確率解析特論レポート課題

長見 剛
野中 淳史
余宮 由佳

1 概要

このレポートでは焼きなまし法により、実数平面上に分布する点をすべてたどり、元の位置に戻るまでの総移動距離の大域的最小値を求めた結果をまとめる。

まず初めに、python による焼きなまし法のアルゴリズムの枠組みを部分的に示すとともに、その説明を加えた。次に、そのアルゴリズムに対して、①螺旋状に分布している点、②星状に分布している点、③県庁所在地の 3 つの具体例を適応した結果をまとめた。最後に実行結果に対する結論を記した。

2 アルゴリズムについて

初めに、分布する点の数を 47 個、摂動回数 k を 400、温度関数で使用する定数cと変数tの取る値の上限 end をそれぞれ 30、2500 として設定しておく。点の数は、具体例でアルゴリズムを適応する際に用いる都道府県の数に合わせてある。

```
import random
import math
import numpy as np
import copy
import matplotlib.pyplot as plt

#点の数
n = 47
#クラスタリング・スケジュール(温度関数)の定数
c = 30
#クラスタリング・スケジュール(温度関数)の上界
end = 2500
#摂動回数の設定
k = 400
```

次に、温度関数 T(t)を$\frac{c}{\sqrt{t}+1}$ $\left(=\frac{30}{\sqrt{t}+1}\right)$として定義する。

```
#クラスタリング・スケジュール(温度関数)の設定
def T(temper):
    return c/(math.sqrt(temper)+1)
```

さらに、最初の経路は以下のように、まず city に 0 から n(=47)までの数値を順に入れていき、ランダムに入れ替えることとした。

```
#初めにめぐる点の順番
city = [i for i in range(n)]
random.shuffle(city)
```

点の位置(x座標とy座標)を次のように実数平面上にとり、その点に対して 0 から順番にn-1(=39)まで番号を振る。

① 螺旋状に分布する点

対数螺旋の式を用いて、螺旋状になるようにx座標とy座標を定めた。

```
#点の位置
x = [theta*math.cos(0.5*theta) for theta in range(n)]
y = [theta*math.sin(0.5*theta) for theta in range(n)]
x_y = {}
for i,j,r in zip(range(n),x,y):
    x_y[i] = [j, r]
```

② 星状に分布する点

ハイポトロコイドの方程式を用いて星状になるようにx座標とy座標を定めた。

```
x = [(2.5-1)*math.cos(0.27*theta) + 0.8*math.cos((2.5-
1)/1*0.27*theta) for theta in range(n)]
y = [(2.5-1)*math.sin(0.27*theta) - 0.8*math.sin((2.5-
1)/1*0.27*theta) for theta in range(n)]
x_y = {}
for i,j,r in zip(range(n),x,y):
    x_y[i] = [j, r]
```

③ 県庁所在地

各県庁所在地の経度をx、緯度をyとして定めた

```
x = [141.345505,140.74,141.1525,140.87194,140.1025,140.36333,140.46778,14
0.44667,139.88361,139.06083,139.64889,140.12333,139.69167,139.6425,139.02
361,137.21139,136.62556,136.22194,138.56833,138.18111,136.72222,138.38306
,136.90667,136.50861,135.86833,135.75556,135.52,135.18306,135.83278,135.1
675,134.23833,133.05056,133.935,132.45944,131.47139,134.55944,134.04333,1
32.76611,133.53111,130.41806,130.29889,129.87361,130.74167,131.6125,131.4
2389,130.55806,127.68111]
y = [43.06417,40.82444,39.70361,38.26889,39.71861,38.24056,37.75,36.34139
,36.56583,36.39111,35.85694,35.60472,35.68944,35.44778,37.90222,36.69528,
36.59444,36.06528,35.66389,36.65139,35.39111,34.97694,35.18028,34.73028,3
5.00444,35.02139,34.68639,34.69139,34.68528,34.22611,35.50361,35.47222,34
.66167,34.39639,34.18583,34.06583,34.34028,33.84167,33.55972,33.60639,33.
24944,32.74472,32.78972,33.23806,31.91111,31.56028,26.2125]
x_y = {}
for i,j,r in zip(range(n),x,y):
    x_y[i] = [j, r]
```

三平方の定理を用いて、設定した点の座標と経路をもとに総移動距離を求める関数 d を定義する。

```
#距離を求める関数の生成
def d(toshi):
    kyori = 0
    for i in range(n):
        if toshi[i] != toshi[n-1]:
            kyori += math.sqrt((x_y[toshi[i+1]][0]-
x_y[toshi[i]][0])**2+(x_y[toshi[i+1]][1]-x_y[toshi[i]][1])**2)
        else:
            kyori += math.sqrt((x_y[toshi[0]][0]-
x_y[toshi[i]][0])**2+(x_y[toshi[0]][1]-x_y[toshi[i]][1])**2)
    return kyori
```

時刻tを 0 から end (=1600)まで変化させ、各時刻において k(=400)回摂動を生成する。

ここで、下記の「＃摂動の生成」の部分については、各点の番号をランダムに 2 つ選び$a < b$となるようにa, bにおいた。そして、a番目からb番目までの巡回する順番を逆にし、それを city_near とした。

また「＃遷移」の部分は、摂動における移動距離の総和が、元の順番による移動距離の総和よりも小さい場合には順番を更新し、逆の場合においては、確率$exp\left\{-\frac{d(city-near)-d(city)}{T(t)}\right\}$で更新

し、確率$1-exp\left\{-\frac{d(city-near)-d(city)}{T(t)}\right\}$で順番を変えないものとした。

```
#大域的最小値の生成
for t in range(end):
    for i in range(k):
        #摂動の生成
        a = random.randint(1,n-1)
        b = random.randint(a+1,n)
        city_near = copy.copy(city)
        city_near[a:b] = reversed(city_near[a:b])
        #遷移
        if d(city) >= d(city_near):
            city = city_near
        else:
            P = (math.e)**(-(d(city_near)-d(city))/T(t))
            m = np.random.choice([0,1],p = [1-P,P])
            if m == 1:
                city = city_near
```

3　具体的な適用例を 3 種類

1. 螺旋状の分布

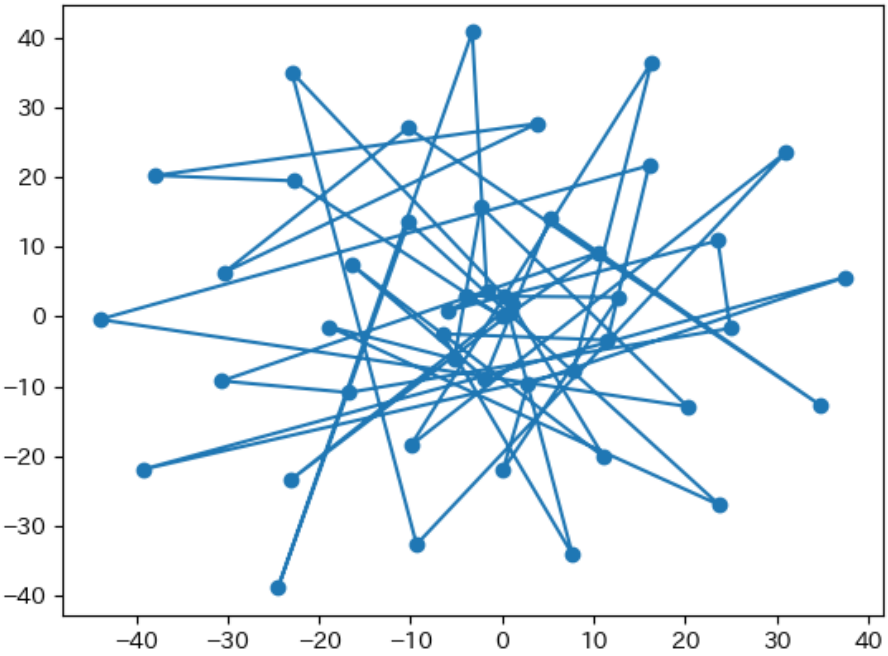
上記で述べたアルゴリズムを用いて、螺旋状に分布する点における初期の距離と大域的最小値を求めた結果、次のようになった。

初期の距離：1595.090791020644

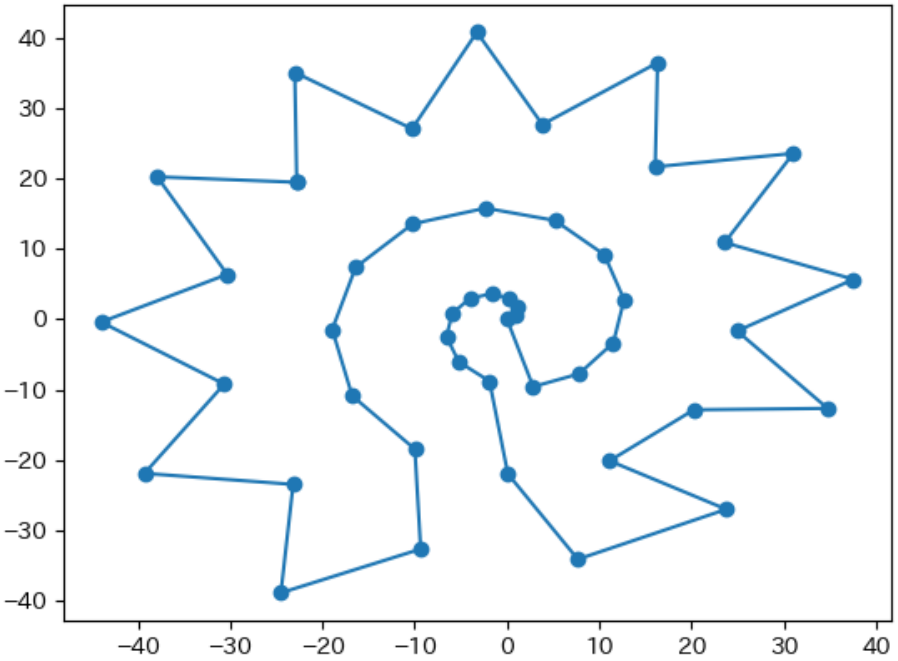
最小値：508.6667659817132

また、グラフは次のようになった。

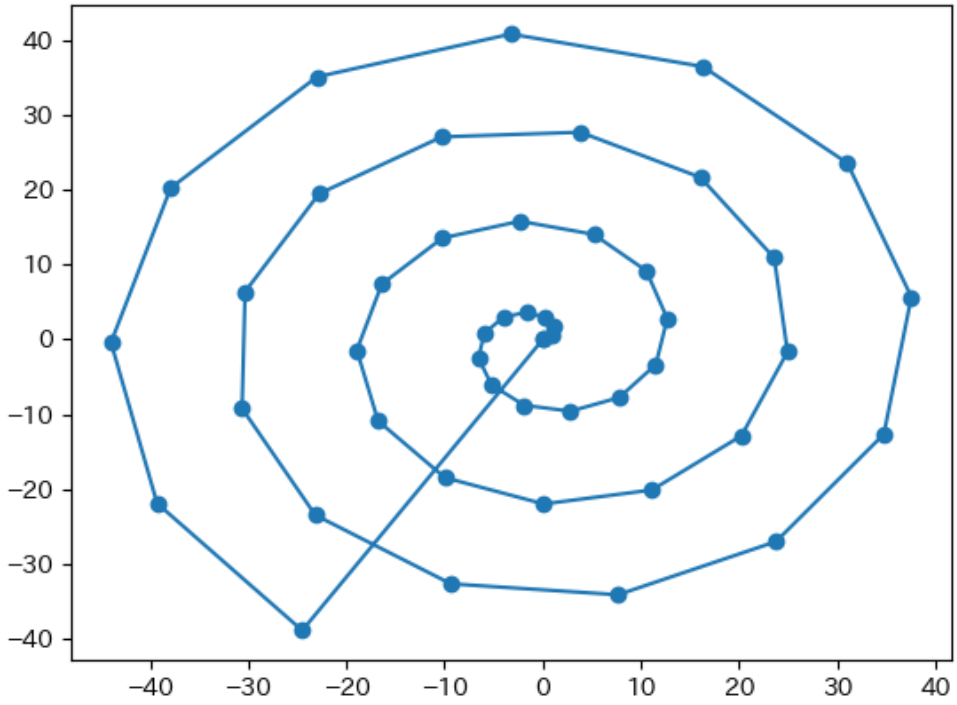
・初期


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

・結果

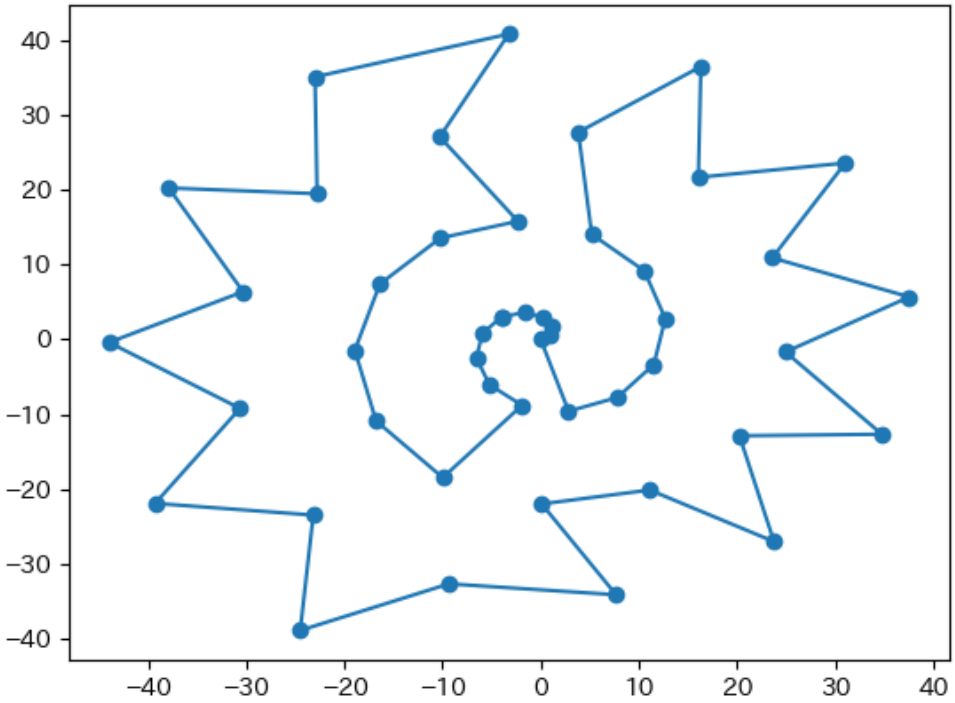

40
30
20
10
0
−10
−20
−30
−40
−40
−30
−20
−10
0
10
20
30
40

実行した結果、螺旋状に描かれないが、初期の状態と比較して改善されていることが分かった。また、次のように螺旋状に描いた場合の距離より、焼きなまし法で求めたもののほうが、より小さい距離を示していることが分かった。



螺旋状の場合の距離： 573.6127831109641

また、摂動回数を 2500 回から 10000 回にまで増やしたところ次のようになった。



大域的最小値 517.713775370516

摂動回数が 2500 回の時と比較して、数値が大きくなっているが、螺旋状の経路と比較すると小さい値をとっていることがわかる。

2. 星状の分布

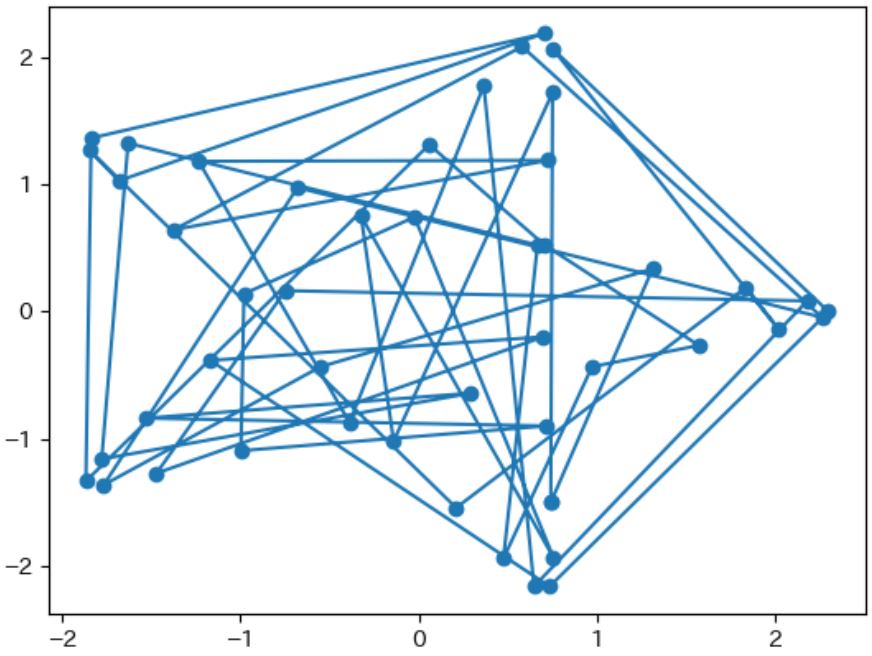
星状に分布する点における初期の距離と大域的最小値を紹介したアルゴリズムを用いて求めた結果、次のようになった。

初期の距離：104.82780796396561

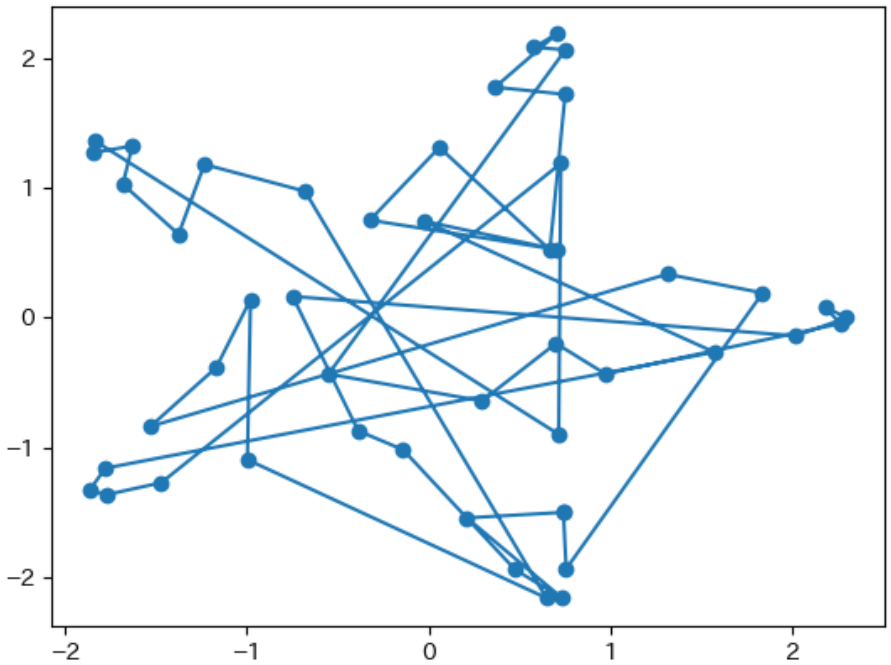
大域的最小値：50.726599178302074

また、グラフは次のようになった。

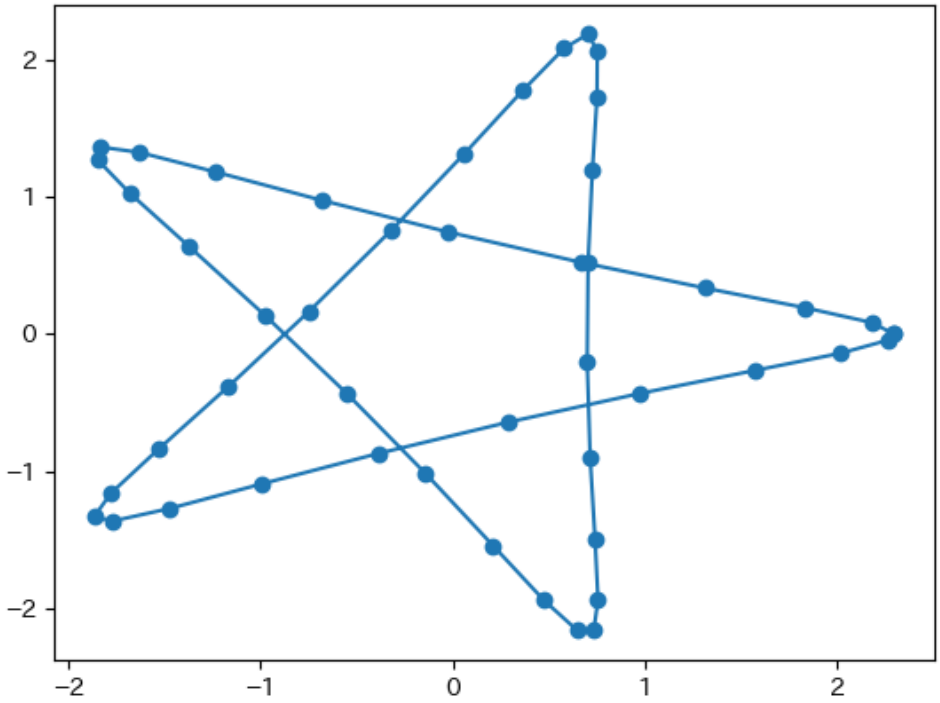
・初期



・結果

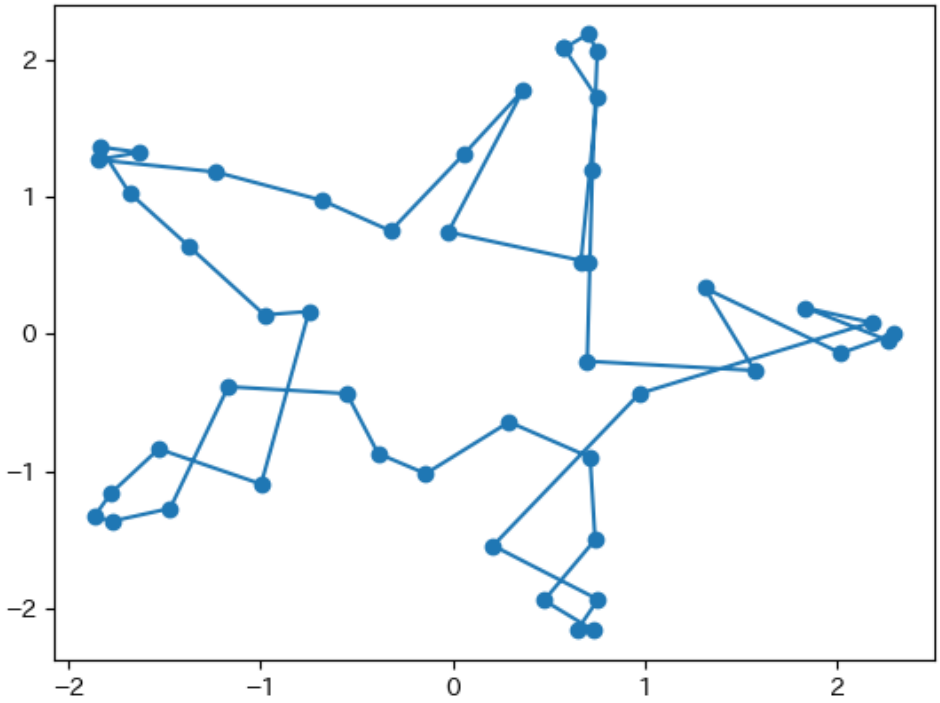


実行した結果、初期の状態より改善はされたが、図を見ると距離が最小となっていないように見える。実際、次のように星状に描いた場合、焼きなまし法により出た結果よりも小さい値が求められた。



星状の場合の距離： 21.929526487716775

また、摂動回数を 2500 から 10000 にまで増やした結果、次のようになった。



大域的最小値：26.09765360445637

摂動回数が 2500 の時と比べ改善されたが、星状に描いた場合に比べて距離は大きくなっており、明らかに最小の距離となっていないことがわかる

3. 県庁所在地

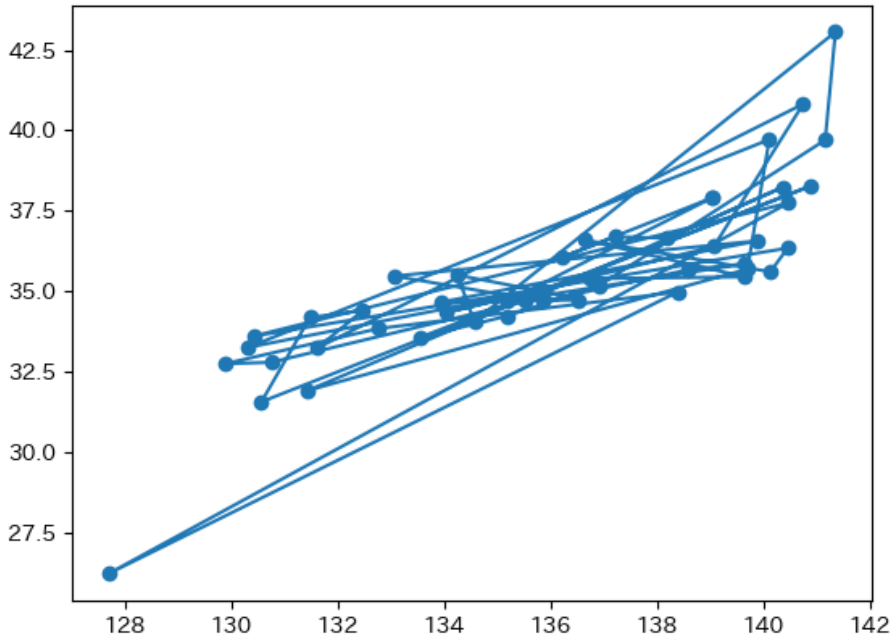
星状に分布する点における初期の距離と、大域的最小値を求めた結果、次のようになった。

初期の距離：239.7481803292344

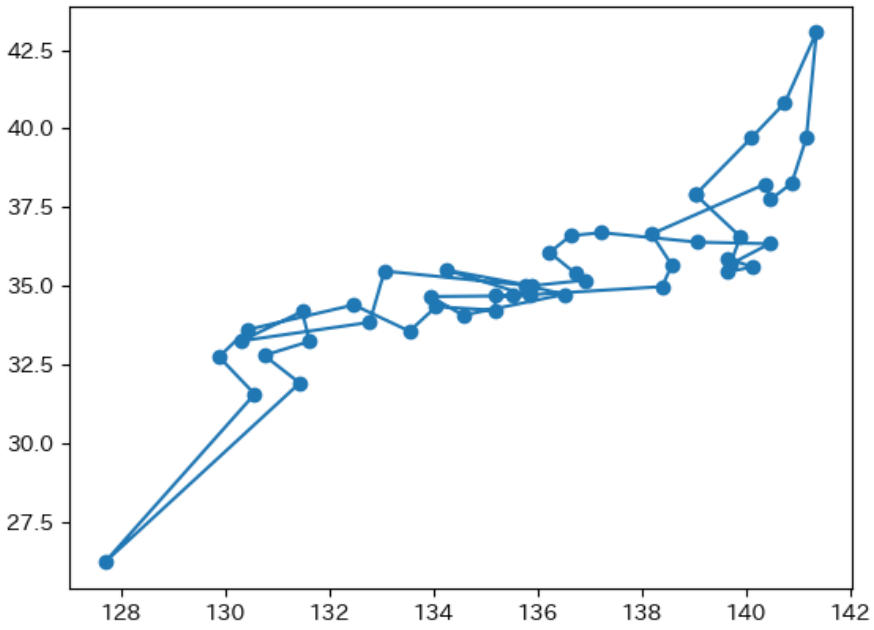
域的最小値：72.98839897084996

また、グラフは次のようになった。

・初期

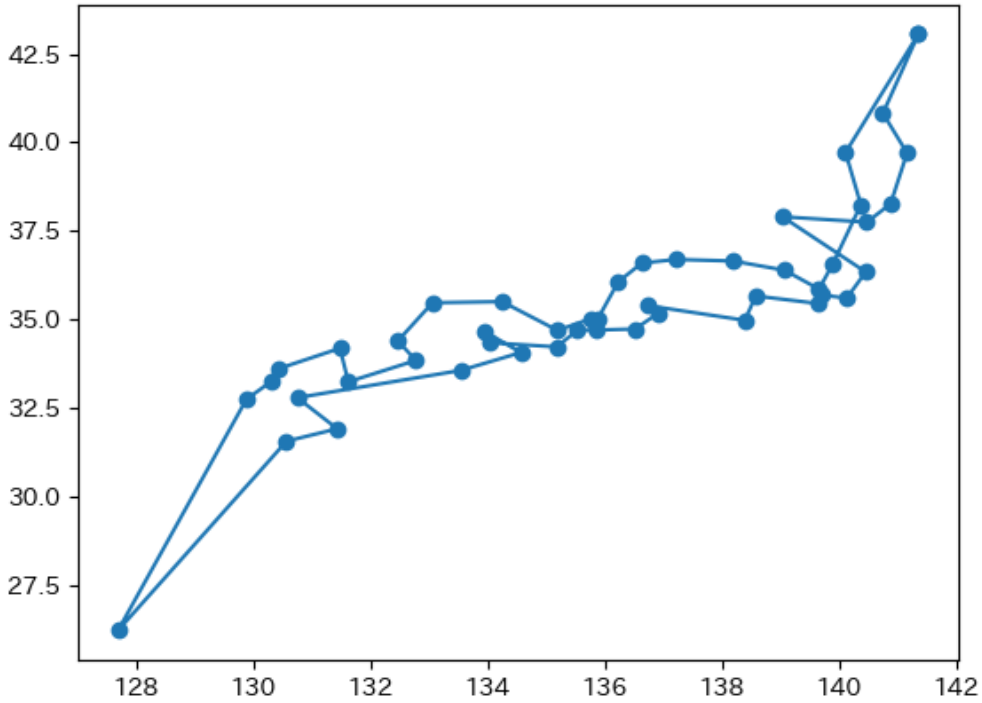


・結果



ここで、点の位置は各県庁所在地の経度と緯度をもとに定め、表示している。

実行した結果、初期の状態より改善はされたが、図を見ると線が交差しているところが数多く存在し、距離が最小となっていないように見える。ここで、さらに摂動回数を 10000 回に増やしてみたところ、次のようになった。



大域的最小値：60.780592530430276

摂動回数が 2500 回のときとくらべ改善されたが、図を見ると、まだ最小値といえそうにないことがわかる。

4 結論

螺旋状に分布する点と、星状に分布する点、県庁所在地の 3 つで焼きなまし法を行った結果、上記のように 1 で示したアルゴリズムでは最適な経路を求めることは難しいが、初期のランダムな経路と比べると改善されることが分かった。また、摂動回数を 4 倍にまで増加させても悪化しうることも分かった。そのため、焼きなまし法が有効となる点の分布の特徴についても考慮する必要がある。