Math 126 Problem 13: Point Reconstruction from Distances

Spring 2027

Submission Instructions

Please upload your work to Gradescope by **11:59pm on Monday, March 22, 2027 (Pacific time).**

- You are encouraged to discuss ideas and work with your classmates. However, you *must acknowledge* your collaborators at the top of each solution on which you collaborated with others and you *must write* your solutions independently.
- Your solutions to theory questions must be written legibly, or typeset in LaTeX or markdown.
- I recommend that you write your solutions in a Python notebook using Google Colab.
- You should submit your solutions as a *single PDF* via the assignment on Gradescope.

Grading: The point of the problem set is for *you* to learn. To this end, I hope to disincentivize the use of LLMs by *not* grading your work for correctness. Instead, you will grade your own work by comparing it to my solutions. This self-grade is due the Friday *after* the problem set is due, also on Gradescope.
Original notebook: Google Colab.

Point Reconstruction from Distances

Suppose you are given all pairwise distances between a set of points $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$. You can assume that $d \ll n$. Let $\mathbf{D} \in \mathbb{R}^{n \times n}$ be the distance matrix with $\mathbf{D}_{i,j} = \|\mathbf{x}_i - \mathbf{x}_j\|_2^2$. You would like to recover the location of the original points, at least up to possible rotations and translations which do not change pairwise distances.

Assume that $\sum_{i=1}^n \mathbf{x}_i = \mathbf{0}$, i.e., the points are centered at the origin.

Part A

Describe a polynomial time algorithm for learning $\sum_{i=1}^n \|\mathbf{x}_i\|_2^2$.

Hint: Expand $\|\mathbf{x}_i - \mathbf{x}_j\|_2^2$.

Part B

Inspired by the above approach, describe an efficient algorithm for learning $\|\mathbf{x}_i\|_2^2$ for each i.

Part C

Describe an algorithm for recovering a set of points $\mathbf{x}_1, \ldots, \mathbf{x}_n$ which realize the distances in $\mathbf{D}$.

Hint: This is where you will use the SVD! It might help to prove that $\mathbf{D}$ has rank $\leq d + 2$.

Part D

Implement your algorithm and run it on the U.S. cities dataset below. Note that the distances in the file are unsquared Euclidean distances, so you need to square them to obtain $\mathbf{D}$. Plot your estimated city locations on a 2D plot and label the cities to make it clear how the plot is oriented

Starter Code and Data

```
import numpy as np

distances = {
  'Boston' : [0,400,851,1551,1769,1605,2596,1137,1255,1123,188,1282,271,2300,483,1038,2099,2699,2493,393],
  'Buffalo' : [400,0,454,1198,1370,1286,2198,803,1181,731,292,883,279,1906,178,662,1699,2300,2117,292],
  'Chicago' : [851,454,0,803,920,940,1745,482,1188,355,713,432,666,1453,410,262,1260,1858,1737,597],
  'Dallas' : [1551,1198,803,0,663,225,1240,420,1111,862,1374,586,1299,887,1070,547,999,1483,1681,1185],
  'Denver' : [1769,1370,920,663,0,879,831,879,1726,700,1631,488,1579,586,1320,796,371,949,1021,1494],
  'Houston' : [1605,1286,940,225,879,0,1374,484,968,1056,1420,794,1341,1017,1137,679,1200,1645,1891,1220],
  'Los Angeles' : [2596,2198,1745,1240,831,1374,0,1603,2339,1524,2451,1315,2394,357,2136,1589,579,347,959,2300],
  'Memphis' : [1137,803,482,420,879,484,1603,0,872,699,957,529,881,1263,660,240,1250,1802,1867,765],
  'Miami' : [1255,1181,1188,1111,1726,968,2339,872,0,1511,1092,1397,1019,1982,1010,1061,2089,2594,2734,923],
  'Minneapolis' : [1123,731,355,862,700,1056,1524,699,1511,0,1018,290,985,1280,743,466,987,1584,1395,934],
  'New York' : [188,292,713,1374,1631,1420,2451,957,1092,1018,0,1144,83,2145,317,875,1972,2571,2408,230],
  'Omaha' : [1282,883,432,586,488,794,1315,529,1397,290,1144,0,1094,1036,836,354,833,1429,1369,1014],
  'Philadelphia' : [271,279,666,1299,1579,1341,2394,881,1019,985,83,1094,0,2083,259,811,1925,2523,2380,123],
  'Phoenix' : [2300,1906,1453,887,586,1017,357,1263,1982,1280,2145,1036,2083,0,1828,1272,504,653,1114,1973],
  'Pittsburgh' : [483,178,410,1070,1320,1137,2136,660,1010,743,317,836,259,1828,0,559,1668,2264,2138,192],
  'Saint,Louis' : [1038,662,262,547,796,679,1589,240,1061,466,875,354,811,1272,559,0,1162,1744,1724,712],
  'Salt Lake,City' : [2099,1699,1260,999,371,1200,579,1250,2089,987,1972,833,1925,504,1668,1162,0,600,701,1848],
  'San Francisco' : [2699,2300,1858,1483,949,1645,347,1802,2594,1584,2571,1429,2523,653,2264,1744,600,0,678,2442],
  'Seattle' : [2493,2117,1737,1681,1021,1891,959,1867,2734,1395,2408,1369,2380,1114,2138,1724,701,678,0,2329],
  'Washington,D.C.' : [393,292,597,1185,1494,1220,2300,765,923,934,230,1014,123,1973,192,712,1848,2442,2329,0]
 }

cities = list(distances.keys())
D_squareroot = np.array([distances[city] for city in cities])
# Square entries of D
D = np.square(D_squareroot)
```