

確率解析特論 I 最終レポート

情報数学班

2024 年 8 月 6 日

目次

1	はじめに	1
2	プログラムの紹介と説明	1
3	結果	2
3.1	温度関数の比較	3
4	摂動の比較	5
5	付録	8

1 はじめに

焼きなまし法を用いて、東京を起点に世界の首都を一度ずつ訪問し、再び東京へと戻る総移動距離の大域最適解を求めた。本レポートでは、使用したプログラムの紹介と説明、結果と考察について述べる。

2 プログラムの紹介と説明

本節では、焼きなまし法において重要な役割を果たす評価関数、温度関数、および焼きなまし法のコードの関連部分を紹介する。コード全体の説明については付録の章で解説する。

温度関数は以下の 2 種類を用いた。

一つ目は $T(t) = c_1 e^{-c_2 t}$ であり、対応するコードは以下の通りである。

```
1 def cooling_function1(T, init_temp, end_temp, start_time, max_time):  
2     res = end_temp + (init_temp - end_temp)*math.exp(-((time() - start_time)/max_time)*5) #c1*  
3     exp(-c2*t) の実装  
     return res
```

二つ目は $T(t) = \frac{c}{\log(t+2)}$ であり、対応するコードは以下の通りである。

```
1 def cooling_function2(T, init_temp, end_temp, start_time, max_time):
2     t = (time() - start_time)/max_time
3     res = end_temp + (init_temp - end_temp) / math.log(t + 2)
4     return res
```

評価関数の説明に先立ち、距離の計算方法について説明する。距離の計算には球面三角法を用いており、これにより円弧上の距離を求めた。コードは以下の通りである。

```
1 def spherical_trigonometry(lat_p, lon_p, lat_q, lon_q):
2     res = np.sin(lat_p) * np.sin(lat_q) + np.cos(lat_p) * np.cos(lon_q - lon_p)
3     res = np.arccos(res)
4     R = 6378 # 地球の半径 (km)
5     res *= R
6
7     return res
```

評価関数では都市の位置を入れ替えた際の移動距離の変化を計算する。この関数では入れ替え前と入れ替え後の距離をそれぞれ計算している。そして、その差を評価基準として用いる。対応するコードは以下の通りである。

```
1 def cost_function(x, y, Countries_list):
2     now_cost, next_cost = 0, 0
3     x_pre_country, x_country, x_next_country = Countries_list[x-1:x+2]
4     y_pre_country, y_country, y_next_country = Countries_list[y-1:y+2]
5     now_cost = spherical_trigonometry(lat[x_pre_country], lon[x_pre_country], lat[x_country],
6         lon[x_country])\
7         + spherical_trigonometry(lat[x_country], lon[x_country], lat[x_next_country], lon[
8             x_next_country])\
9         + spherical_trigonometry(lat[y_pre_country], lon[y_pre_country], lat[y_country], lon[
10             y_country])\
11         + spherical_trigonometry(lat[y_country], lon[y_country], lat[y_next_country], lon[
12             y_next_country])
13     next_cost = spherical_trigonometry(lat[x_pre_country], lon[x_pre_country], lat[y_country],
14         lon[y_country])\
15         + spherical_trigonometry(lat[y_country], lon[y_country], lat[x_next_country], lon[
16             x_next_country])\
17         + spherical_trigonometry(lat[y_pre_country], lon[y_pre_country], lat[x_country], lon[
18             x_country])\
19         + spherical_trigonometry(lat[x_country], lon[x_country], lat[y_next_country], lon[
20             y_next_country])
21     return now_cost, next_cost
```

3 結果

まず、実行方法について説明する。実行時間は2秒と設定し、初期状態から焼きなまし法を適用した。この時点での総距離は1,424,968.4161400648 kmである。なお、地球儀上にプロットした結果は、以下のURLから確認できる。



URL:https://belka-247d8560.github.io/StochasticAnalysis/template/init_root

3.1 温度関数の比較

摂動は、経路から 2 都市をランダムに選び、その 2 都市を入れ替える方法を使用した。この際、異なる温度関数によって結果がどのように変化するかを観察した。

表 1 焼きなまし法を 100 回行った結果

温度関数	平均結果	最小の距離
e^{-t}	463132.01444166445	406531.4913191385
$\frac{1}{\log(t+2)}$	462349.9561414368	394556.62172458577

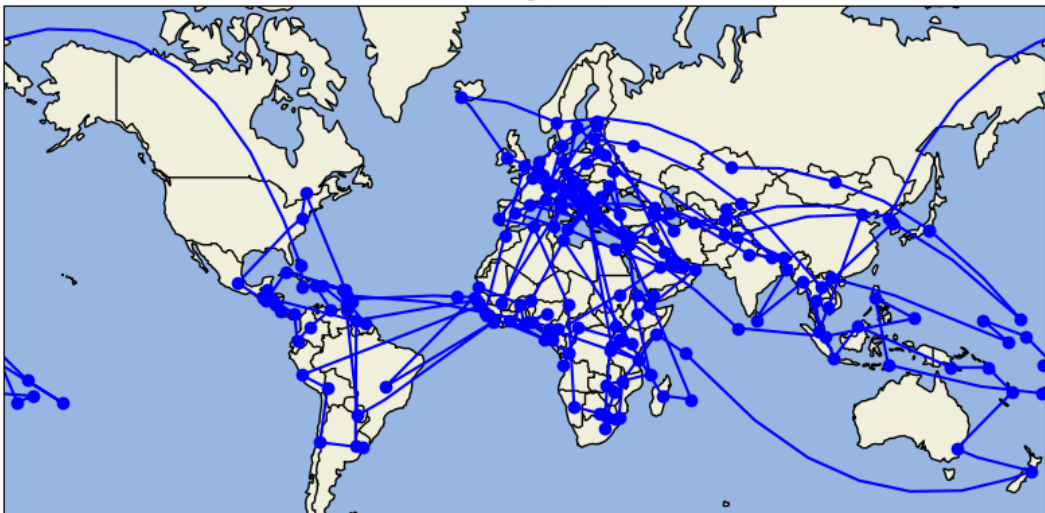
焼きなまし法を各温度関数で 100 回ずつ実行した結果が表 1 に示されている。この結果から、今回の条件では温度関数によって劇的な結果の改善が見られないことがわかった。

温度関数が e^{-t} の場合



URL:<https://belka-247d8560.github.io/StochasticAnalysis/template/temp1>

温度関数が $1/\log(t+2)$ の場合



URL:<https://belka-247d8560.github.io/StochasticAnalysis/template/temp2>

4 摂動の比較

次に、摂動の選び方による結果の違いを検証した。温度関数としては e^{-t} を使用した。摂動の選び方として以下の3つを用いた。

1. 2都市をランダムに選び、位置を入れ替える。
2. 都市とインデックスをランダムに選び、都市をインデックスに挿入する。
3. 2都市をランダムに選び、その間の順路を逆順にする。

表2 焼きなまし法を100回行った結果

摂動	平均結果	最小の距離	摂動回数
1の方法	460294.3393812061	397143.37396547245	40418
2の方法	438229.86759620294	360077.7264521412	52523
3の方法	263529.5690601708	244654.73533840023	75053

摂動の選び方による結果の違いを検証した結果、以下のような傾向が確認された。

1の方法の場合

この方法では、最小距離も他の方法に比べて大きく、探索の効率がやや低いことが示唆される。

2の方法の場合

この方法では、摂動による改善が比較的効果的であり、平均距離と最小距離の両方が方法1よりも短縮されている。摂動回数も多いため、より広範囲に探索が行われたと考えられる。

3の方法の場合

この方法では、最も大きな改善が見られ、平均距離と最小距離がいずれも他の方法に比べて大幅に短縮されている。摂動回数も最も多く、探索が効果的に行われたことがわかる。

総じて、3つの摂動方法の中では「方法3: 2都市間の順路を逆順にする」が最も効果的であり、他の方法と比べて顕著に短い経路を見つけることができた。この結果から、摂動の選び方が焼きなまし法の性能に大きく影響することが確認できた。

2都市の入れ替え



URL:<https://belka-247d8560.github.io/StochasticAnalysis/template/two-city-swap>

挿入による入れ替え



URL:<https://belka-247d8560.github.io/StochasticAnalysis/template/insert-swap>

2都市間の順路を逆順



URL:<https://belka-247d8560.github.io/StochasticAnalysis/template/two-city-reverse>

5 付録

以下に今回使用したコードの全体を記載する。コードは <https://github.com/Belka-247d8560/StochasticAnalysis> にも掲載している。

```
1  """
2  ### ライブラリをインポートする上での注意
3  手元のPython環境次第ではインポートに失敗することがあります。その際はターミナルで
4  ```
5  pip install -r requirements.txt
6  ```
7  と試してください。
8  """
9
10 # 必要なライブラリのインポート
11 import pandas as pd
12 import chardet
13 import numpy as np
14 import math
15 import random
16 from tqdm import tqdm
17 from time import time
18 import matplotlib.pyplot as plt
19 import japanize_matplotlib
20 import plotly.graph_objs as go
21 import plotly.io as pio
22 import cartopy.crs as ccrs
23 import cartopy.feature as cfeature
24
25 #データの読み込み
26 file_path = "capital_cities_2024.csv"
27
28 with open(file_path, 'rb') as f:
29     result = chardet.detect(f.read())
30
31 encoding = result['encoding']
32 df = pd.read_csv(file_path, encoding=encoding)
33 df = df[["ja_ctry", "x", "y"]]
34 df = df.rename(columns={"ja_ctry": "国", "x": "lon", "y": "lat"})
35 #国・地域名をリストとして取得
36 Countries = df["国"].to_list()
37 #日本スタート、日本終了のリストにする。
38 Countries.remove('日本国')
39 Countries = ['日本国'] + Countries + ['日本国']
40 #緯度、経度を辞書として保存(コードの書きやすさのため)
41 lat, lon = dict(), dict()
42 for index, row in df.iterrows():
43     lat[row["国"]] = np.radians(row["lat"])
44     lon[row["国"]] = np.radians(row["lon"])
45
46 #球面三角法の定義
47 def spherical_trigonometry(lat_p, lon_p, lat_q, lon_q):
48     res = np.sin(lat_p) * np.sin(lat_q) + np.cos(lat_p) * np.cos(lat_q) * np.cos(lon_q - lon_p)
49     res = np.arccos(res)
50     R = 6378 # 地球の半径 (km)
51     res *= R
52
53     return res
54
55 #総距離の計算関数の定義
56 def calculating_the_total_distance(Countries_list):
57     N = len(Countries_list)
58     res = 0
59     for i in range(N-1):
60         now_country, next_country = Countries_list[i], Countries_list[i+1]
61         res += spherical_trigonometry(lat[now_country], lon[now_country], lat[next_country], lon
62                                     [next_country])
63     return res
```



```

63
64 #メルカトル図法での結果の描画
65 def mercator_plot(Countries_list, filename):
66     lats = [np.degrees(lat[Country]) for Country in Countries_list]
67     lons = [np.degrees(lon[Country]) for Country in Countries_list]
68     plt.figure(figsize=(10, 6))
69     ax = plt.axes(projection=ccrs.Mercator())
70     ax.add_feature(cfeature.COASTLINE)
71     ax.add_feature(cfeature.BORDERS)
72     ax.add_feature(cfeature.LAND)
73     ax.add_feature(cfeature.OCEAN)
74     plt.plot(lons, lats, marker='o', linestyle='-', color='b', transform=ccrs.Geodetic())
75     plt.xlabel("経度")
76     plt.ylabel("緯度")
77     plt.title(filename)
78     plt.grid(True)
79     plt.show()
80
81 #地球儀上への結果の描画
82 def globe_plot(Countries_list, filename=None):
83     lats = [df.loc[df['国']==Country, 'lat'].values[0] for Country in Countries_list]
84     lons = [df.loc[df['国']==Country, 'lon'].values[0] for Country in Countries_list]
85
86     line = go.Scattergeo(
87         lon=lons,
88         lat=lats,
89         mode='lines+markers',
90         line=dict(color='yellow', width=2),
91         marker=dict(size=8, color='red')
92     )
93
94     layout = go.Layout(
95         geo=dict(
96             projection=dict(type='orthographic'),
97             showcountries=True,
98         ),
99         margin=dict(r=10, l=10, b=10, t=10)
100     )
101
102     fig = go.Figure(data=[line], layout=layout)
103     pio.show(fig)
104
105     if filename:
106         pio.write_html(fig, file = filename)
107
108 #初期値の総距離の計算、描画
109 print("初期値:", *Countries)
110 print("総距離:", calculating_the_total_distance(Countries), "km")
111 #メルカトル図法での初期値の描画
112 mercator_plot(Countries, '初期値の経路')
113 #地球儀上への初期値の描画
114 globe_plot(Countries, 'init_root')
115
116 #温度関数による結果の比較
117 #温度関数の定義
118 def cooling_function1(init_temp, end_temp, start_time, max_time):
119     T_max = math.exp(0)
120     T_min = math.exp(-max_time)
121     T = math.exp(-(time() - start_time))
122     res = end_temp + (init_temp - end_temp)*((T - T_min) / (T_max - T_min))
123     return res
124
125 def cooling_function2(init_temp, end_temp, start_time, max_time):
126     T_max = 1/math.log(2)
127     T_min = 1/math.log(max_time + 2)
128     T = 1/math.log((time() - start_time) + 2)
129     res = end_temp + (init_temp - end_temp)*((T - T_min) / (T_max - T_min))
130     return res
131
132 #評価関数
133 def cost_function(x, y, Countries_list):
134     now_cost, next_cost = 0, 0
135     x_pre_country, x_country, x_next_country = Countries_list[x-1:x+2]
136     y_pre_country, y_country, y_next_country = Countries_list[y-1:y+2]

```

```

136     now_cost = spherical_trigonometry(lat[x_pre_country], lon[x_pre_country], lat[x_country],
137         lon[x_country])\
138         + spherical_trigonometry(lat[x_country], lon[x_country], lat[x_next_country], lon[
139             x_next_country])\
140         + spherical_trigonometry(lat[y_pre_country], lon[y_pre_country], lat[y_country], lon
141             [y_country])\
142         + spherical_trigonometry(lat[y_country], lon[y_country], lat[y_next_country], lon[
143             y_next_country])
144     next_cost = spherical_trigonometry(lat[x_pre_country], lon[x_pre_country], lat[y_country],
145         lon[y_country])\
146         + spherical_trigonometry(lat[y_country], lon[y_country], lat[x_next_country], lon[
147             x_next_country])\
148         + spherical_trigonometry(lat[y_pre_country], lon[y_pre_country], lat[x_country], lon
149             [x_country])\
150         + spherical_trigonometry(lat[x_country], lon[x_country], lat[y_next_country], lon[
151             y_next_country])
152     return now_cost, next_cost
153
154 #焼きなまし法
155 def simulated_annealing(Countries_list, cooling_function, cost_function, init_temp, end_temp,
156     start_time, max_time):
157     T = init_temp
158     while time() - start_time <= max_time:
159         a, b = random.choices(range(1, len(Countries_list)-1), k=2)
160         now_cost, next_cost = cost_function(a, b, Countries_list)
161         if next_cost < now_cost or random.random() < np.exp(-np.abs(now_cost - next_cost)/T):
162             Countries_list[a], Countries_list[b] = Countries_list[b], Countries_list[a]
163         T = cooling_function(init_temp, end_temp, start_time, max_time)
164     return Countries_list
165
166 #1つ目の温度関数で焼きなまし法を100回行う
167 min_result = []
168 min_distance = float('inf')
169 average_distance = 0
170 for i in tqdm(range(100)):
171     result = simulated_annealing(Countries_list=Countries.copy(), cooling_function=
172         cooling_function1, cost_function=cost_function, init_temp=10000, end_temp=0.001,
173         start_time=time(), max_time=2)
174     distance = calculating_the_total_distance(result)
175     if distance < min_distance:
176         min_distance = distance
177         min_result = result.copy()
178     average_distance += distance
179 print("最小の距離:", min_distance)
180 print("最小の距離での巡回順番:", *min_result)
181 print("平均:", average_distance/100)
182
183 #メルカトル図法での最小の結果の描画
184 mercator_plot(min_result, '温度関数が$e^{-t}$の場合')
185 #地球儀上への最小の結果の描画
186 globe_plot(Countries, 'temp1')
187
188 #2つ目の温度関数で焼きなまし法を100回行う
189 min_result = []
190 min_distance = float('inf')
191 average_distance = 0
192 for i in tqdm(range(100)):
193     result = simulated_annealing(Countries_list=Countries.copy(), cooling_function=
194         cooling_function2, cost_function=cost_function, init_temp=10000, end_temp=0.001,
195         start_time=time(), max_time=2)
196     distance = calculating_the_total_distance(result)
197     if distance < min_distance:
198         min_distance = distance
199         min_result = result.copy()
200     average_distance += distance
201 print("最小の距離:", min_distance)
202 print("最小の距離での巡回順番:", *min_result)
203 print("平均:", average_distance/100)
204
205 #メルカトル図法での最小の結果の描画
206 mercator_plot(min_result, '温度関数が$1/\log(t+2)$の場合')
207 #地球儀上への最小の結果の描画
208 globe_plot(Countries, 'temp2')

```

```

196
197 # 摂動による結果の比較
198 def cooling_function(init_temp, end_temp, start_time, max_time):
199     T_max = math.exp(0)
200     T_min = math.exp(-max_time)
201     T = math.exp(-(time() - start_time))
202     res = end_temp + (init_temp - end_temp)*((T - T_min) / (T_max - T_min))
203     return res
204
205 ### 2都市をランダムに選び、入れ替える方法の場合
206 def cost_function(x, y, Countries_list):
207     now_cost, next_cost = 0, 0
208     x_pre_country, x_country, x_next_country = Countries_list[x-1:x+2]
209     y_pre_country, y_country, y_next_country = Countries_list[y-1:y+2]
210     now_cost = spherical_trigonometry(lat[x_pre_country], lon[x_pre_country], lat[x_country],
211                                     lon[x_country])\
212         + spherical_trigonometry(lat[x_country], lon[x_country], lat[x_next_country], lon[
213                                     x_next_country])\
214         + spherical_trigonometry(lat[y_pre_country], lon[y_pre_country], lat[y_country], lon[
215                                     y_country])\
216         + spherical_trigonometry(lat[y_country], lon[y_country], lat[y_next_country], lon[
217                                     y_next_country])
218     next_cost = spherical_trigonometry(lat[x_pre_country], lon[x_pre_country], lat[y_country],
219                                     lon[y_country])\
220         + spherical_trigonometry(lat[y_country], lon[y_country], lat[x_next_country], lon[
221                                     x_next_country])\
222         + spherical_trigonometry(lat[y_pre_country], lon[y_pre_country], lat[x_country], lon[
223                                     x_country])\
224         + spherical_trigonometry(lat[x_country], lon[x_country], lat[y_next_country], lon[
225                                     y_next_country])
226     return now_cost, next_cost
227
228 def simulated_annealing(Countries_list, cooling_function, cost_function, init_temp, end_temp,
229                         start_time, max_time):
230     T = init_temp
231     # cnt = 0
232     while time() - start_time <= max_time:
233         # cnt += 1
234         a, b = random.choices(range(1, len(Countries_list)-1), k=2)
235         now_cost, next_cost = cost_function(a, b, Countries_list)
236         if next_cost < now_cost or random.random() < np.exp(-np.abs(now_cost - next_cost)/T):
237             Countries_list[a], Countries_list[b] = Countries_list[b], Countries_list[a]
238             T = cooling_function(init_temp, end_temp, start_time, max_time)
239         # print('摂動回数:', cnt)
240     return Countries_list
241
242 #焼きなまし法を100回行う
243 min_result = []
244 min_distance = float('inf')
245 average_distance = 0
246 for i in tqdm(range(100)):
247     result = simulated_annealing(Countries_list=Countries.copy(), cooling_function=
248                                 cooling_function, cost_function=cost_function, init_temp=10000, end_temp=0.001,
249                                 start_time=time(), max_time=2)
250     distance = calculating_the_total_distance(result)
251     if distance < min_distance:
252         min_distance = distance
253         min_result = result.copy()
254     average_distance += distance
255 print("最小の距離:", min_distance)
256 print("最小の距離での巡回順番:", *min_result)
257 print("平均:", average_distance/100)
258
259 #メルカトル図法での最小の結果の描画
260 mercator_plot(min_result, '2都市の入れ替え')
261 #地球儀上への最小の結果の描画
262 globe_plot(Countries, 'two-city-swap')
263
264 #挿入での方法の場合
265 def cost_function(x, y, Countries_list):
266     now_cost, next_cost = 0, 0
267     x_pre_country, x_country, x_next_country = Countries_list[x-1:x+2]
268     y_pre_country, y_next_country = Countries_list[y:y+2]

```

```

258     now_cost = spherical_trigonometry(lat[x_pre_country], lon[x_pre_country], lat[x_country],
259         lon[x_country])\
260         + spherical_trigonometry(lat[x_country], lon[x_country], lat[x_next_country], lon[
261             x_next_country])\
262         + spherical_trigonometry(lat[y_pre_country], lon[y_pre_country], lat[y_next_country
263             ], lon[y_next_country])
264     Countries_list.pop(x)
265     Countries_list.insert(y, x_country)
266     next_cost = spherical_trigonometry(lat[x_pre_country], lon[x_pre_country], lat[
267         x_next_country], lon[x_next_country])\
268         + spherical_trigonometry(lat[y_pre_country], lon[y_pre_country], lat[x_country], lon
269             [x_country])\
270         + spherical_trigonometry(lat[x_country], lon[x_country], lat[y_next_country], lon[
271             y_next_country])
272     return now_cost, next_cost
273
274 def simulated_annealing(Countries_list, cooling_function, cost_function, init_temp, end_temp,
275     start_time, max_time):
276     T = init_temp
277     # cnt = 0
278     while time() - start_time <= max_time:
279         # cnt += 1
280         a, b = random.choices(range(1, len(Countries_list)-1), k=2)
281         now_cost, next_cost = cost_function(a, b, Countries_list)
282         if not (next_cost < now_cost or random.random() < np.exp(-np.abs(now_cost - next_cost)/T
283             )):
284             Country = Countries_list.pop(b)
285             Countries_list.insert(a, Country)
286             T = cooling_function(init_temp, end_temp, start_time, max_time)
287         # print('摂動回数:', cnt)
288         return Countries_list
289
290 #焼きなまし法を100回行う
291 min_result = []
292 min_distance = float('inf')
293 average_distance = 0
294 for i in tqdm(range(100)):
295     result = simulated_annealing(Countries_list=Countries.copy(), cooling_function=
296         cooling_function, cost_function=cost_function, init_temp=10000, end_temp=0.001,
297         start_time=time(), max_time=2)
298     distance = calculating_the_total_distance(result)
299     if distance < min_distance:
300         min_distance = distance
301         min_result = result.copy()
302     average_distance += distance
303 print("最小の距離:", min_distance)
304 print("最小の距離での巡回順番:", *min_result)
305 print("平均:", average_distance/100)
306
307 #メルカトル図法での最小の結果の描画
308 mercator_plot(min_result, '挿入による入れ替え')
309 #地球儀上への最小の結果の描画
310 globe_plot(Countries, 'insert-swap')
311
312 #順路を逆順にする方法の場合
313 def cost_function(x, y, Countries_list):
314     now_cost, next_cost = 0, 0
315     x_pre_country, x_country = Countries_list[x-1:x+1]
316     y_country, y_next_country = Countries_list[y:y+2]
317     now_cost = spherical_trigonometry(lat[x_pre_country], lon[x_pre_country], lat[x_country],
318         lon[x_country])\
319         + spherical_trigonometry(lat[y_country], lon[y_country], lat[y_next_country], lon[
320             y_next_country])
321     next_cost = spherical_trigonometry(lat[x_pre_country], lon[x_pre_country], lat[y_country],
322         lon[y_country])\
323         + spherical_trigonometry(lat[x_country], lon[x_country], lat[y_next_country], lon[
324             y_next_country])
325     return now_cost, next_cost
326
327 def simulated_annealing(Countries_list, cooling_function, cost_function, init_temp, end_temp,
328     start_time, max_time):
329     T = init_temp
330     # cnt = 0

```

```

316     while time() - start_time <= max_time:
317         # cnt += 1
318         a, b = random.choices(range(1, len(Countries_list)-1), k=2)
319         now_cost, next_cost = cost_function(a, b, Countries_list)
320         if next_cost < now_cost or random.random() < np.exp(-np.abs(now_cost - next_cost)/T):
321             Countries_list[a:b+1] = Countries_list[a:b+1][::-1]
322             T = cooling_function(init_temp, end_temp, start_time, max_time)
323         # print('摂動回数:', cnt)
324         return Countries_list
325
326 #焼きなまし法を100回行う
327 min_result = []
328 min_distance = float('inf')
329 average_distance = 0
330 for i in tqdm(range(100)):
331     result = simulated_annealing(Countries_list=Countries.copy(), cooling_function=
332         cooling_function, cost_function=cost_function, init_temp=10000, end_temp=0.001,
333         start_time=time(), max_time=2)
334     distance = calculating_the_total_distance(result)
335     if distance < min_distance:
336         min_distance = distance
337         min_result = result.copy()
338     average_distance += distance
339 print("最小の距離:", min_distance)
340 print("最小の距離での巡回順番:", *min_result)
341 print("平均:", average_distance/100)
342
343 #メルカトル図法での最小の結果の描画
344 mercator_plot(min_result, '2都市間の順路を逆順')
345 #地球儀上への最小の結果の描画
346 globe_plot(Countries, 'two-city-reverse')

```