1:  
 tmp = np.array(np.random.uniform(0, 1, size=[1, 1]))[0][0]  
 print(tmp)  
 **if** self.cate:  
 **if** tmp >= 0.2:  
 self.tor = (0, self.tor[1])  
 **else**:  
 self.tor = (min(self.tor[0], 5), self.tor[1])  
 **else**:  
 **if** tmp >= 0.2:  
 self.tor = (self.tor[0], 0)  
 **else**:  
 self.tor = (self.tor[0], min(self.tor[1], 5))  
 **else**:  
 **if** self.cate:  
 self.tor = (0, self.tor[1])  
 **else**:  
 self.tor = (self.tor[0], 0)  
 print(self.tor)  
 **return  
  
 def run**(self, origin\_data):  
 **global** count  
 **global** qiedian  
 print('Start Searching')  
 t1 = time.time()  
 greed\_obj.p\_cur = P\_START  
 print(P\_START)  
 greed\_obj.category()  
 ava\_data, \_ = DataProcessor.find\_avaliable(origin\_data, greed\_obj, P\_START, 1)  
 prime\_choice = greed\_obj.choose(ava\_data)  
 greed\_obj.result\_line.append(prime\_choice[0])  
 greed\_obj.result\_length += prime\_choice[2]  
 greed\_obj.after\_choose(ava\_data, prime\_choice)  
 greed\_obj.coord\_line.append(greed\_obj.p\_cur)  
 print(greed\_obj.p\_cur)  
 # \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 **try**:  
 **while True**:  
 greed\_obj.category()  
 ava\_data, t\_flag = DataProcessor.find\_avaliable(origin\_data, greed\_obj, greed\_obj.coord\_line[-2], 1)  
 # Ending  
 **if** t\_flag:  
 greed\_obj.result\_length += math.sqrt(math.pow((P\_TARGET[0]-greed\_obj.p\_cur[0]), 2) +\  
 math.pow((P\_TARGET[1]-greed\_obj.p\_cur[1]), 2) +\  
 math.pow((P\_TARGET[2]-greed\_obj.p\_cur[2]), 2))  
 **break** prime\_choice = greed\_obj.choose(ava\_data)  
 greed\_obj.result\_line.append(prime\_choice[0])  
 greed\_obj.result\_length += prime\_choice[2]  
 greed\_obj.after\_choose(ava\_data, prime\_choice)  
 greed\_obj.coord\_line.append(greed\_obj.p\_cur)  
 print(greed\_obj.p\_cur)  
 print(P\_TARGET)  
 print('Reaching Target')  
 print('Finish')  
 print(greed\_obj.result\_line)  
 **with** open('case2.txt', 'a') **as** f:  
 f.write('0 ')  
 **for** item **in** greed\_obj.result\_line[0:-1]:  
 f.write(item)  
 f.write(' ')  
 f.write(greed\_obj.result\_line[-1])  
 f.write(' 326')  
 f.write('\n')  
 print('实际规划路程: %f' % greed\_obj.result\_length)  
 llength = math.sqrt(math.pow((P\_TARGET[0]-P\_START[0]), 2) +\  
 math.pow((P\_TARGET[1]-P\_START[1]), 2) +\  
 math.pow((P\_TARGET[2]-P\_START[2]), 2))  
 print('A-B直线距离: %f' % llength)  
 greed\_obj.coord\_line.append(P\_TARGET)  
 x\_l = []  
 y\_l = []  
 z\_l = []  
 **for** item **in** greed\_obj.coord\_line:  
 x\_l.append(item[0])  
 y\_l.append(item[1])  
 z\_l.append(item[2])  
 # fig = plt.figure()  
 # ax = Axes3D(fig)  
 # ax.plot3D(x\_l, y\_l, z\_l, c='r')  
 # ax.set\_xlabel('x')  
 # ax.set\_ylabel('y')  
 # ax.set\_zlabel('z')  
 # plt.show()  
 count += 1  
 t2 = time.time()  
 print('总用时: %f' % (t2 - t1))  
 **return  
 except** ValueError **as** e:  
 print(e)  
 # print('尝试失败一次')  
 **return** 99  
  
  
# Const P\_TARGET  
**def calc\_projection**(p\_avaliable, p\_cur, dmax, flag=1):  
 result\_f = 0  
 e\_try = (p\_avaliable[0]-p\_cur[0], p\_avaliable[1]-p\_cur[1], p\_avaliable[2]-p\_cur[2])  
 result = e\_try[0]\*eAB[0] + e\_try[1]\*eAB[1] + e\_try[2]\*eAB[2]  
 e\_try\_model = math.sqrt(math.pow(e\_try[0], 2) + \  
 math.pow(e\_try[1], 2) + \  
 math.pow(e\_try[2], 2))  
 cos\_angle = result/e\_try\_model  
 angle = math.acos(cos\_angle)  
 **if** flag **is** 1:  
 # if greed\_obj.cate:  
 # limit = b2-(greed\_obj.tor[1] + p\_avaliable[4]\*delta)  
 # else:  
 # limit = a1-(greed\_obj.tor[0] + p\_avaliable[4]\*delta)  
 result\_f = 1\*result\*math.cos(angle)# + 4\*limit/delta  
 **elif** flag == 2:  
 **if** greed\_obj.cate:  
 limit = b2-(greed\_obj.tor[1] + p\_avaliable[4]\*delta)  
 **else**:  
 limit = a1-(greed\_obj.tor[0] + p\_avaliable[4]\*delta)  
 result\_f = result\*math.cos(angle) + 4\*limit/delta - 2\*(12000-dmax)  
 **elif** flag == 3:  
 **if** greed\_obj.cate:  
 limit = b2-(greed\_obj.tor[1] + p\_avaliable[4]\*delta)  
 **else**:  
 limit = a1-(greed\_obj.tor[0] + p\_avaliable[4]\*delta)  
 # result\_f = 1\*result\*math.cos(angle) + 0.2\*limit/delta  
 result\_f = 1\*result\*math.cos(angle)  
 **return** result\_f  
  
  
# For Testing  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 # TDD \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 # df\_test = pd.read\_excel('test1.xlsx')  
 # print(df\_test[['编号', 'X坐标（单位: m）', 'Y坐标（单位:m）', 'Z坐标（单位: m）']])  
 # dObj = DataProcessor('test1.xlsx')  
 # df\_test = dObj.get\_source()  
 # print(df\_test['Y坐标（单位:m）'])  
 # print(df\_test)  
 # \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 # Get method should be override  
 # p\_cur = df\_test.loc[0, :]  
 # \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 # print(p\_cur[0], p\_cur[1], p\_cur[2])  
 # narrow\_data\_0 = df\_test[(df\_test['X坐标（单位: m）'] < (p\_cur[0]+15000)) &  
 # (df\_test['Y坐标（单位:m）'] < (p\_cur[1]+15000)) &  
 # (df\_test['Y坐标（单位:m）'] > (p\_cur[1]-15000)) &  
 # (df\_test['Z坐标（单位: m）'] < (p\_cur[2]+15000)) &  
 # (df\_test['Z坐标（单位: m）'] > (p\_cur[2]-15000)) &  
 # ((df\_test['校正点类型'] == 1) | (df\_test['校正点类型'] == 'A 点') | (df\_test['校正点类型'] == 'B点'))  
 # ][['X坐标（单位: m）', 'Y坐标（单位:m）', 'Z坐标（单位: m）']]  
 # list = narrow\_data\_0.index.tolist()  
 # narrow\_data\_0 = narrow\_data\_0.drop([0])  
 # print(narrow\_data\_0.drop([0]))  
 # print(narrow\_data\_0.shape[0])  
 # list\_0 = narrow\_data\_0.index.tolist()  
 # narrow\_data\_0['dist\_line'] = 0.0  
 # for index in list\_0:  
 # p\_temp = narrow\_data\_0.loc[index, :]  
 # square = math.pow((p\_temp[0]-p\_cur[0]), 2) + \  
 # math.pow((p\_temp[1]-p\_cur[1]), 2) + \  
 # math.pow((p\_temp[2]-p\_cur[2]), 2)  
 # dist\_calc = math.sqrt(square)  
 # narrow\_data\_0.loc[index, 'dist\_line'] = dist\_calc  
 # if dist\_calc >= 15000:  
 # narrow\_data\_0 = narrow\_data\_0.drop([index])  
 # print(narrow\_data\_0)  
 # print(narrow\_data\_0.shape[0])  
 # list\_choice = narrow\_data\_0.index.tolist()  
 # greedyone = GreedyObject(rule\_flag=1)  
 # greedyone.choose(narrow\_data\_0, p\_cur)  
 # \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 # Init Process  
 dObj = DataProcessor('test2.xlsx')  
 origin\_data = dObj.get\_source()  
 # greed\_obj = GreedyObject(1)  
 # greed\_obj.run(origin\_data)  
 # Not using Thread for now  
 **for** i **in** range(0, 500):  
 greed\_obj = GreedyObject(3)  
 greed\_obj.\_\_init\_\_(3)  
 greed\_obj.run(origin\_data)  
 print('成功: %d次' % count)  
 print('成功概率为: %f' % (count/500))

2.

**from** scipy.optimize **import** fsolve  
**import** numpy **as** np  
**import** math  
**import** pandas **as** pd  
**import** openpyxl **as** op

**def calc\_O\_coord**(p\_A, p\_B, p\_C):  
 eAB = (p\_B[0]-p\_A[0], p\_B[1]-p\_A[1], p\_B[2]-p\_A[2])  
 eAC = (p\_C[0]-p\_A[0], p\_C[1]-p\_A[1], p\_C[2]-p\_A[2])  
 eBC = (p\_C[0]-p\_B[0], p\_C[1]-p\_B[1], p\_C[2]-p\_B[2])  
  
 m\_AB = math.sqrt(math.pow(eAB[0], 2) + \  
 math.pow(eAB[1], 2) + \  
 math.pow(eAB[2], 2))  
  
 m\_AC = math.sqrt(math.pow(eAC[0], 2) + \  
 math.pow(eAC[1], 2) + \  
 math.pow(eAC[2], 2))  
  
 m\_BC = math.sqrt(math.pow(eBC[0], 2) + \  
 math.pow(eBC[1], 2) + \  
 math.pow(eBC[2], 2))  
  
 eAB\_norm = (eAB[0]/m\_AB, eAB[1]/m\_AB, eAB[2]/m\_AB)  
 eAC\_norm = (eAC[0]/m\_AC, eAC[1]/m\_AC, eAC[2]/m\_AC)  
 eBC\_norm = (eBC[0]/m\_BC, eBC[1]/m\_BC, eBC[2]/m\_BC)  
 e\_vertical = np.cross(eAB\_norm, eAC\_norm)  
 eBO = np.cross(e\_vertical, eAB\_norm)  
 m\_BO = math.sqrt(math.pow(eBO[0], 2) + \  
 math.pow(eBO[1], 2) + \  
 math.pow(eBO[2], 2))  
 eBO\_norm = (eBO[0]/m\_BO, eBO[1]/m\_BO, eBO[2]/m\_BO)  
 p\_o = **None** p\_O1 = (p\_B[0]+200\*eBO\_norm[0], p\_B[1]+200\*eBO\_norm[1], p\_B[2]+200\*eBO\_norm[2])  
 p\_O2 = (p\_B[0]-200\*eBO\_norm[0], p\_B[1]-200\*eBO\_norm[1], p\_B[2]-200\*eBO\_norm[2])  
 costheta = np.dot(eAB\_norm, eBC\_norm)  
 theta = math.acos(costheta)  
 # Using theta to filter solution  
 **if** theta >= math.pi/2:  
 p\_o = **None** # Compare dist to p\_O1 and p\_O2 to choose circle  
 dist1 = math.sqrt(math.pow((p\_O1[0]-p\_C[0]), 2) + \  
 math.pow((p\_O1[1]-p\_C[1]), 2) + \  
 math.pow((p\_O1[2]-p\_C[2]), 2))  
 dist2 = math.sqrt(math.pow((p\_O2[0]-p\_C[0]), 2) + \  
 math.pow((p\_O2[1]-p\_C[1]), 2) + \  
 math.pow((p\_O2[2]-p\_C[2]), 2))  
 **if** dist1 < dist2:  
 p\_o = p\_O1  
 **elif** dist1 == dist2:  
 p\_o = 1  
 **else**:  
 p\_o = p\_O2  
 **return** p\_o, e\_vertical  
  
  
**def calc\_D**(x, p\_A, p\_B, p\_C, p\_O, e\_vertical):  
 **return** np.array([  
 math.pow((x[0]-p\_O[0]), 2)+math.pow((x[1]-p\_O[1]), 2)+math.pow((x[2]-p\_O[2]), 2)-200\*200,  
 (x[0]-p\_O[0])\*(x[0]-p\_C[0])+(x[1]-p\_O[1])\*(x[1]-p\_C[1])+(x[2]-p\_O[2])\*(x[2]-p\_C[2]),  
 e\_vertical[0]\*x[0]+e\_vertical[1]\*x[1]+e\_vertical[2]\*x[2]  
 ])  
  
  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 # p\_A = (11392.9607416196, 56973.0182393612, 4097.85801775604)  
 df = pd.read\_excel('test1.xlsx')  
 wb = op.load\_workbook('circle.xlsx')  
 ws = wb.get\_active\_sheet()  
 # list\_index = ['237', '233', '598', '561', '485', '230', '204']  
 # df\_use = df.loc[[237, 233, 598, 561, 485, 230, 204], :]  
 point = [(0.0, 50000.0, 5000.0),  
 (11378.1726511, 51578.6462114014, 8507.54931763946),  
 (13673.89264016, 41761.0953816907, 2505.40244099741),  
 (25370.7267465017, 37771.3408601215, 2360.07858040412),  
 (34875.3903546956, 41542.8865332295, 3425.81494649522),  
 (40926.3162136232, 44081.4092780479, 1511.78763315207),  
 (49728.7278799095, 51092.0318641745, 3311.95653333703),  
 (54832.8870194109, 49179.2191080384, 49179.2191080384),  
 (61702.4691037079, 47818.7450700809, 6619.77069908595),  
 (69899.8143232754, 55457.2043281517, 1785.70124084295),  
 (80222.4027047731, 56403.3505412292, 7760.45863251371),  
 (87120.6704551626, 61363.5183965708, 288.585871405052),  
 (100000.0, 59652.34338, 5022.001164)]  
 rows = 1  
 cols = 5  
 list\_index = ['0', '578', '77', '165', '215', '244', '306', '2', '198', '166', '583', '368', '612']  
 df\_use = df.loc[[0, 578, 77, 165, 215, 244, 306, 2, 198, 166, 583, 368, 612], :]  
 **with** open('yuanxin.txt', 'wb') **as** f:  
 # ws.cell(row=rows, column=cols).value = point[0][0]  
 # ws.cell(row=rows, column=cols+1).value = point[0][1]  
 # ws.cell(row=rows, column=cols+2).value = point[0][2]  
 # ws.cell(row=rows+1, column=cols).value = point[1][0]  
 # ws.cell(row=rows+1, column=cols+1).value = point[1][1]  
 # ws.cell(row=rows+1, column=cols+2).value = point[1][2]  
 rows += 2  
 **for** i **in** range(0, 11):  
 df\_last = df\_use.loc[int(list\_index[i]), :]  
 df\_cur = df\_use.loc[int(list\_index[i+1]), :]  
 df\_next = df\_use.loc[int(list\_index[i+2]), :]  
 p\_A = (df\_last[1], df\_last[2], df\_last[3])  
 p\_B = (df\_cur[1], df\_cur[2], df\_cur[3])  
 p\_C = (df\_next[1], df\_next[2], df\_next[3])  
 p\_O, e\_vertical = calc\_O\_coord(p\_A, p\_B, p\_C)  
 # print(p\_O)  
 f.write(str(p\_O[0]).encode()+b', '+str(p\_O[1]).encode()+b', '+str(p\_O[2]).encode()+b'\n')  
 p\_D = fsolve(calc\_D, [p\_B[0], p\_B[1], p\_B[2]], args=(p\_A, p\_B, p\_C, p\_O, e\_vertical))  
 print(p\_D)  
 # f.write('切点: '.encode()+str(p\_D[0]).encode()+b', '+str(p\_D[1]).encode()+b', '+str(p\_D[2]).encode()+b'\n')  
 ws.cell(row=rows, column=cols).value = p\_D[0]  
 ws.cell(row=rows, column=cols+1).value = p\_D[1]  
 ws.cell(row=rows, column=cols+2).value = p\_D[2]  
 ws.cell(row=rows, column=cols+4).value = p\_O[0]  
 ws.cell(row=rows, column=cols+5).value = p\_O[1]  
 ws.cell(row=rows, column=cols+6).value = p\_O[2]  
 rows += 1  
 # ws.cell(row=rows, column=cols).value = point[i+2][0]  
 # ws.cell(row=rows, column=cols+1).value = point[i+2][1]  
 # ws.cell(row=rows, column=cols+2).value = point[i+2][2]  
 # rows += 1  
 m\_OD = math.sqrt(math.pow((p\_D[0]-p\_O[0]), 2) + \  
 math.pow((p\_D[1]-p\_O[1]), 2) + \  
 math.pow((p\_D[2]-p\_O[2]), 2))  
 print(m\_OD)  
 wb.save('circle.xlsx')  
 wb.close()