Some New Thoughts on Distance Geometry Problem: Error Accumulation, Laplacian and Stress Majorization

Zhenli Sheng
szl@lsec.cc.ac.cn

Institute of Computational Mathematics and Scientific/Engineering Computing,
Academy of Mathematics and Systems Science,
Chinese Academy of Sciences

Sept 16, 2014
seminar talk

Outline

Outline

the Problem

Distance Geometry Problem (DGP)

For a graph $G = (V, E)$, given a distance matrix D (L and U, respectively) on E and an integer d, find $x_1, x_2, \ldots, x_n \in \mathbb{R}^d$, such that

$$\|x_i - x_j\| = d_{ij}, \quad (i,j) \in E. \tag{1}$$

or

$$l_{ij} \leq \|x_i - x_j\| \leq u_{ij}, \quad (i,j) \in E. \tag{2}$$

the Problem

Distance Geometry Problem (DGP)

For a graph $G = (V, E)$, given a distance matrix D (L and U, respectively) on E and an integer d, find $x_1, x_2, \ldots, x_n \in \mathbb{R}^d$, such that

$$\|x_i - x_j\| = d_{ij}, \quad (i,j) \in E. \tag{1}$$

or

$$l_{ij} \leq \|x_i - x_j\| \leq u_{ij}, \quad (i,j) \in E. \tag{2}$$

- NP-hard in general

the Problem

Distance Geometry Problem (DGP)

For a graph $G = (V, E)$, given a distance matrix D (L and U, respectively) on E and an integer d, find $x_1, x_2, \ldots, x_n \in \mathbb{R}^d$, such that

$$\|x_i - x_j\| = d_{ij}, \quad (i,j) \in E. \tag{1}$$

or

$$l_{ij} \leq \|x_i - x_j\| \leq u_{ij}, \quad (i,j) \in E. \tag{2}$$

- NP-hard in general
  - polynomial-time solvable instances: K-trilateration graph

the Problem

Distance Geometry Problem (DGP)

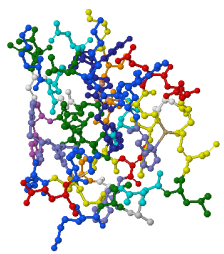
For a graph $G = (V, E)$, given a distance matrix D (L and U, respectively) on E and an integer d, find $x_1, x_2, \ldots, x_n \in \mathbb{R}^d$, such that

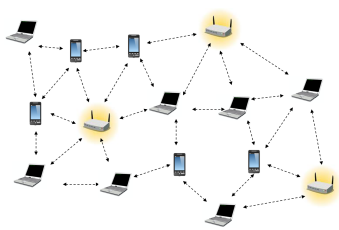
$$\|x_i - x_j\| = d_{ij}, \quad (i,j) \in E. \tag{1}$$

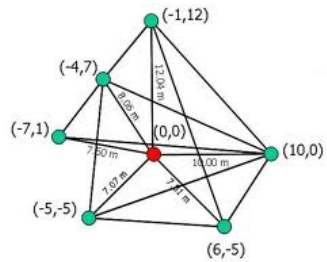
or

$$l_{ij} \leq \|x_i - x_j\| \leq u_{ij}, \quad (i,j) \in E. \tag{2}$$

- NP-hard in general
  - polynomial-time solvable instances: K-trilateration graph
- Applications: protein structure determination, sensor network localization, graph drawing, multidimensional scaling, etc







the Problem

Distance Geometry Problem (DGP)

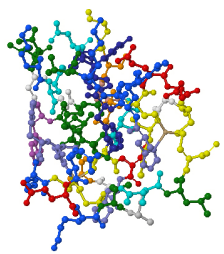
For a graph $G = (V, E)$, given a distance matrix D (L and U, respectively) on E and an integer d, find $x_1, x_2, \ldots, x_n \in \mathbb{R}^d$, such that

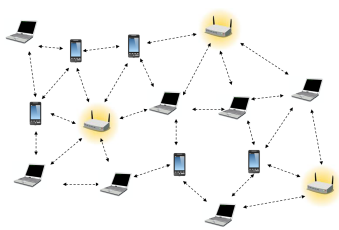
$$\|x_i - x_j\| = d_{ij}, \quad (i,j) \in E. \tag{1}$$

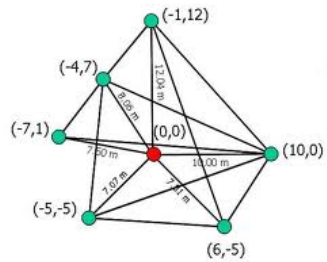
or

$$l_{ij} \leq \|x_i - x_j\| \leq u_{ij}, \quad (i,j) \in E. \tag{2}$$

- NP-hard in general
  - polynomial-time solvable instances: K-trilateration graph
- Applications: protein structure determination, sensor network localization, graph drawing, multidimensional scaling, etc







- In most instances, distances are **local**, **sparse** and **noisy**.

Solution Methods

- Matrix Decomposition Method (Blumenthal 1953, Torgerson,1958)
- The Embedding Algorithm (Crippen-Havel, 1988)
- Global Smoothing Algorithm (Moré-Wu, 1997)
- **Geometric Buildup Method** (Dong-Wu, 2002, Sit-Wu-Yuan, 2009)
- SDP Relaxation Method (Biswas-Toh-Ye, 2007)
- Nearest Euclidean Distance Matrix (Qi-Yuan, 2013)
- The Branch-and-Prune Algorithm (Liberti-Lavor-Maculan-Mucherino, 2014)
- and so on ...

Models: error functions

The problem is usually modeled as an unconstrained **global** optimization problem.

- Stress function

$$Stress(x_1, x_2, \ldots, x_n) = \sum_{(i,j)\in E} \omega_{ij}(\|x_i - x_j\| - d_{ij})^2 \tag{3}$$

Models: error functions

The problem is usually modeled as an unconstrained **global** optimization problem.

- Stress function

$$Stress(x_1, x_2, \ldots, x_n) = \sum_{(i,j)\in E} \omega_{ij}(\|x_i - x_j\| - d_{ij})^2 \tag{3}$$

  - ω_{ij} specify the degree of confidence or simply weights to balance all the terms
  - $\omega_{ij} = 1$: raw stress function
  - $\omega_{ij} = 1/d_{ij}^2$: relative errors

Models: error functions

The problem is usually modeled as an unconstrained **global** optimization problem.

- Stress function

$$Stress(x_1, x_2, \ldots, x_n) = \sum_{(i,j)\in E} \omega_{ij}(\|x_i - x_j\| - d_{ij})^2 \tag{3}$$

  - ω_{ij} specify the degree of confidence or simply weights to balance all the terms
  - $\omega_{ij} = 1$: raw stress function
  - $\omega_{ij} = 1/d_{ij}^2$: relative errors

- Smoothed Stress function

$$SStress(x_1, x_2, \ldots, x_n) = \sum_{(i,j)\in E} \omega_{ij}(\|x_i - x_j\|^2 - d_{ij}^2)^2 \tag{4}$$

Models: error functions

The problem is usually modeled as an unconstrained **global** optimization problem.

- Stress function

$$Stress(x_1, x_2, \ldots, x_n) = \sum_{(i,j)\in E} \omega_{ij}(\|x_i - x_j\| - d_{ij})^2 \tag{3}$$

  - ω_{ij} specify the degree of confidence or simply weights to balance all the terms
  - $\omega_{ij} = 1$: raw stress function
  - $\omega_{ij} = 1/d_{ij}^2$: relative errors

- Smoothed Stress function

$$SStress(x_1, x_2, \ldots, x_n) = \sum_{(i,j)\in E} \omega_{ij}(\|x_i - x_j\|^2 - d_{ij}^2)^2 \tag{4}$$

- Absolute Error function

$$AbsErr(x_1, x_2, \ldots, x_n) = \sum_{(i,j)\in E} \omega_{ij} \left|\|x_i - x_j\|^2 - d_{ij}^2\right| \tag{5}$$

Models: error functions

The problem is usually modeled as an unconstrained **global** optimization problem.

- Stress function

$$Stress(x_1, x_2, \ldots, x_n) = \sum_{(i,j)\in E} \omega_{ij}(\|x_i - x_j\| - d_{ij})^2 \tag{3}$$

  - ω_{ij} specify the degree of confidence or simply weights to balance all the terms
  - $\omega_{ij} = 1$: raw stress function
  - $\omega_{ij} = 1/d_{ij}^2$: relative errors

- Smoothed Stress function

$$SStress(x_1, x_2, \ldots, x_n) = \sum_{(i,j)\in E} \omega_{ij}(\|x_i - x_j\|^2 - d_{ij}^2)^2 \tag{4}$$

- Absolute Error function

$$AbsErr(x_1, x_2, \ldots, x_n) = \sum_{(i,j)\in E} \omega_{ij} \left|\|x_i - x_j\|^2 - d_{ij}^2\right| \tag{5}$$

- our proposed function

$$f(x_1, x_2, \ldots, x_n) = \sum_{(i,j)\in E} \omega_{ij} h(\frac{\|x_i - x_j\|}{d_{ij}}), \quad h(x) = \begin{cases} \frac{1}{2}(x-1)^2 & x \geq 1 \\ x - 1 - ln(x) & x < 1 \end{cases} \tag{6}$$

Geometric Buildup Method

Key ideas:

- Observation: a point in $\mathbb{R}^d$ can be uniquely determined by $d+1$ distances to the determined points (not in a $d-1$ dimensional hyperplane)
- Determine the points one by one

Geometric Buildup Method

Key ideas:

- Observation: a point in $\mathbb{R}^d$ can be uniquely determined by $d+1$ distances to the determined points (not in a $d-1$ dimensional hyperplane)
- Determine the points one by one

Algorithm 2: Geometric Buildup Method for sparse noisy anchor-free DGP

Input: A symmetric distance matrix D to specify part of the pairwise distances
Output: the set of determined points and the corresponding coordinates X

1 Find four points that are not in the same plane
2 Determined the positions of the four points with the distance among them
3 Repeat:
for *each of the undetermined points* **do**
4 | **if** *the point has* l ($l \geq 4$) *distances to determined points that are not in the same plane* **then**
5 | | Determine the position(s) by linear (nonlinear) least square
6 If no more points can be determined in the loop, stop.

Motivation: a short story

When I was an undergraduate student, I read the papers about Geometric Buildup Method and wrote `MATLAB` code to implement it, I found that

Motivation: a short story

When I was an undergraduate student, I read the papers about Geometric Buildup Method and wrote `MATLAB` code to implement it, I found that

Error Accumulation is a DEVIL!

Motivation: a short story

When I was an undergraduate student, I read the papers about Geometric Buildup Method and wrote `MATLAB` code to implement it, I found that

Error Accumulation is a DEVIL!

I came up with two ideas:

- design strategies to prevent error accumulation

Motivation: a short story

When I was an undergraduate student, I read the papers about Geometric Buildup Method and wrote `MATLAB` code to implement it, I found that

Error Accumulation is a DEVIL!

I came up with two ideas:

- design strategies to prevent error accumulation (linear/nonlinear least square)

Motivation: a short story

When I was an undergraduate student, I read the papers about Geometric Buildup Method and wrote `MATLAB` code to implement it, I found that

Error Accumulation is a DEVIL!

I came up with two ideas:

- design strategies to prevent error accumulation (linear/nonlinear least square)
- divide and conquer

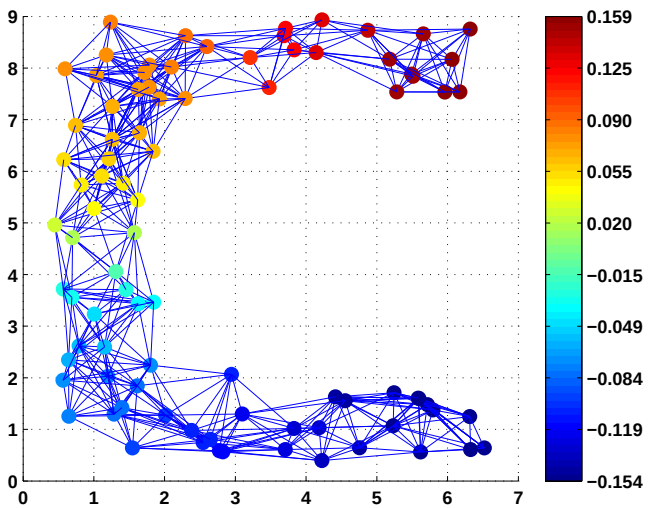
Motivation: a short story

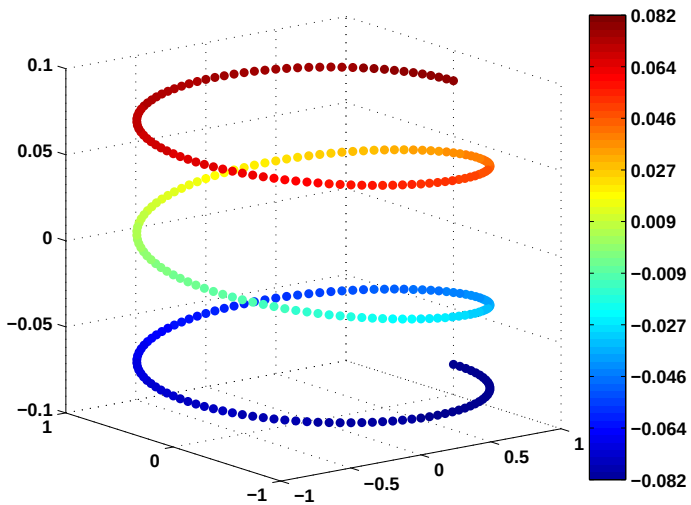
When I was an undergraduate student, I read the papers about Geometric Buildup Method and wrote `MATLAB` code to implement it, I found that

Error Accumulation is a DEVIL!

I came up with two ideas:

- design strategies to prevent error accumulation (linear/nonlinear least square)
- divide and conquer (Laplacian matrix)





Outline

2. **Error Accumulation**
   - analysis of error accumulation
   - our improvement

Why Error Accumulation matters?

small errors + iterative(sequential) algorithm = **error accumulation**

Why Error Accumulation matters?

small errors + iterative(sequential) algorithm = **error accumulation**

A simple example in computational mathematics:
Define

$$doubling(x) = \begin{cases} 2x & 0 \leq x \leq 1/2 \\ 2x - 1 & 1/2 < x \leq 1 \end{cases} \tag{7}$$

Iterate: $x_{k+1} = doubling(x_k)$. If starts at $x_0 = 0.4$, it should cycle in this way: $0.4 \to 0.8 \to 0.6 \to 0.2 \to 0.4 \to \cdots$

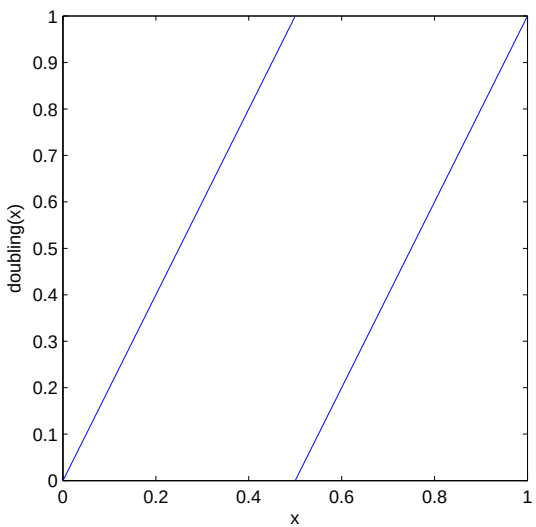
Why Error Accumulation matters?

small errors + iterative(sequential) algorithm = **error accumulation**

A simple example in computational mathematics:
Define

$$doubling(x) = \begin{cases} 2x & 0 \le x \le 1/2 \\ 2x - 1 & 1/2 < x \le 1 \end{cases} \tag{7}$$

Iterate: $x_{k+1} = doubling(x_k)$. If starts at $x_0 = 0.4$, it should cycle in this way: $0.4 \to 0.8 \to 0.6 \to 0.2 \to 0.4 \to \cdots$ However,



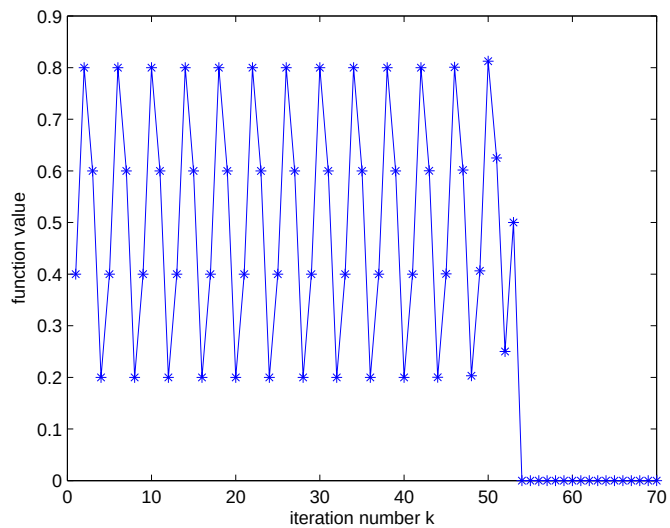


Figure 1 : small round errors destroy a simple iteration

Error Accumulation in Geometric GBA

- now we analysis the case: exact distances + linear least square

Error Accumulation in Geometric GBA

- now we analysis the case: exact distances + linear least square
- point j is to be determined, which has l distances to the determined points $1, 2, \ldots, l$, the computed coordinates are x_i, assume the ground truth are $\widehat{x}_i$, denote location error of point i by $\delta x_i = x_i - \widehat{x}_i$ and assume $\delta x_i = O(\delta x)$
- we need solve $\min_{x_j} \|Ax_j - b\|$, where

$$A = -2[x_{i+1} - x_i], \quad b = [(d_{i+1,j}^2 - d_{ij}^2) - (\|x_{i+1}\|^2 - \|x_i\|^2)] \tag{8}$$

Error Accumulation in Geometric GBA

- now we analysis the case: exact distances + linear least square
- point j is to be determined, which has l distances to the determined points $1, 2, \ldots, l$, the computed coordinates are x_i, assume the ground truth are $\widehat{x}_i$, denote location error of point i by $\delta x_i = x_i - \widehat{x}_i$ and assume $\delta x_i = O(\delta x)$
- we need solve $\min_{x_j} \|Ax_j - b\|$, where

$$A = -2[x_{i+1} - x_i], \quad b = [(d_{i+1,j}^2 - d_{ij}^2) - (\|x_{i+1}\|^2 - \|x_i\|^2)] \tag{8}$$

then we have

$$\|x_{i+1}\|^2 - \|x_i\|^2 = \|\widehat{x}_{i+1} + \delta x_{i+1}\|^2 - \|\widehat{x}_i + \delta x_i\|^2 \tag{9}$$

$$= \|\widehat{x}_{i+1}\|^2 - \|\widehat{x}_i\|^2 + 2\widehat{x}_{i+1}\delta x_{i+1} - 2\widehat{x}_i\delta x_i + \delta x_{i+1}^2 - \delta x_i^2 \tag{10}$$

therefore, $A = \widehat{A} + O(\delta x), \quad b = \widehat{b} + O(\delta x)$

$$\|Ax_j - b\| = \|(\widehat{A} + \delta A)(\widehat{x}_j + \delta x_j) - (\widehat{b} + \delta b)\| \tag{11}$$

$$= \|\widehat{A}\widehat{x} - \widehat{b} + \widehat{A}\delta x_j - O(\delta x)\| \tag{12}$$

Error Accumulation in Geometric GBA

- now we analysis the case: exact distances + linear least square
- point j is to be determined, which has l distances to the determined points $1, 2, \ldots, l$, the computed coordinates are x_i, assume the ground truth are $\widehat{x}_i$, denote location error of point i by $\delta x_i = x_i - \widehat{x}_i$ and assume $\delta x_i = O(\delta x)$
- we need solve $\min_{x_j} \|Ax_j - b\|$, where

$$A = -2[x_{i+1} - x_i], \quad b = [(d_{i+1,j}^2 - d_{ij}^2) - (\|x_{i+1}\|^2 - \|x_i\|^2)] \tag{8}$$

then we have

$$\|x_{i+1}\|^2 - \|x_i\|^2 = \|\widehat{x}_{i+1} + \delta x_{i+1}\|^2 - \|\widehat{x}_i + \delta x_i\|^2 \tag{9}$$

$$= \|\widehat{x}_{i+1}\|^2 - \|\widehat{x}_i\|^2 + 2\widehat{x}_{i+1}\delta x_{i+1} - 2\widehat{x}_i\delta x_i + \delta x_{i+1}^2 - \delta x_i^2 \tag{10}$$

therefore, $A = \widehat{A} + O(\delta x), \quad b = \widehat{b} + O(\delta x)$

$$\|Ax_j - b\| = \|(\widehat{A} + \delta A)(\widehat{x}_j + \delta x_j) - (\widehat{b} + \delta b)\| \tag{11}$$

$$= \|\widehat{A}\widehat{x} - \widehat{b} + \widehat{A}\delta x_j - O(\delta x)\| \tag{12}$$

- error in each step comes from
  - errors **inherit** from previous steps
  - computational error: round error and **ill-conditioned** coefficient matrix A

a toy example

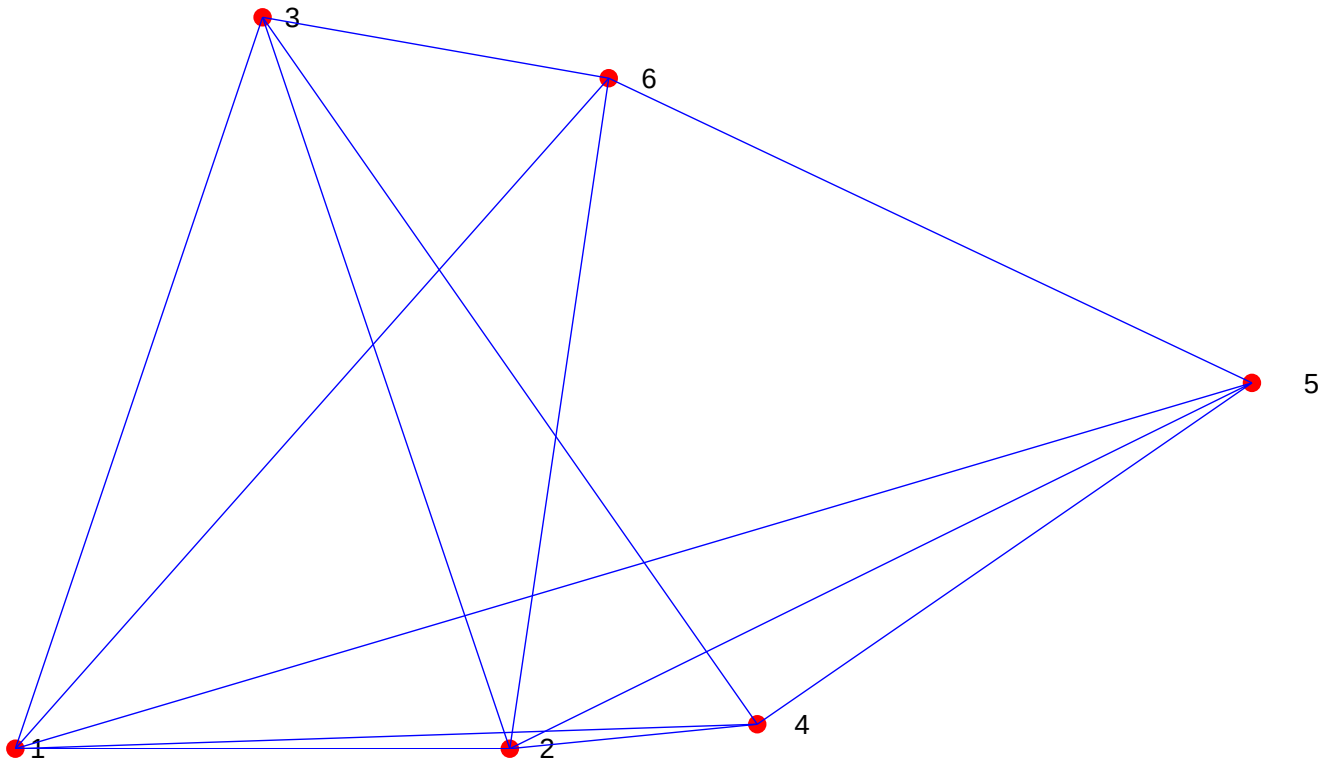


Figure 2 : a toy example to illustrate the importance of the computational order

our improvement: specified computational order

- We exploit two kinds of information to choose point j
  - **structure information**: the largest number of known distances
    Let $\mathcal{Y}$ be index set of determined points and $\mathcal{N} = \{1, 2, \ldots, n\} \backslash \mathcal{Y}$ the undetermined ones, we define

$$p(i) = \sharp\{d_{ik} \neq 0 \ : \ k \in \mathcal{Y}\}, \quad \textit{for } i \in \mathcal{N} \tag{13}$$

    and

$$\mathcal{I} = \{i : p(i) = \max_{k \in \mathcal{N}} p(k)\} \tag{14}$$

our improvement: specified computational order

- We exploit two kinds of information to choose point j
  - **structure information**: the largest number of known distances
    Let $\mathcal{Y}$ be index set of determined points and $\mathcal{N} = \{1, 2, \ldots, n\} \backslash \mathcal{Y}$ the undetermined ones, we define

$$p(i) = \sharp\{d_{ik} \neq 0 \ : \ k \in \mathcal{Y}\}, \quad \textit{for } i \in \mathcal{N} \tag{13}$$

    and

$$\mathcal{I} = \{i : p(i) = \max_{k \in \mathcal{N}} p(k)\} \tag{14}$$

  - **distance information**: closest to the determined points

$$j = \arg\min_{i \in \mathcal{I}} \sum_{k \in \mathcal{Y}} d_{ik} \tag{15}$$

our improvement: specified computational order

- We exploit two kinds of information to choose point j
  - **structure information**: the largest number of known distances
    Let $\mathcal{Y}$ be index set of determined points and $\mathcal{N} = \{1, 2, \ldots, n\} \backslash \mathcal{Y}$ the undetermined ones, we define

$$p(i) = \sharp\{d_{ik} \neq 0 \ : \ k \in \mathcal{Y}\}, \quad \textit{for } i \in \mathcal{N} \tag{13}$$

    and

$$\mathcal{I} = \{i : p(i) = \max_{k \in \mathcal{N}} p(k)\} \tag{14}$$

  - **distance information**: closest to the determined points

$$j = \arg\min_{i \in \mathcal{I}} \sum_{k \in \mathcal{Y}} d_{ik} \tag{15}$$

- Through this computational order
  - remove oscillation – "go far away and come back" [a movie]
  - utilize more distance information in total
  - almost always good-conditional $A \rightarrow$ computational difficulty is avoided

another improvement: stress minimization

- After computing X_j by linear/nonlinear least square, we further

$$\textit{minimize} \quad f(j, N(j)) \tag{16}$$

  where f is the stress function introduced before, to slightly adjust the locations of point j and its neighbours, exploiting only the given distances among them.

another improvement: stress minimization

- After computing X_j by linear/nonlinear least square, we further

$$\textit{minimize} \quad f(j, N(j)) \tag{16}$$

  where f is the stress function introduced before, to slightly adjust the locations of point j and its neighbours, exploiting only the given distances among them.
- benefits:
  - only the given distances are involved in problem (16), no errors from previous calculation
  - problem (16) is small-sized, not so much cost is added
  - least square solution can be viewed as a good initial point for this "difficult" problem

Numerical results

ID	num	per	degree			RMSD	fval	time(s)		
			max	min	ave			before	post	total
1PTQ	402	8.79	61	6	35.3	1.53e-01	75.91	3.9	0.3	4.2
1HOE	558	6.55	65	11	36.5	1.02e-01	107.14	5.8	0.4	6.2
1LFB	641	5.57	59	8	35.7	1.79e-01	124.43	7.3	0.5	7.8
1PHT	811	5.37	75	7	43.5	1.78e-01	197.98	10.6	0.7	11.3
1POA	914	4.07	67	8	37.2	1.64e-01	181.64	11.3	0.8	12.0
1AX8	1003	3.74	59	7	37.5	1.29e-01	205.92	10.5	0.9	11.4
1F39	1534	2.43	62	7	37.2	1.80e-01	310.26	16.3	1.8	18.1
1RGS	2015	1.87	66	4	37.7	1.98e-01	415.34	22.3	2.6	24.9
1KDH	2846	1.36	64	5	38.8	1.73e-01	608.00	35.2	4.9	40.0
1BPM	3671	1.12	64	4	40.9	1.22e-01	850.69	51.9	6.7	58.6
1RHJ	3740	1.10	61	5	41.2	1.21e-01	883.84	53.7	6.9	60.7
1HQQ	3944	1.00	64	5	39.5	1.78e-01	886.72	53.6	6.6	60.2
1TOA	4292	0.94	62	4	40.1	1.79e-01	969.63	61.8	7.4	69.2
1MQQ	5681	0.75	66	7	42.4	1.17e-01	1373.82	92.6	12.0	104.7

Table 1 : cutoff=6Å, noise=5%

Outline

Laplacian Matrix

Definition: Laplacian matrix

Given a graph $G = (V, E)$ and the parameter t, the Laplacian matrix L is defined by $L = D - W$, where

$$\omega_{ij} = \begin{cases} exp(-\|x_i - x_j\|^2/t) & if\ (i,j) \in E \\ 0 & otherwise \end{cases} \tag{17}$$

and D is a diagonal matrix with

$$D_{ii} = \sum_{j \neq i} \omega_{ij}.$$

$$L_{ij} = \begin{cases} -\omega_{ij} & i \neq j \\ \sum_{k \neq i} \omega_{ik} & i = j \end{cases} \tag{18}$$

[1] Chung, Fan RK. *Spectral graph theory*. Vol. 92. American Mathematical Soc., 1997.

Laplacian Matrix

Definition: Laplacian matrix

Given a graph $G = (V, E)$ and the parameter t, the Laplacian matrix L is defined by $L = D - W$, where

$$\omega_{ij} = \begin{cases} exp(-\|x_i - x_j\|^2/t) & if\ (i,j) \in E \\ 0 & otherwise \end{cases} \tag{17}$$

and D is a diagonal matrix with

$$D_{ii} = \sum_{j \neq i} \omega_{ij}.$$

$$L_{ij} = \begin{cases} -\omega_{ij} & i \neq j \\ \sum_{k \neq i} \omega_{ik} & i = j \end{cases} \tag{18}$$

- Note that when $t = \infty$, W degenerates to the adjacency matrix.
- Research about Laplacian matrix is called **spectral graph theory**[1].
- Two normalized variations:
  - $L_{sym} := D^{-1/2} L D^{-1/2} = I - D^{-1/2} W D^{-1/2}$
  - $L_{rw} := D^{-1} L = I - D^{-1} W$

[1] Chung, Fan RK. *Spectral graph theory*. Vol. 92. American Mathematical Soc., 1997.

Some properties of Laplacian matrix

For a graph G and its Laplacian matrix L with eigenvalues $\lambda_0 \leq \lambda_1 \leq \cdots \leq \lambda_{n-1}$

1. For every vector $f \in \mathbb{R}^n$ we have

$$f^T L f = \frac{1}{2} \sum_{i,j=1}^{n} \omega_{ij} (f_i - f_j)^2 \tag{19}$$

2. L is symmetric and positive-semidefinite.
3. λ_0 is always 0 because every Laplacian matrix has an eigenvector $v_0 = [1, 1, \ldots, 1]^T$.
4. The number of times 0 appears as an eigenvalue in the Laplacian is the number of connected components in the graph.

Proof of (1):

$$f^T L F = f^T D f - f^T W f = \sum_{i=1}^{n} d_{ii} f_i^2 - \sum_{i,j=1}^{n} f_i f_j \omega_{ij}$$

$$= \frac{1}{2} \Big(\sum_{i=1}^{n} d_{ii} f_i^2 - 2 \sum_{i,j=1}^{n} f_i f_j \omega_{ij} + \sum_{j=1}^{n} d_{jj} f_j^2 \Big) = \frac{1}{2} \sum_{i,j=1}^{n} \omega_{ij} (f_i - f_j)^2$$

Nonlinear Dimensionality Reduction

Dimensionality Reduction Problem

Given a set of n points $x_1, \ldots, x_n$ in $\mathbb{R}^l$, find a set of points $y_1, \ldots, y_n$ in $\mathbb{R}^m (m \ll l)$ such that y_i "represents" x_i.

Relation to DGP:

- In noisy case, the distances can not be **precisely** realized in low-dimensional space.
- If the distances do not contradict with each other, the points can be precisely realized in high-dimensional space.
- a two-dimensional example

Optimal Embedding

Problem: Map the weighted graph G to a line such that the adjacent vertices stay as close as possible.

Let $y = (y_1, y_2, \ldots, y_n)^T$ be such a map, so we want to minimize

$$\sum_{(i,j)\in E} (y_i - y_j)^2 \omega_{ij} = y^T L y \tag{20}$$

Optimal Embedding

Problem: Map the weighted graph G to a line such that the adjacent vertices stay as close as possible.

Let $y = (y_1, y_2, \ldots, y_n)^T$ be such a map, so we want to minimize

$$\sum_{(i,j)\in E} (y_i - y_j)^2 \omega_{ij} = y^T L y \tag{20}$$

Therefore,

$$\min_{y^T D y = 1, y^T D e = 0} y^T L y \tag{21}$$

gives us a reasonable solution.

Optimal Embedding

Problem: Map the weighted graph G to a line such that the adjacent vertices stay as close as possible.

Let $y = (y_1, y_2, \ldots, y_n)^T$ be such a map, so we want to minimize

$$\sum_{(i,j)\in E} (y_i - y_j)^2 \omega_{ij} = y^T L y \tag{20}$$

Therefore,

$$\min_{y^T D y = 1, y^T D e = 0} y^T L y \tag{21}$$

gives us a reasonable solution.

General case: Embed the graph into m-dimensional Euclidean space.

Let $Y = [y_1 \ y_2 \ \ldots \ y_n]^T \in \mathbb{R}^{n\times m}$ that each row gives a coordinate. Similarity we need to minimize

$$\sum_{(i,j)\in E} \|y_i - y_j\|^2 \omega_{ij} = tr(Y^T L Y) \tag{22}$$

Courant-Fischer Theorem

Courant-Fischer Theorem

Let A be a symmetric $n \times n$ matrix with eigenvalues $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$ and corresponding eigenvectors $v_1, v_2, \ldots, v_n$. For $1 \leq k \leq n$, let S_k denote the span of $v_1, \ldots, v_k$ (with $S_0 = \{0\}$), and let $S_k^{\perp}$ denote the orthogonal complement of S_k. Then

$$\lambda_k = \min_{\|x\|=1, x \in S_{k-1}^{\perp}} x^T A x = \min_{x \in S_{k-1}^{\perp}} \frac{x^T A x}{x^T x}$$

Nonlinear Dimensionality Reduction Algorithm

Nonlinear Dimensionality Reduction Algorithm framework (Belkin-Niyogi, 2002)

Step 1 Constructing the adjacency graph
(a) ϵ-neighborhood (b) k-nearest neighbors

Step 2 Choosing the weights
(a) Heat kernel (b) Simple-minded

Step 3 Eigenmaps: $x_i \to (f_1(i), \ldots, f_m(i))$, where $Lf_i = \lambda_i Df_i$, $0 = \lambda_0 \leq \lambda_1 \leq \ldots \leq \lambda_{n-1}$.

Grid Points Example 1

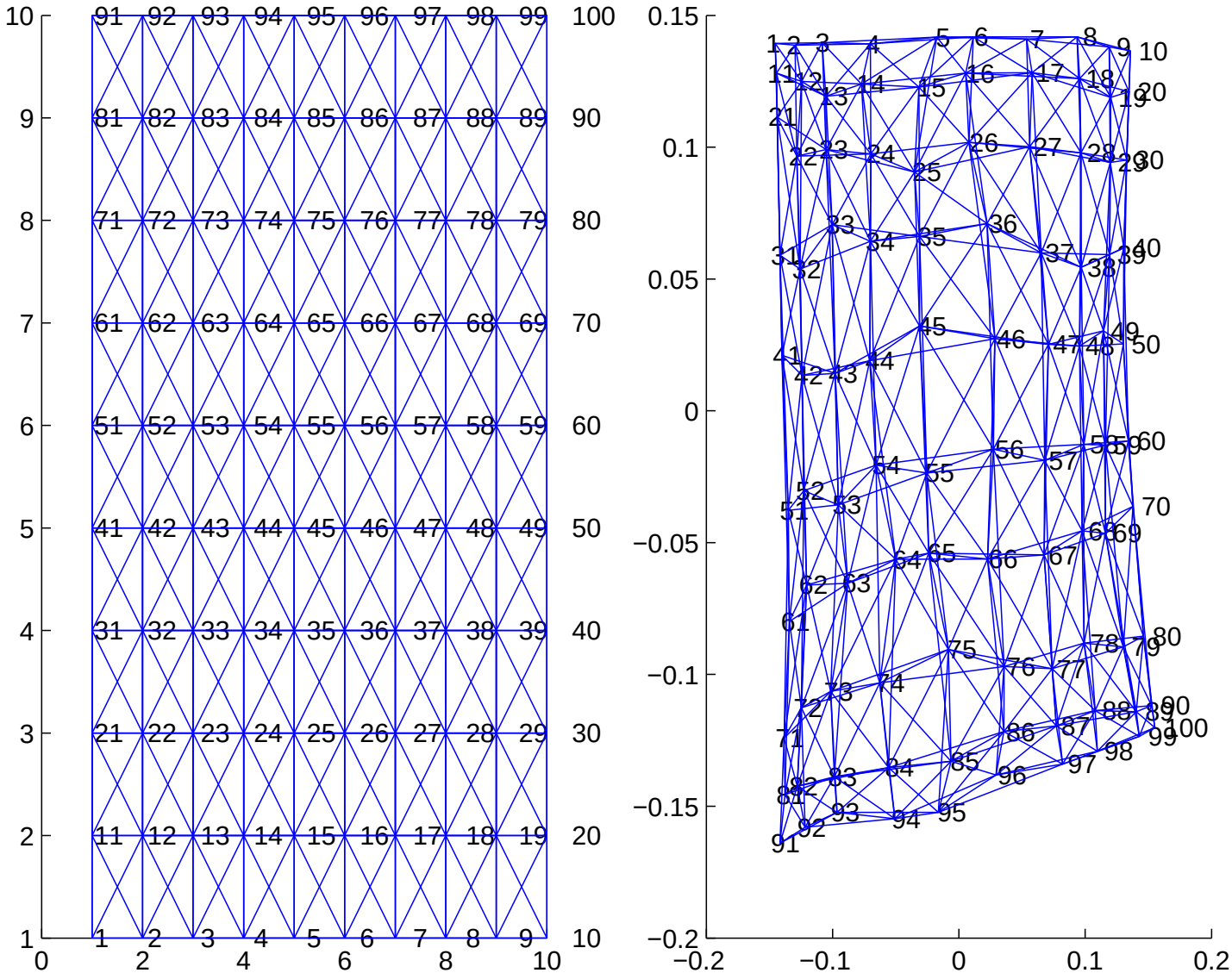


Figure 3 : cutoff=2, degree: [12,5,10], noise = 20%

Grid Points Example 2

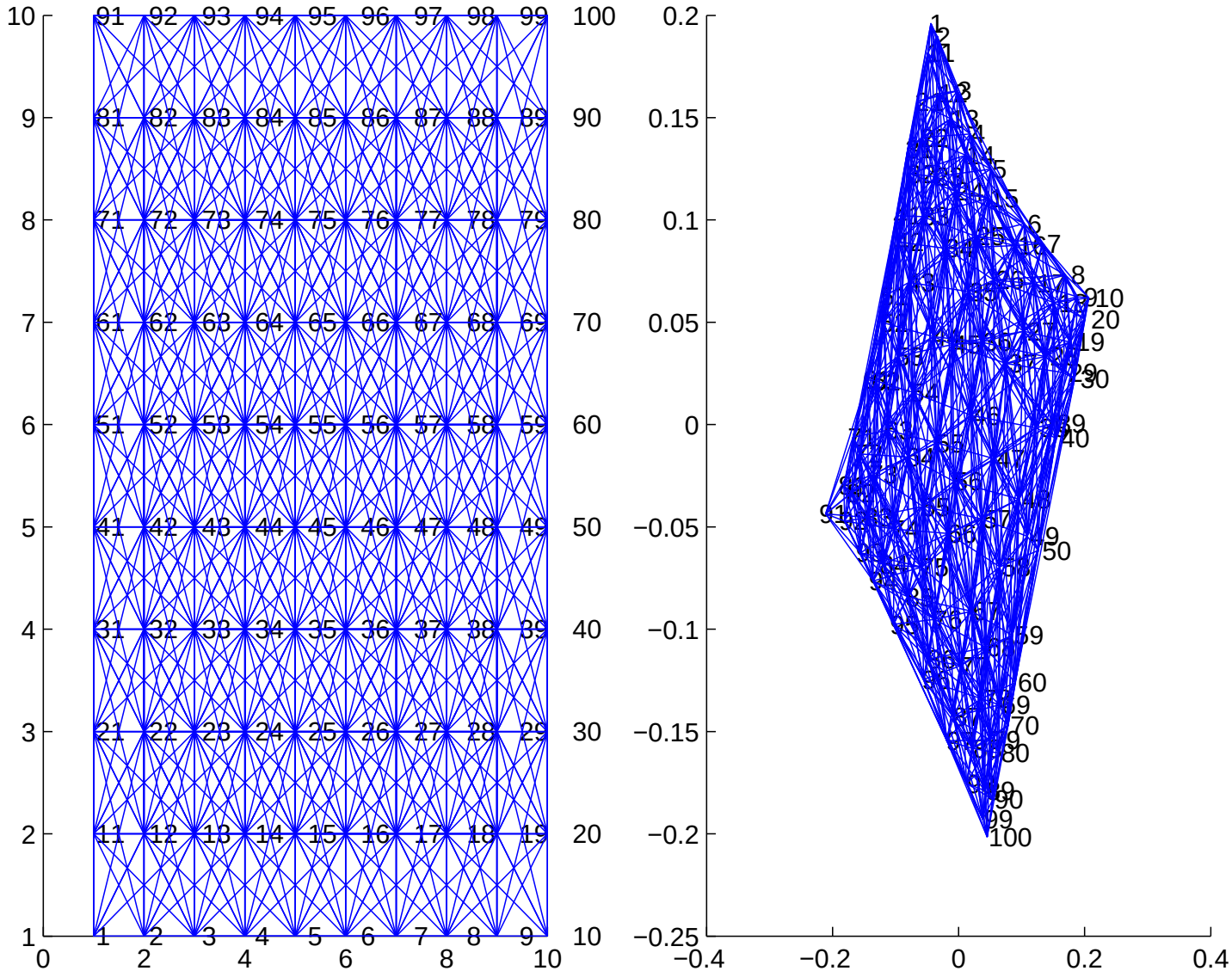
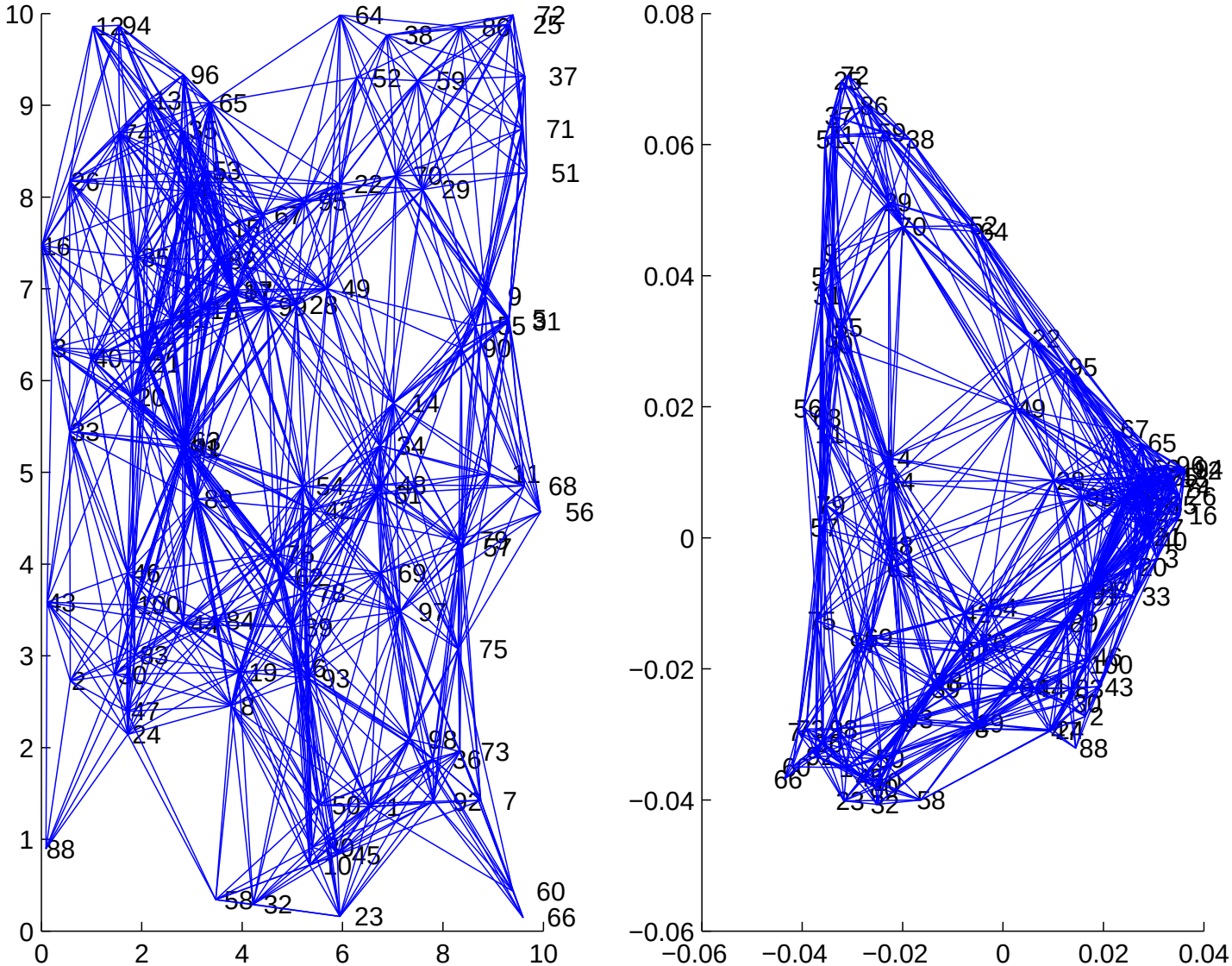
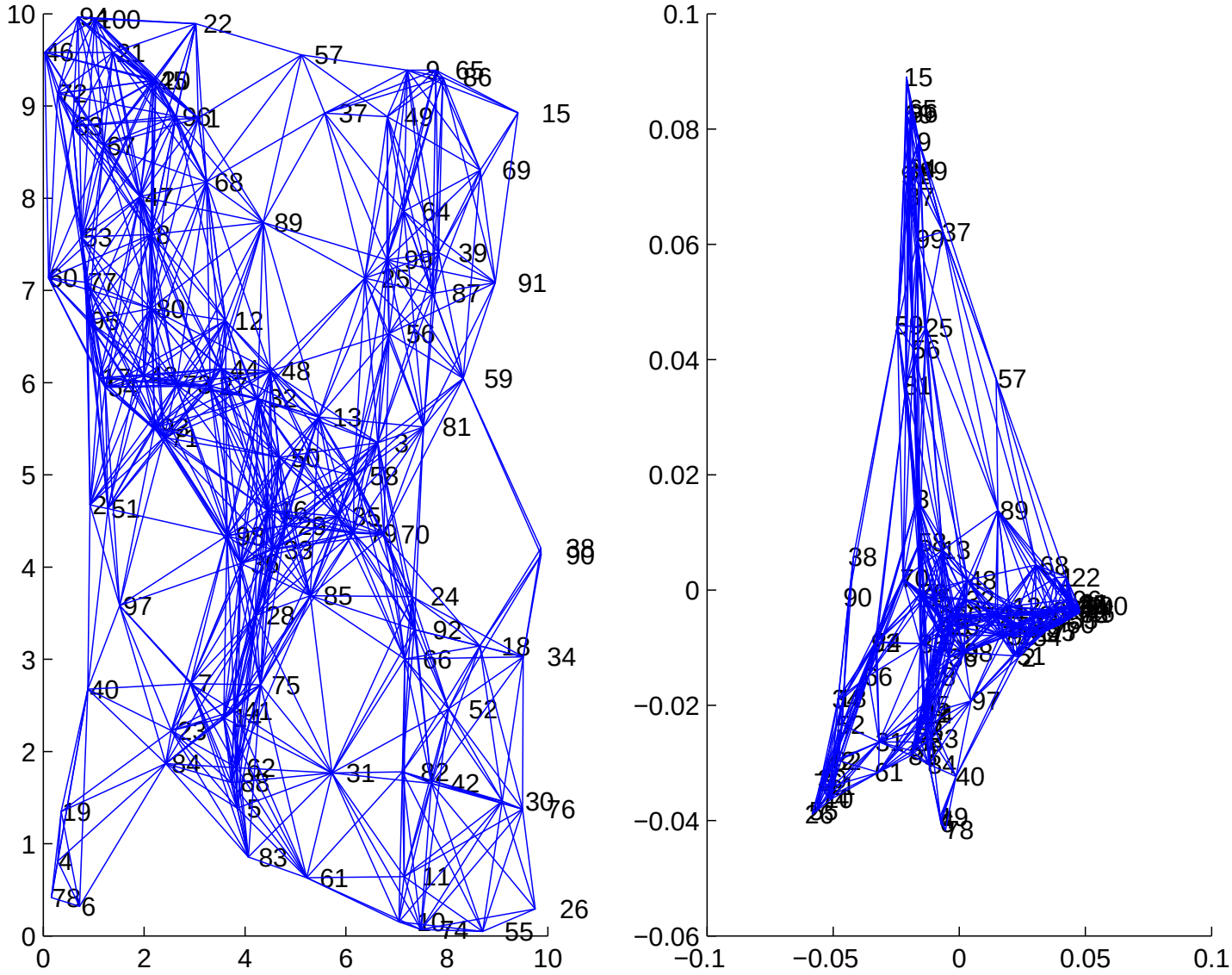


Figure 4 : cutoff=3, degree: [20,10,21.2], noise = 20%

a Random Example



another Random Example



Laplacian Eigenmap-based Localization Algorithm

Laplacian Eigenmap-based Localization Algorithm for DGP

Step 1 Construct the Laplacian matrix L and calculate its eigenvalue decomposition $L = VDV^T$ such that the eigenvalues in D are in ascending order.

Step 2 Set $X0 = V(:, 2 : m + 1)$ and scale it to the proper magnitude.

Step 3 Using $X0$ as the initial point, apply error function minimization method to further improve the result.

Laplacian Eigenmap-based Localization Algorithm

Laplacian Eigenmap-based Localization Algorithm for DGP

Step 1 Construct the Laplacian matrix L and calculate its eigenvalue decomposition $L = VDV^T$ such that the eigenvalues in D are in ascending order.

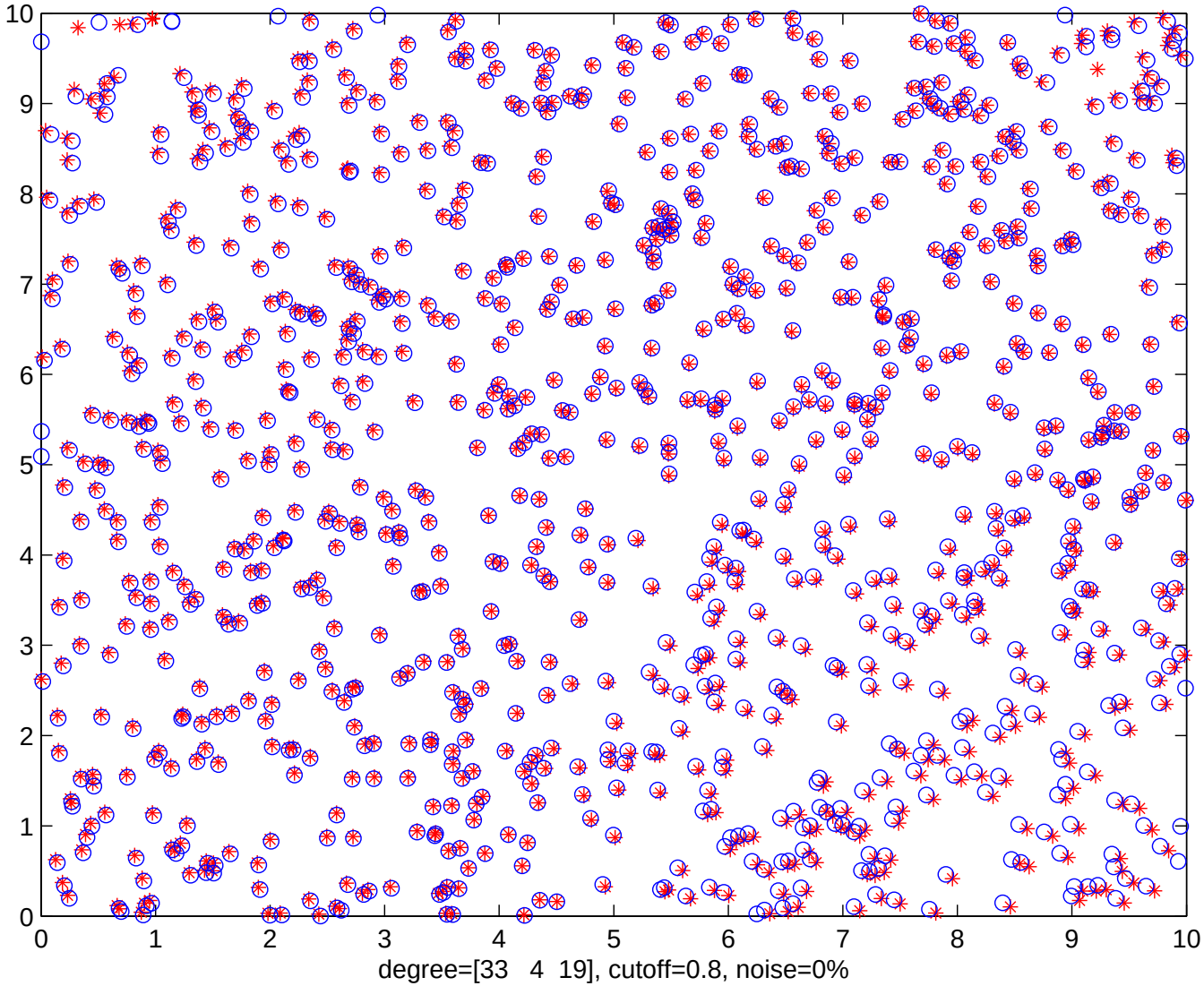
Step 2 Set $X0 = V(:, 2 : m + 1)$ and scale it to the proper magnitude.

Step 3 Using $X0$ as the initial point, apply error function minimization method to further improve the result.

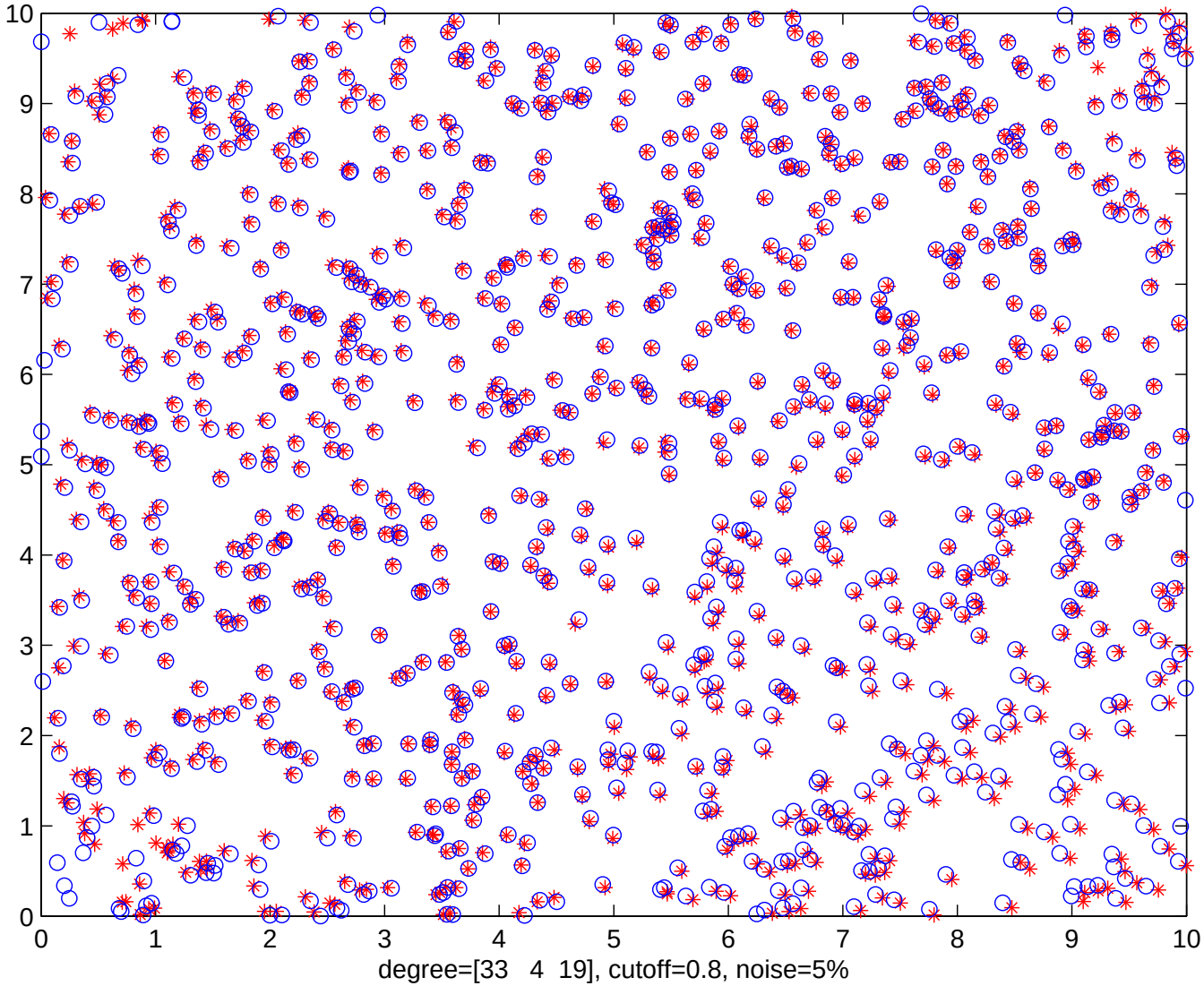
Scaling Method

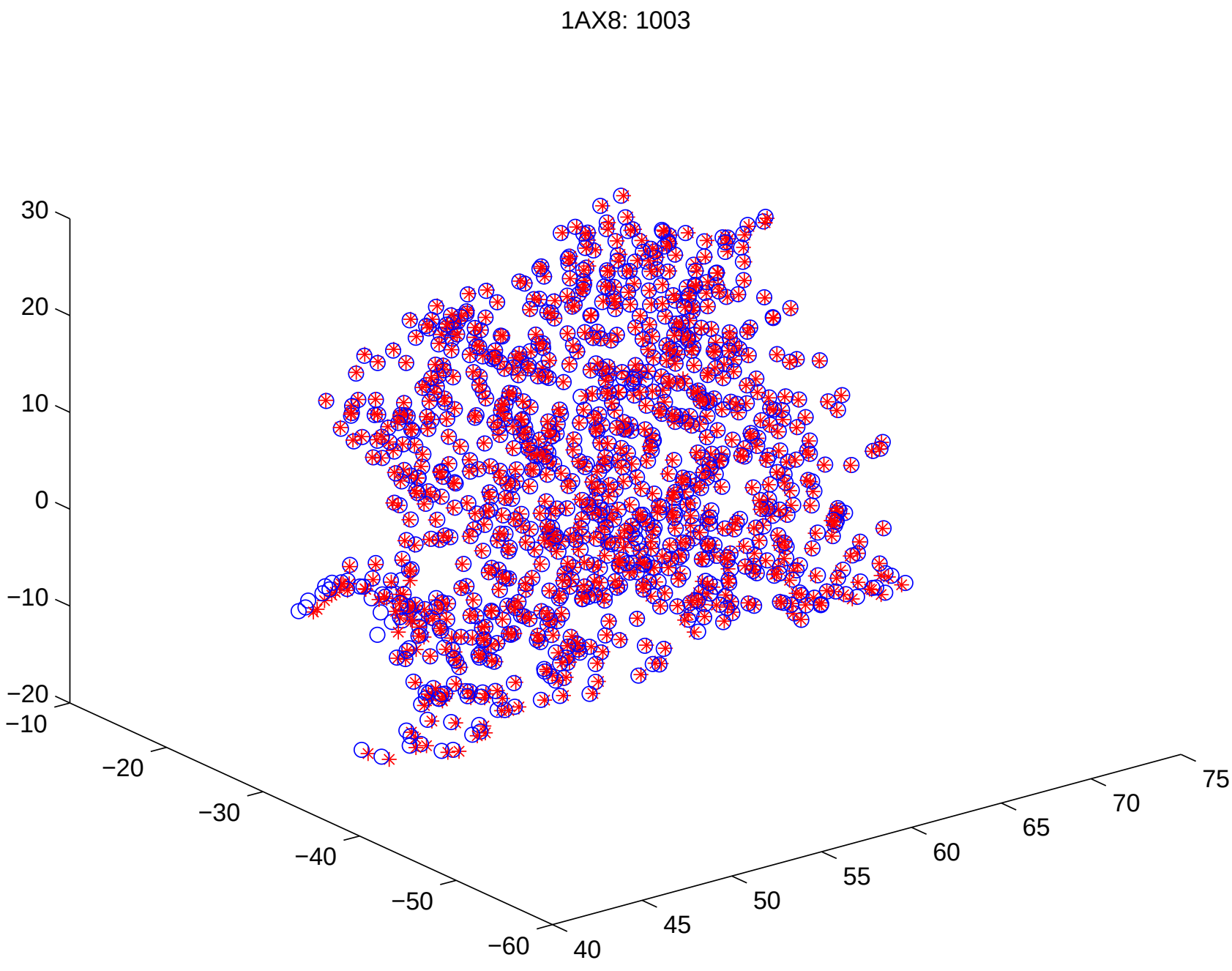
1. Construct a new distance matrix $\tilde{D}$ according to $X0$ and E.
2. Calculate
$$ratio = \frac{sum(D)}{sum(\tilde{D})}.$$
3. Set $X0 = X0 * ratio$.

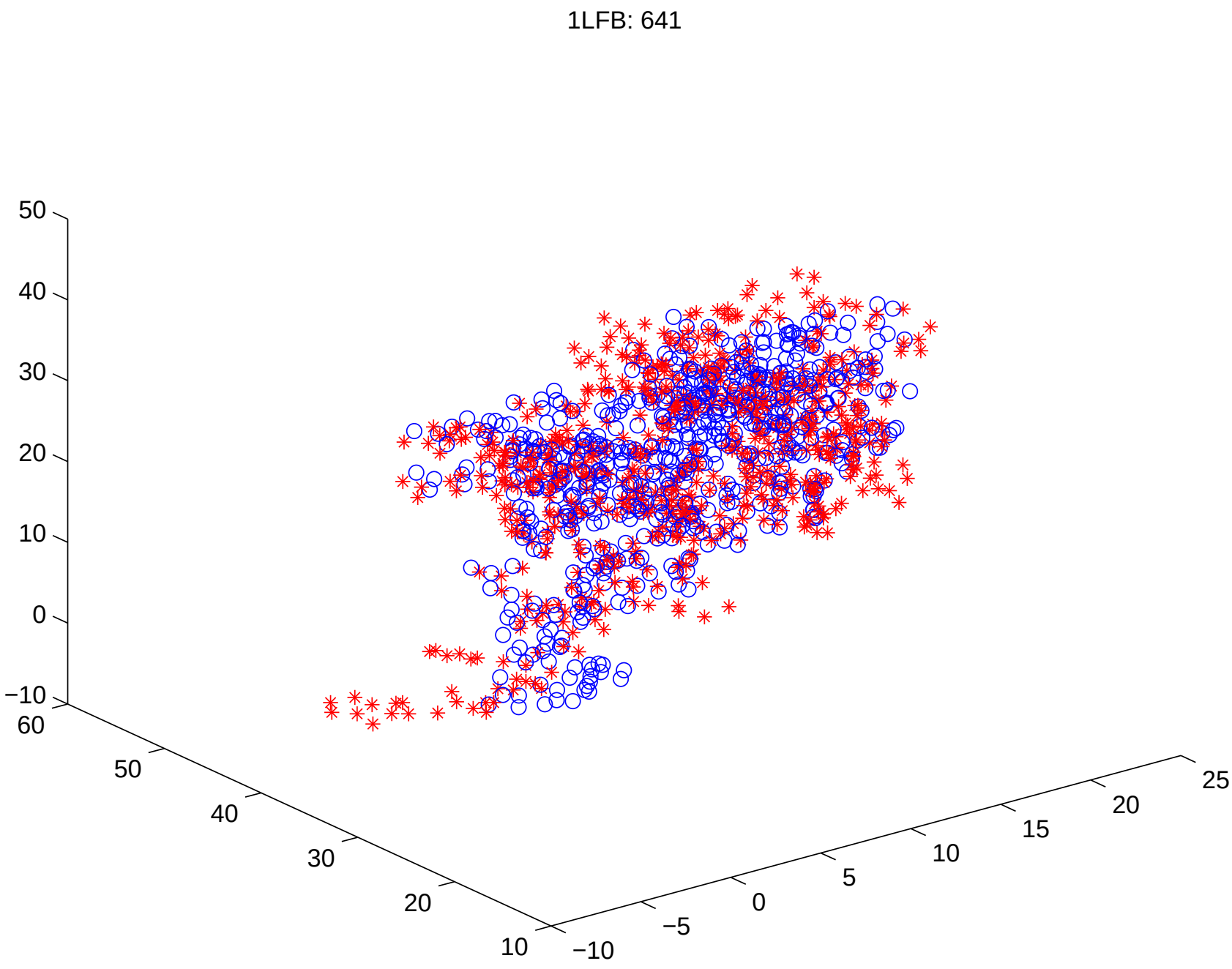
Random Example



Random Example




1AX8: 1003
30
20
10
0
−10
−20
−10
−20
−30
−40
−50
−60
40
45
50
55
60
65
70
75


1LFB: 641
50
40
30
20
10
0
−10
60
50
40
30
20
10
−10
−5
0
5
10
15
20
25

Remarks

- Pros
  - very simple and fast
  - relatively robust to the noise
  - good in uniformly generated random instances
- Cons
  - sensitive to the parameter t and no theoretically-guaranteed good way to choose it
  - terrible in most of real protein instances, except for 1PTQ, 1HOE, 1AX8

Spectral Clustering

Normalized spectral clustering according to Shi and Malik (2000)

Input: Similarity matrix $S \in \mathbb{R}^{n \times n}$, number k of clusters to construct.

- Construct a similarity graph by one of the ways described in Section 2. Let W be its weighted adjacency matrix.
- Compute the unnormalized Laplacian L.
- **Compute the first k generalized eigenvectors $u_1, \ldots, u_k$ of the generalized eigenproblem $Lu = \lambda Du$.**
- Let $U \in \mathbb{R}^{n \times k}$ be the matrix containing the vectors $u_1, \ldots, u_k$ as columns.
- For $i = 1, \ldots, n$, let $y_i \in \mathbb{R}^k$ be the vector corresponding to the i-th row of U.
- Cluster the points $(y_i)_{i=1,\ldots,n}$ in $\mathbb{R}^k$ with the k-means algorithm into clusters $C_1, \ldots, C_k$.

Output: Clusters $A_1, \ldots, A_k$ with $A_i = \{j | \, y_j \in C_i\}$.

Remarks about Spectral Clustering for DGP

- good news: clusters have been observed in numerical results

Remarks about Spectral Clustering for DGP

- good news: clusters have been observed in numerical results
- main difficulties:
  - no obvious clusters exist in proteins
  - hard to choose the prescribed cluster number k
  - stable stitch step need reliable **overlapping points**

Outline

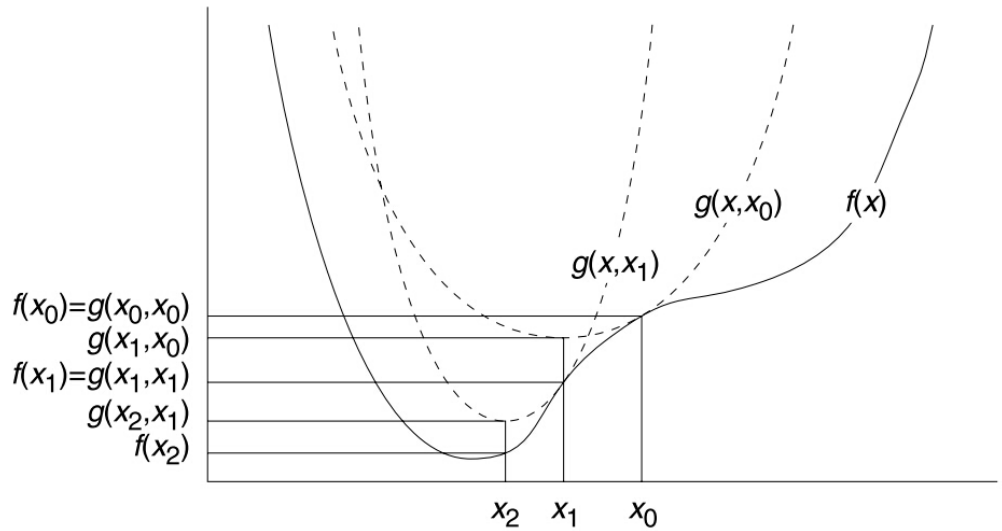
Principle of Majorization[2]

- The central idea of the majorization method is to replace iteratively the original complicated function $f(x)$ by an auxiliary function $g(x, z)$, where z in $g(x, z)$ is some fixed value.

[2] Borg, Ingwer, and Patrick JF Groenen. *Modern multidimensional scaling: Theory and applications*. Springer, 2005.

Principle of Majorization[2]

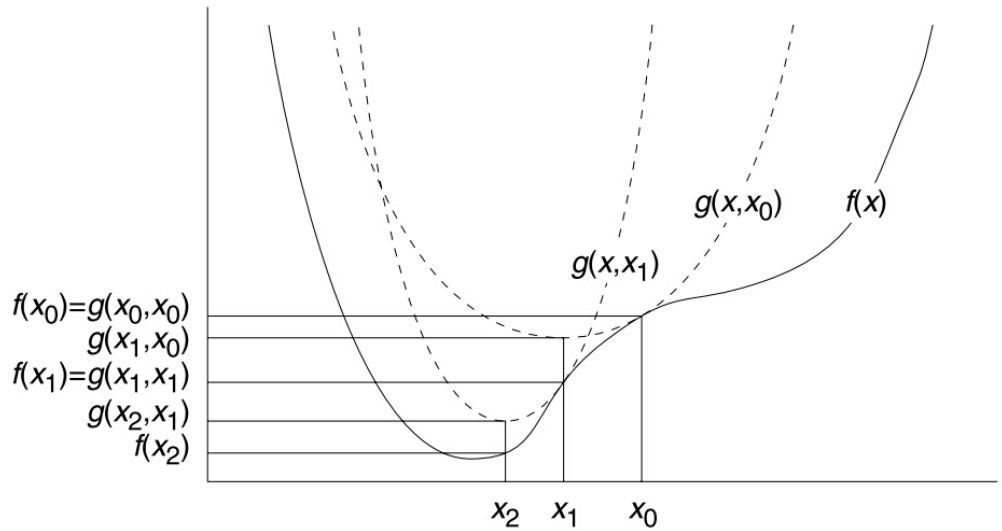
- The central idea of the majorization method is to replace iteratively the original complicated function $f(x)$ by an auxiliary function $g(x, z)$, where z in $g(x, z)$ is some fixed value.
- $g(x, z)$ is a *majorization function* of $f(x)$ if it satisfies
  - $g(x, z)$ should be simpler to minimize than $f(x)$.
  - the original function must always be smaller than or at most equal to the auxiliary function, i.e., $f(x) \leq g(x, z)$.
  - the auxiliary function should touch the surface at the so-called *supporting point* z, i.e., $f(z) = g(z, z)$.



[2] Borg, Ingwer, and Patrick JF Groenen. *Modern multidimensional scaling: Theory and applications*. Springer, 2005.

Principle of Majorization[2]

- The central idea of the majorization method is to replace iteratively the original complicated function $f(x)$ by an auxiliary function $g(x, z)$, where z in $g(x, z)$ is some fixed value.
- $g(x, z)$ is a *majorization function* of $f(x)$ if it satisfies
  - $g(x, z)$ should be simpler to minimize than $f(x)$.
  - the original function must always be smaller than or at most equal to the auxiliary function, i.e., $f(x) \leq g(x, z)$.
  - the auxiliary function should touch the surface at the so-called *supporting point* z, i.e., $f(z) = g(z, z)$.



- *monotone decreasing*: let $x^* = \arg\min_x g(x, z)$, then we have

$$f(x^*) \leq g(x^*, z) \leq g(z, z) = f(z) \tag{23}$$

[2] Borg, Ingwer, and Patrick JF Groenen. *Modern multidimensional scaling: Theory and applications*. Springer, 2005.

Stress Majorization(1)

- A widely used technique in Multidimensional Scaling (MDS) community.
- MDS is a means of visualizing the level of similarity of individual cases of a dataset. An MDS algorithm aims to place each object in N-dimensional space such that the between-object distances are preserved as well as possible[3].

[3] http://en.wikipedia.org/wiki/Multidimensional_scaling

Stress Majorization(1)

- A widely used technique in Multidimensional Scaling (MDS) community.
- MDS is a means of visualizing the level of similarity of individual cases of a dataset. An MDS algorithm aims to place each object in N-dimensional space such that the between-object distances are preserved as well as possible[3].
- Let $X = (x_1, x_2, \ldots, x_n)^T \in \mathbb{R}^{n \times r}$, where each row is the coordinate of one point, and each column denotes one axis, rewrite

$$Stress(X) = \sum_{i<j} \omega_{ij} (\|x_i - x_j\| - d_{ij})^2 \tag{24}$$

$$= \sum_{i<j} \omega_{ij} d_{ij}^2 + \sum_{i<j} \omega_{ij} \|x_i - x_j\|^2 - 2 \sum_{i<j} \omega_{ij} d_{ij} \|x_i - x_j\| \tag{25}$$

[3] http://en.wikipedia.org/wiki/Multidimensional_scaling

Stress Majorization(1)

- A widely used technique in Multidimensional Scaling (MDS) community.
- MDS is a means of visualizing the level of similarity of individual cases of a dataset. An MDS algorithm aims to place each object in N-dimensional space such that the between-object distances are preserved as well as possible[3].
- Let $X = (x_1, x_2, \ldots, x_n)^T \in \mathbb{R}^{n \times r}$, where each row is the coordinate of one point, and each column denotes one axis, rewrite

$$Stress(X) = \sum_{i<j} \omega_{ij}(\|x_i - x_j\| - d_{ij})^2 \tag{24}$$

$$= \sum_{i<j} \omega_{ij} d_{ij}^2 + \sum_{i<j} \omega_{ij} \|x_i - x_j\|^2 - 2 \sum_{i<j} \omega_{ij} d_{ij} \|x_i - x_j\| \tag{25}$$

- The second term $\sum_{i<j} \omega_{ij} \|x_i - x_j\|^2 = Tr(X^T L^\omega X)$, where

$$L_{ij}^\omega = \begin{cases} -\omega_{ij} & i \neq j \\ \sum_{k \neq i} \omega_{ik} & i = j \end{cases} \tag{26}$$

[3] http://en.wikipedia.org/wiki/Multidimensional_scaling

Stress Majorization(2)

- Now we bound the third term. By Cauchy-Schwartz inequality,

$$(x_i - x_j)^T (z_i - z_j) \leq \|x_i - x_j\| \|z_i - z_j\| \tag{27}$$

then we have

$$\sum_{i<j} \omega_{ij} d_{ij} \|x_i - x_j\| \geq \sum_{i<j} \omega_{ij} d_{ij} inv(\|z_i - z_j\|)((x_i - x_j)^T (z_i - z_j)) = Tr(X^T L^Z Z) \tag{28}$$

where $inv(x) = 1/x$ when $x \neq 0$ and 0 otherwise, and

$$L_{ij}^Z = \begin{cases} -\omega_{ij} d_{ij} inv(\|z_i - z_j\|) & i \neq j \\ -\sum_{j \neq i} L_{ij}^Z & i = j \end{cases} \tag{29}$$

Stress Majorization(2)

- Now we bound the third term. By Cauchy-Schwartz inequality,

$$(x_i - x_j)^T (z_i - z_j) \leq \|x_i - x_j\| \|z_i - z_j\| \tag{27}$$

then we have

$$\sum_{i<j} \omega_{ij} d_{ij} \|x_i - x_j\| \geq \sum_{i<j} \omega_{ij} d_{ij} inv(\|z_i - z_j\|)((x_i - x_j)^T (z_i - z_j)) = Tr(X^T L^Z Z) \tag{28}$$

where $inv(x) = 1/x$ when $x \neq 0$ and 0 otherwise, and

$$L_{ij}^Z = \begin{cases} -\omega_{ij} d_{ij} inv(\|z_i - z_j\|) & i \neq j \\ -\sum_{j \neq i} L_{ij}^Z & i = j \end{cases} \tag{29}$$

- In summary, $Stress(X)$ can be bounded by $F^Z(X)$ defined as

$$F^Z(X) = \sum_{i<j} \omega_{ij} d_{ij}^2 + Tr(X^T L^\omega X) - 2\, Tr(X^T L^Z Z) \tag{30}$$

Stress Majorization(2)

- Now we bound the third term. By Cauchy-Schwartz inequality,

$$(x_i - x_j)^T (z_i - z_j) \leq \|x_i - x_j\| \|z_i - z_j\| \tag{27}$$

then we have

$$\sum_{i<j} \omega_{ij} d_{ij} \|x_i - x_j\| \geq \sum_{i<j} \omega_{ij} d_{ij} inv(\|z_i - z_j\|)((x_i - x_j)^T (z_i - z_j)) = Tr(X^T L^Z Z) \tag{28}$$

where $inv(x) = 1/x$ when $x \neq 0$ and 0 otherwise, and

$$L_{ij}^Z = \begin{cases} -\omega_{ij} d_{ij} inv(\|z_i - z_j\|) & i \neq j \\ -\sum_{j \neq i} L_{ij}^Z & i = j \end{cases} \tag{29}$$

- In summary, $Stress(X)$ can be bounded by $F^Z(X)$ defined as

$$F^Z(X) = \sum_{i<j} \omega_{ij} d_{ij}^2 + Tr(X^T L^\omega X) - 2\, Tr(X^T L^Z Z) \tag{30}$$

- The minima of $F^Z(X)$ are given by solving

$$L^\omega X = L^Z Z \tag{31}$$

Stress Majorization(2)

- Now we bound the third term. By Cauchy-Schwartz inequality,

$$(x_i - x_j)^T (z_i - z_j) \leq \|x_i - x_j\| \|z_i - z_j\| \tag{27}$$

then we have

$$\sum_{i<j} \omega_{ij} d_{ij} \|x_i - x_j\| \geq \sum_{i<j} \omega_{ij} d_{ij} inv(\|z_i - z_j\|)((x_i - x_j)^T (z_i - z_j)) = Tr(X^T L^Z Z) \tag{28}$$

where $inv(x) = 1/x$ when $x \neq 0$ and 0 otherwise, and

$$L_{ij}^Z = \begin{cases} -\omega_{ij} d_{ij} inv(\|z_i - z_j\|) & i \neq j \\ -\sum_{j \neq i} L_{ij}^Z & i = j \end{cases} \tag{29}$$

- In summary, $Stress(X)$ can be bounded by $F^Z(X)$ defined as

$$F^Z(X) = \sum_{i<j} \omega_{ij} d_{ij}^2 + Tr(X^T L^\omega X) - 2\, Tr(X^T L^Z Z) \tag{30}$$

- The minima of $F^Z(X)$ are given by solving

$$L^\omega X = L^Z Z \tag{31}$$

- the algorithm converges to a stationary point linearly.

Thank you for your attention!

szl@lsec.cc.ac.cn