Traveling Salesman Problem

The traveling salesman problem, or TSP, is a generalization of the sorting problem. In the TSP, we are given a set \( \{c_1, c_2, \ldots, c_N\} \) of cities and for each pair \( \{c_i, c_j\} \) of distinct cities a distance \( d(c_i, c_j) \). Our goal is to find an ordering \( \pi \) of the cities that minimizes the quantity

\[
\sum_{i=1}^{N-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(N)}, c_{\pi(1)})
\]

This quantity is referred to as the tour length, since it is the length of the tour a salesman would make when visiting the cities in the order specified by the permutation, returning at the end to the initial city. TSP is called symmetric if the distances satisfy \( d(c_i, c_j) = d(c_j, c_i) \) for \( 1 \leq i, j \leq N \).

As an illustration, the following is an example of TSP ordering of American cities.

The symmetric traveling salesman problem has many applications, from VLSI chip fabrication to X-ray crystallography, and a long history. It is “intractable” because any algorithm for finding optimal tours must have a worst-case running time that grows faster than any polynomial. This gives rise to many heuristic algorithms that find near-optimal tours quickly. In this lab, you will have to implement a simple heuristic, called the Nearest Neighbor Algorithm.

**Nearest Neighbor Algorithm:**

We construct an ordering \( \pi(1), \ldots, \pi(N) \) of the cities, such that the initial \( \pi(1) \) is chosen as a random integer between 1 and \( N \), while every other \( \pi(i), i = 2, \ldots, N \), is chosen by the following rule:

- Among the cities whose index is not already selected to ordering \( \{\pi(1), \pi(2), \ldots, \pi(i-1)\} \), find the city \( c_k \) with the closest distance to \( c_{\pi(i-1)} \) and set \( \pi(i) = k \).

The resulting tour traverses the cities in the order defined by \( c_{\pi(1)}, \ldots, c_{\pi(N)} \).
**TASKS**

1. **(2 points) Write the pseudocode** for the Nearest Neighbor Algorithm. Assume that each city $c_i$ is represented by its location $(x_i, y_i)$ and that the distance $d(c_i, c_j)$ is simple Euclidean distance calculated as

   $$d(c_i, c_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

   Input to the algorithm is set of pairs $S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$ and its output is ordering $\pi = \pi(1), \pi(2), \ldots, \pi(N)$.

2. **(2 points) Estimate** the runtime $T(N)$ of the Nearest Neighbor Algorithm.

3. **(4 points) Implement** the Nearest Neighbor Algorithm in Matlab. You should do it by writing the function:

   ```matlab
   function ordering = NN_TSP(cities)
   ``

   that takes `cities` which is a matrix of dimension $N \times 2$ as input (where each row represents a city and columns are $x$ and $y$ coordinates of its location) and produces `ordering` as output. **Submit** the Matlab code you wrote.

4. **(2 points) Test** the algorithm on $N = 10, 100, 1000$ randomly generated cities. You can generate $N$ random cities as

   ```matlab
   cities = rand(N, 2);
   ``

   You can plot the original ordering (path that traverses all cities in order from city 1 to city $N$) as

   ```matlab
   plot(cities(:,1),cities(:,2))
   ``

   You can plot the resulting ordering of cities as

   ```matlab
   plot(cities(ordering,1),cities(ordering,2))
   ``

   Provide the plots of the original ordering and ordering after the Nearest Neighbor Algorithm for each $N$. List the actual run time for each $N$. **Discuss** how successful is the resulting ordering. Do you see some imperfections in the solution (places where you think the algorithm could have produced better solution)?