User Manual for **SFSDP**: a **S**parse Version of **F**ull **S**emi**D**efinite **P**rogramming Relaxation for Sensor Network Localization Problems

Sunyoung Kim[⋆], Masakazu Kojima[†], Hayato Waki[‡], and Makoto Yamashita[♯]

August 2008, Revised January 2010

Abstract.

SFSDP is a Matlab package for solving sensor network localization problems. The package contains four functions, SFSDP.m, SFSDPplus.m, generateProblem.m, test_SFSDP.m, and some numerical examples. The function SFSDP.m is a Matlab implementation of the semidefinite programming (SDP) relaxation proposed in the recent paper by Kim, Kojima and Waki for sensor network localization problems, as a sparse version of the full semidefinite programming relaxation (FSDP) by Biswas and Ye. To improve the efficiency of FSDP, SFSDP.m exploits the aggregated and correlative sparsity of a sensor network localization problem. The function SFSDPplus.m analyzes the input data of a sensor network localization problem, solves the problem, and displays graphically computed locations of sensors. The function generateProblem.m creates numerical examples of sensor network localization problems with representative anchor locations. The function test_SFSDP.m is for numerical experiments using SFSDPplus.m with test problems generated by generateProblem.m. The package **SFSDP** and this manual are available at

http://www.is.titech.ac.jp/∼kojima/SFSDP

Key words.

Sensor network localization problems, Semidefinite programming relaxation, Sparsity exploitation, Matlab software package.

⋆ Department of Mathematics, Ewha W. University, 11-1 Dahyun-dong, Sudaemoon-gu, Seoul 120-750 Korea. S. Kim's research was supported by KOSEF 2009-007-1314. *skim@ewha.ac.kr*

† Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2-12-1 Oh-Okayama, Meguro-ku, Tokyo 152-8552 Japan. M. Kojima's research was supported by Grant-in-Aid for Scientific Research (B) 19310096. *kojima@is.titech.ac.jp*

‡ Department of Computer Science, The University of Electro-Communications 1-5-1 Chofugaoka, Chofu-shi, Tokyo, Japan. H. Waki's research was supported by Grant-in-Aid for JSPS Fellows 20003236. *hayato.waki@jsb.cs.uec.ac.jp*

♯ Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2-12-1 Oh-Okayama, Meguro-ku, Tokyo 152-8552 Japan. M. Yamashita's research was supported by Grant-in-Aid for Young Scientists (B) 21710148. *Makoto.Yamashita@is.titech.ac.jp*

1 Introduction

For a network of n sensors, where $n > m$, a sensor network localization problem is to locate m sensors that fit the given distances if a subset of distances and some sensors of known position (called anchors) are provided. Various approaches [1, 6, 7, 10, 11, 19] have been proposed for the problem to approximate the solutions. Full semidefinite programming relaxation (FSDP) was introduced by Biswas and Ye in [2], and a number of solution methods based on SDP relaxation have followed [3, 4, 5, 15, 20].

We introduce a Matlab package SFSDP for solving sensor network localization problems by SDP relaxation. The main function SFSDP.m of the package is an implementation of the SDP relaxation proposed in the recent paper by Kim, Kojima and Waki [12]. SFSDP.m is intended to improve the efficiency of Biswas and Ye's FSDP [2] by exploiting the sparsity, the aggregated and correlative sparsity [9, 14, 13], of sensor network problems. The quality of obtained solution by SFSDP.m remains equivalent to that by FSDP. As a result, SFSDP.m can handle larger-sized sensor network problems, e.g., up to 6000 sensors in 2-dimensional case, than FSDP.

SFSDP.m can solve the problem with exact and noisy distances. To describe a form of the sensor network localization problem that can be solved by SFSDP.m, we consider a problem with m sensors and m_a $(= n - m)$ anchors. Let $\rho > 0$ be a radio range, which determines the set $\mathcal{N}_x^\rho$ of pairs of sensors p and q such that their unknown (Euclidean) distance d_{pq} is not greater than ρ, and the set $\mathcal{N}_a^\rho$ of pairs of a sensor p and an anchor r such that their distance d_{pr} does not exceed ρ;

$$\left.\begin{array}{rcl} \mathcal{N}_x^\rho & = & \{(p,q) : 1 \le p < q \le m, \|\bar{\boldsymbol{x}}_p - \bar{\boldsymbol{x}}_q\| \le \rho\}, \\ \mathcal{N}_a^\rho & = & \{(p,r) : 1 \le p \le m, m+1 \le r \le n, \|\bar{\boldsymbol{x}}_p - \boldsymbol{a}_r\| \le \rho\}, \end{array}\right\} \tag{1}$$

where $\bar{\boldsymbol{x}}_p$ denotes unknown location of sensor p and $\boldsymbol{a}_r$ known location of anchor r. Let $\mathcal{N}_x$ be a subset of $\mathcal{N}_x^\rho$ and and $\mathcal{N}_a$ a subset of $\mathcal{N}_a^\rho$. For ℓ-dimensional problem, an $\ell \times m$ matrix variable $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m) \in \mathbb{R}^{\ell \times m}$ denotes location of the sensors. SFSDP.m can solve the problem of $\ell = 2$ or 3. By introducing zero objective function and the distance equations as constraints, we have the following form of the sensor network localization problem with exact distances.

$$\left.\begin{array}{ll} \text{minimize} & 0 \\ \text{subject to} & d_{pq}^2 = \|\boldsymbol{x}_p - \boldsymbol{x}_q\|^2 \;\; (p,q) \in \mathcal{N}_x, \\ & d_{pr}^2 = \|\boldsymbol{x}_p - \boldsymbol{a}_r\|^2 \;\; (p,r) \in \mathcal{N}_a. \end{array}\right\} \tag{2}$$

When the distance involves noise, the following problem is considered.

$$\left.\begin{array}{ll} \text{minimize} & \displaystyle\sum_{(p,q)\in\mathcal{N}_x} \left(\epsilon_{pq}^+ + \epsilon_{pq}^-\right) + \sum_{(p,r)\in\mathcal{N}_a} \left(\epsilon_{pr}^+ + \epsilon_{pr}^-\right) \\ \text{subject to} & \hat{d}_{pq}^2 = \|\boldsymbol{x}_p - \boldsymbol{x}_q\|^2 + \epsilon_{pq}^+ - \epsilon_{pq}^- \;\; (p,q) \in \mathcal{N}_x, \\ & \hat{d}_{pr}^2 = \|\boldsymbol{x}_p - \boldsymbol{a}_r\|^2 + \epsilon_{pr}^+ - \epsilon_{pr}^- \;\; (p,r) \in \mathcal{N}_a, \\ & \epsilon_{pq}^+ \ge 0,\; \epsilon_{pq}^- \ge 0,\; (p,q) \in \mathcal{N}_x, \\ & \epsilon_{pr}^+ \ge 0,\; \epsilon_{pr}^- \ge 0, (p,r) \in \mathcal{N}_a. \end{array}\right\} \tag{3}$$

Here $\epsilon_{pq}^+ + \epsilon_{pq}^-$ (or $\epsilon_{pr}^+ + \epsilon_{pr}^-$) indicates 1-norm error in the estimated distance $\hat{d}_{pq}$ between sensors p and q (or an estimated distance $\hat{d}_{pr}$ between sensor p and anchor r, respectively).

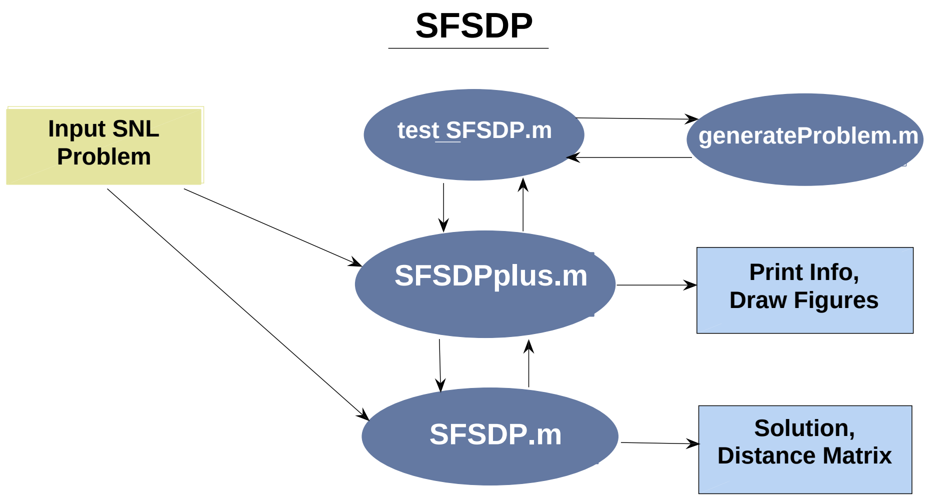


Figure 1: The structure of SFSDP

When a sensor network problem of the form (2) or (3) has many equality constraints that may be redundant, the resulting SDP relaxation problem can be too large to solve. To deal with such a problem, SFSDP.m replaces $\mathcal{N}_x$ and $\mathcal{N}_a$ by smaller subsets of them, $\mathcal{N}'_x$ and $\mathcal{N}'_a$, respectively, before applying the sparse SDP relaxation to the problem (2) or (3). Then, the resulting SDP relaxation problem becomes smaller and sparser. This process is a key for solving large scale sensor network localization problems efficiently by SFSDP.m. See Section 4.1 of [12] for more details. We assume that either (i) (noisy) distance information is available between a fairly large number of sensors and anchors in the original problem (2) or (3) to extract a smaller-sized subproblem satisfying the sparsity (the aggregated and correlative sparsity) or (ii) the original problem itself is sparse. If we take $\mathcal{N}^\rho_x$ and $\mathcal{N}^\rho_a$ (or their subsets large enough) for $\mathcal{N}_x$ and $\mathcal{N}_a$, respectively, the assumption (i) is usually satisfied. We should note, however, that SFSDP.m may fail to solve the problem efficiently if neither (i) nor (ii) is satisfied.

Edge-based SDP (ESDP) and node-based SDP (NSDP) relaxations were introduced in [20] to improve the computational efficiency of the original Biswas-Ye SDP relaxation FSDP. These SDP relaxations are further relaxations of FSDP, hence, they are theoretically weaker than FSDP. SFSDP.m, however, is shown to be equivalent to FSDP in [12].

The structure of the package SFSDP is shown in Figure 1. In addition to SFSDP.m, the package includes three functions, SFDPplus.m, generateProblem.m, and test_SFDP.m. The function SFDPplus.m is designed for users who want to solve their own sensor network localization problems. Users can use SFSDP.m via SFSDPplus.m or SFSDP.m directly. After analyzing input data of a given problem, SFSDPplus.m solves the problem by SFSDP.m, and displays graphically computed locations of sensors. Users can call either of SFSDP.m and SFDPplus.m from their own Matlab function that can provide necessary input data.

SFSDP.m calls SDPA [8], available at [17], or SeDuMi [18], available at [16], to solve SDP relaxation problems. For larger problems, using SDPA requires much less computational time. See Appendix.

The other two functions generateProblem.m and test_SFDP.m are for users interested in numerical experiments using SFDPplus.m. The function generateProblem.m creates numerical examples of sensor network localization problems with representative anchor locations. The function test_SFSDP.m is for numerical experiments on SFSDPplus.m applied to test problems generated by generateProblem.m. We discuss in detail input and output for the functions SFSDP.m, SFSDPplus.m, generateProblem.m and test_SFSDP.m in Section 4.

2 Sample Run Using SDPA

The usage of SFSDPplus.m, SFSDP.m, generateProblems.m, and test_SFSDP.m is described in this section.

2.1 SFSDPplus.m

We show how SFSDPplus.m can be executed with an illustrative example. A small problem of 3 sensors and 4 anchors in the two dimensional space is generated with the following xMatrix0 and distanceMatrix0. Assume that the sensors are located at $(0.3, 0.4)$, $(0.3, 0.6)$, and $(0.7, 0.6)$ and the anchors are at $(0, 0), (0, 1), (1, 0)$, and $(1, 1)$. Then, we prepare input data and parameters as follows:

```
>> sDim= 2; noOfSensors= 3; noOfAnchors= 4;
>> pars.free= 0; pars.eps= 1.0000e-05; pars.minDegree= 4; pars.objSW = 1;
>> pars.noisyFac= 0;
```

The size of xMatrix0 is sDim $\times$ (noOfSensors + noOfAnchors) matrix and its elements are:

```
>> xMatrix0
xMatrix0 =
    0.3000    0.3000    0.7000         0         0    1.0000    1.0000
    0.4000    0.6000    0.6000         0    1.0000         0    1.0000
```

The first three columns of xMatrix0, which indicate the true location of sensors, can be omitted for general cases where the locations of sensors are unknown.

The distance information is stored in a matrix called distanceMatrix0. The size of distanacneMatrix0 is noOfSensors $\times$ (noOfSensors + noOfAnchors), and the (p, q)th component of distanceMatrix0 indicates the distance between sensors p and q, or equivalently, between xMatrix0(:,p) and xMatrix0(:,q). Note that distanceMatrix0 is upper triangular; distanceMatrix0$(p, q) = 0$ if $p \geq q$.

```
>> distanceMatrix0
distanceMatrix0 =
  0   0.219579   0.414739   0.501646   0.756645          0          0
  0          0   0.356155   0.629003   0.471282          0          0
  0          0          0          0          0   0.647147   0.468894
```

Then, issue a command:

```
>>[xMatrix,distanceMatrix,info,pars] = SFSDPplus(sDim,noOfSensors,...
                  noOfAnchors,xMatrix0,distanceMatrix0,pars);
```

Then, the following is displayed on the screen.

```
## sDim = 2, noOfSensors = 3, noOfAnchors = 4
## the number of dist. eq. between two sensors  = 3
## the number of dist. eq. between a sensor & an anchor = 6
## the min., max. and ave. degrees over sensor nodes = 4, 4,   4.00
## +0.0000e+00 <= x(1) <= +1.0000e+00
## +0.0000e+00 <= x(2) <= +1.0000e+00
## the max. radio range = 6.7082e-01, the estimated noisy factor = 8.3330e-02

SFSDP --- A Sparse version of FSDP (Biswas and Ye)
Sunyoung Kim, Masakazu Kojima, Hayato Waki and Makoto Yamashita
Version 1.22, January 2010

## sDim = 2, noOfSensors = 3, noOfAnchors = 4
## pars: SDPsolver = sdpa, eps = 1.00e-07
## pars: sparseSW = 1, minDegree = 4, edgeSelectionSW = 1
## pars: objSW = 1, noisyFac = 8.3e-02, regTermFactor = 0.00
## the number of dist. eq. used in SFSDP between two sensors  = 3
## the number of dist. eq. used in SFSDP between a sensor & an anchor = 6
## the min., max. and ave. degrees over sensor nodes = 4, 4,   4.00
## elapsed time for generating an SDP relaxation problem =     0.09
-SeDuMi Wrapper for SDPA Start-
Note: pars information [4th argument] is not used
Free Variables are divided into positive and negative part of LP cone
Converted to SDPA internal data / Starting SDPA main loop
Converting optimal solution to Sedumi format
-SeDuMi Wrapper for SDPA End-
## elapsed time for retrieving an optimal solution =     0.01
## elapsed time for SDP solver =     0.07
## mean error in dist. eq. = 1.34e-02, max. error in dist. eq. = 7.29e-02
## rmsd = 6.21e-02
## see Figure 101
## elapsed time for a gradient method =     0.06
## mean error in dist. eq. = 1.57e-02, max. error in dist. eq. = 4.56e-02
```

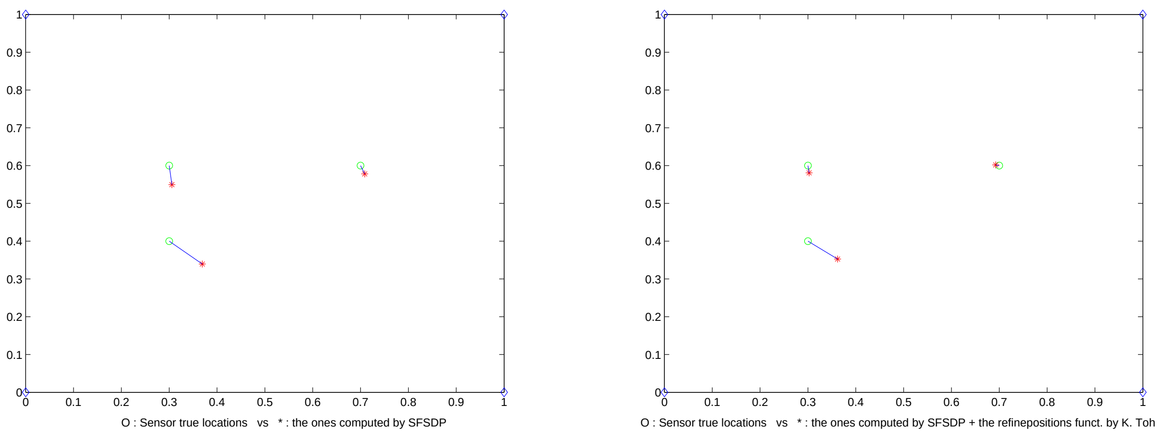


Figure 2: An example with three sensors and four anchors. Before and after the refinement using the gradient method.

```
## rmsd = 4.67e-02
## see Figure 103
```

Figure 2 is displayed at the end of execution. In Figure 2 and throughout, a circle indicates the true location of a sensor, $\star$ the computed location of a sensor, and a line segment a difference between the true and computed location. The input data and parameters of this example are stored in the file examples/example1.mat of the package, and can be loaded by

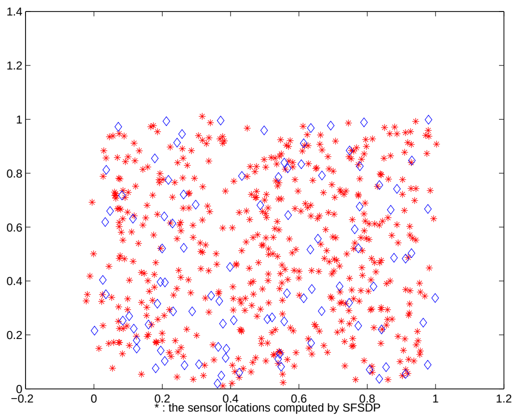
```
>> load example1.mat
```

instead of specifying them from the command window.

Next, we consider a 2-dimensional problem with 500 sensors and 100 anchors placed randomly in the region $[0, 1] \times [0, 1]$ and noisy distances. As in practical applications, we assume that the locations of the sensors are not known. For instance, suppose that xMatrix0 includes only 100 locations of anchors. To solve the problem, the following command can be used after loading the data stored in the file d2n01s500a100ns.mat, which is included in the directory examples of the package.

```
>> load d2n01s500a100ns.mat;
>>[xMatrix,distanceMatrix,info,pars] = SFSDPplus(sDim,noOfSensors,...
                  noOfAnchors,xMatrix0,distanceMatrix0,pars);
## only anchor locations are given
## sDim = 2, noOfSensors = 500, noOfAnchors = 100
## the number of dist. eq. between two sensors  = 8171
## the number of dist. eq. between a sensor & an anchor = 3000
## the min., max. and ave. degrees over sensor nodes = 24, 167,  38.68
## no location for sensors is given

SFSDP --- A Sparse version of FSDP (Biswas and Ye)
```



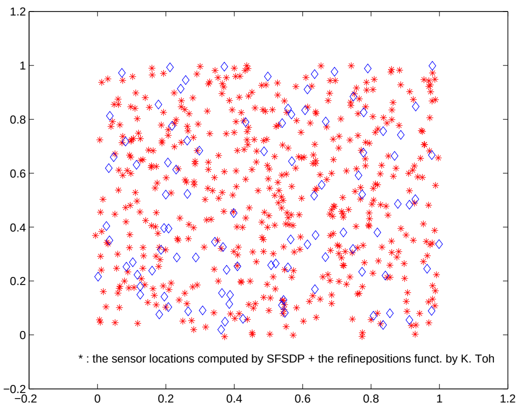


Figure 3: A 2-dimensional problem with 500 sensors (no information on their location) and 100 anchors and noisy distance. Before and after the refinement using the gradient method.

```
Sunyoung Kim, Masakazu Kojima, Hayato Waki and Makoto Yamashita
Version 1.22, January 2010

## sDim = 2, noOfSensors = 500, noOfAnchors = 100
## pars: SDPsolver = sdpa, eps = 1.00e-05
## pars: sparseSW = 1, minDegree = 4, edgeSelectionSW = 1
## pars: objSW = 1, noisyFac = 1.0e-01, regTermFactor = 0.00
## the number of dist. eq. used in SFSDP between two sensors  = 976
## the number of dist. eq. used in SFSDP between a sensor & an anchor = 3000
## the min., max. and ave. degrees over sensor nodes = 8, 112,   9.90
## elapsed time for generating an SDP relaxation problem =     0.52
-SeDuMi Wrapper for SDPA Start-
Note: pars information [4th argument] is not used
Free Variables are divided into positive and negative part of LP cone
Converted to SDPA internal data / Starting SDPA main loop
Converting optimal solution to Sedumi format
-SeDuMi Wrapper for SDPA End-
## elapsed time for retrieving an optimal solution =     0.07
## elapsed time for SDP solver =     1.66
## mean error in dist. eq. = 2.93e-04, max. error in dist. eq. = 9.63e-02
## see Figure 101
## elapsed time for a gradient method =     0.71
## mean error in dist. eq. = 1.99e-04, max. error in dist. eq. = 6.36e-02
## see Figure 103
```

Figure 3 is displayed at the end of execution. After obtaining a solution with SFSDP.m, SFSDPplus.m refines the solution using the function refineposition.m, which is a Matlab implementation of the gradient method provided by Prof. Kim-Chuan Toh. The figure on the right of Figure 3 is attained after applying the function.

We solve the same problem with information on the location of sensors to see how accurately the computed locations of sensors approximates the true locations of sensors. Note that we load d2n01s500a100.mat instead of d2n01s500a100ns.mat.

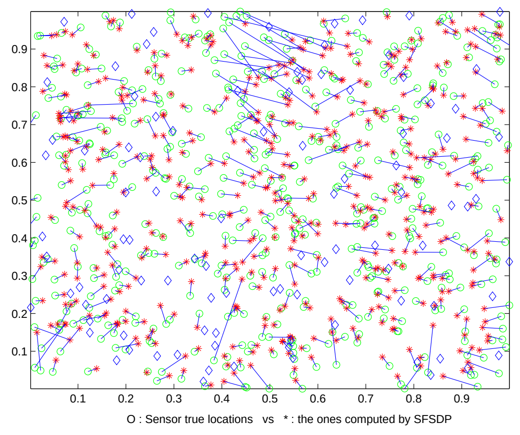
```
>> load d2n01s500a100.mat;
>>[xMatrix,distanceMatrix,info,pars] = SFSDPplus(sDim,noOfSensors,...
                  noOfAnchors,xMatrix0,distanceMatrix0,pars);
## sDim = 2, noOfSensors = 500, noOfAnchors = 100
## the number of dist. eq. between two sensors  = 8171
## the number of dist. eq. between a sensor & an anchor = 3000
## the min., max. and ave. degrees over sensor nodes = 24, 167,  38.68
## +1.5003e-03 <= x(1) <= +9.9912e-01
## +9.7480e-04 <= x(2) <= +9.9948e-01
## the max. radio range = 3.0000e-01, the estimated noisy factor = 9.9337e-02

SFSDP --- A Sparse version of FSDP (Biswas and Ye)
Sunyoung Kim, Masakazu Kojima, Hayato Waki and Makoto Yamashita
Version 1.22, January 2010

## sDim = 2, noOfSensors = 500, noOfAnchors = 100
## pars: SDPsolver = sdpa, eps = 1.00e-05
## pars: sparseSW = 1, minDegree = 4, edgeSelectionSW = 1
## pars: objSW = 1, noisyFac = 1.0e-01, regTermFactor = 0.00
## the number of dist. eq. used in SFSDP between two sensors  = 976
## the number of dist. eq. used in SFSDP between a sensor & an anchor = 3000
## the min., max. and ave. degrees over sensor nodes = 8, 112,   9.90
## elapsed time for generating an SDP relaxation problem =     0.38
-SeDuMi Wrapper for SDPA Start-
Note: pars information [4th argument] is not used
Free Variables are divided into positive and negative part of LP cone
Converted to SDPA internal data / Starting SDPA main loop
Converting optimal solution to Sedumi format
-SeDuMi Wrapper for SDPA End-
## elapsed time for retrieving an optimal solution =     0.08
## elapsed time for SDP solver =     1.66
## mean error in dist. eq. = 2.93e-04, max. error in dist. eq. = 9.63e-02
## rmsd = 2.27e-02
## see Figure 101
## elapsed time for a gradient method =     0.21
## mean error in dist. eq. = 1.99e-04, max. error in dist. eq. = 6.36e-02
## rmsd = 7.76e-03
## see Figure 103
```

Figure 4 is displayed at the end of execution.

We see that solving the same problem with and without the information on the location of sensors results in differences between Figures 3 and 4, and the two output displays.



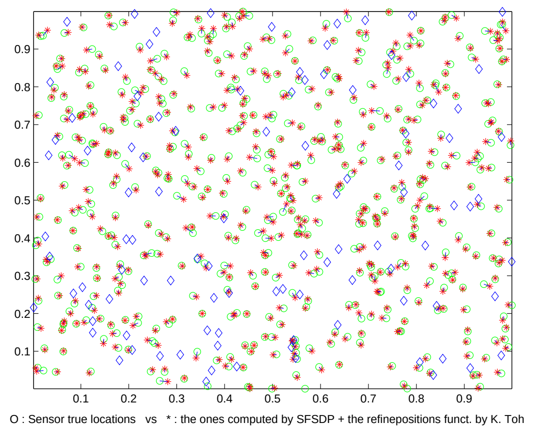


Figure 4: A 2-dimensional problem with 500 sensors (information available on their location) and 100 anchors and noisy distance. Before and after the refinement using the gradient method.

2.2 SFSDP.m

SFSDP.m can be called as follows with the same data as in the previous example. Notice that the output of SFSDP.m is different from SFSDPplus.m, in particular, no figures are shown at the end of execution.

```
>>  load d2n01s500a100.mat;
>>[xMatrix,distanceMatrix,info,pars] = SFSDP(sDim,noOfSensors,...
                   noOfAnchors,xMatrix0,distanceMatrix0,pars);

SFSDP --- A Sparse version of FSDP (Biswas and Ye)
Sunyoung Kim, Masakazu Kojima, Hayato Waki and Makoto Yamashita
Version 1.22, January 2010

## sDim = 2, noOfSensors = 500, noOfAnchors = 100
## pars: SDPsolver = sdpa, eps = 1.00e-05
## pars: sparseSW = 1, minDegree = 4, edgeSelectionSW = 1
## pars: objSW = 1, noisyFac = 1.0e-01, regTermFactor = 0.00
## the number of dist. eq. used in SFSDP between two sensors  = 976
## the number of dist. eq. used in SFSDP between a sensor & an anchor = 3000
## the min., max. and ave. degrees over sensor nodes = 8, 112,   9.90
## elapsed time for generating an SDP relaxation problem =     0.39
-SeDuMi Wrapper for SDPA Start-
Note: pars information [4th argument] is not used
Free Variables are divided into positive and negative part of LP cone
Converted to SDPA internal data / Starting SDPA main loop
Converting optimal solution to Sedumi format
-SeDuMi Wrapper for SDPA End-
## elapsed time for retrieving an optimal solution =     0.06
```

2.3 Generating a problem

For numerical experiments, users can generate a sensor network localization problem using the function generateProblem.m provided in the SFSDP package. After determining the values of parameter needed for generateProblem.m, the function generateProblem.m can be called. Then, it returns xMatrix0 and distanceMatrix0 as output. For example,

```
>> sDim = 2; noisyFac = 0.0; radiorange = 0.3; noOfSensors = 1000;
>> anchorType = 2; noOfAnchors = 100; randSeed = 2001;
>> [xMatrix0,distanceMatrix0] = generateProblem(sDim,noisyFac,...
                 radiorange,noOfSensors,anchorType,noOfAnchors,randSeed);
```

In addition, if users specify parameters such that

```
>> pars.free= 0; pars.eps= 1.0e-05; pars.minDegree= 4; pars.objSW = 0;
>> pars.noisyFac= 0.0;
```

they can solve the problem with the command

```
>>[xMatrix,distanceMatrix,info,pars] = SFSDPplus(sDim,noOfSensors,...
                 noOfAnchors,xMatrix0,distanceMatrix0,pars);
```

Or they can save the input data and parameters in a file such that

```
>> save('example2.mat','sDim','noOfSensors','noOfAnchors','xMatrix0',...
                  'distanceMatrix0','pars');
```

The description of input data and parameters in detail is given in Section 4.

2.4 test_SFSDP.m

The function test_SFSDP.m is included in the package SFSDP for numerical experiments. It can be used as

```
>> test_SFSDP(sDim,noisyFac,radiorange,noOfSensors,anchorType,...
         noOfAnchors,randSeed);
```

For a 2-dimensional problem with noisyFac = 0.3, radiorange=0.3, 500 sensors, anchorType=2, 100 anchors, and randomSeed=2009, which is the same problem as the second example in Section 2.1,

```
>> test_SFSDP(2,0.1,0.3,500,2,100,2009);
## elapsed time for generating a sensor network problem =     0.12
## sDim = 2, noOfSensors = 500, anchorType = 2, noOfAnchors = 100
## radiorange = 3.00e-01, noisyFac = 1.00e-01, randSeed = 2009
## the number of dist. eq. between two sensors  = 25397
## the number of dist. eq. between a sensor & an anchor = 10861
## the min., max. and ave. degrees over sensor nodes = 55, 174, 123.31
## sDim = 2, noOfSensors = 500, noOfAnchors = 100
## the number of dist. eq. between two sensors  = 25397
## the number of dist. eq. between a sensor & an anchor = 10861
## the min., max. and ave. degrees over sensor nodes = 55, 174, 123.31
## +3.0829e-03 <= x(1) <= +9.9940e-01
## +2.7934e-03 <= x(2) <= +9.9797e-01
## the max. radio range = 3.0000e-01, the estimated noisy factor = 9.9976e-02

SFSDP --- A Sparse version of FSDP (Biswas and Ye)
Sunyoung Kim, Masakazu Kojima, Hayato Waki and Makoto Yamashita
Version 1.22, January 2010

## sDim = 2, noOfSensors = 500, noOfAnchors = 100
## pars: SDPsolver = sdpa, eps = 1.00e-07
## pars: sparseSW = 1, minDegree = 4, edgeSelectionSW = 1
## pars: objSW = 1, noisyFac = 1.0e-01, regTermFactor = 0.00
## the number of dist. eq. used in SFSDP between two sensors  = 991
## the number of dist. eq. used in SFSDP between a sensor & an anchor = 3000
## the min., max. and ave. degrees over sensor nodes = 8, 129,   9.96
## elapsed time for generating an SDP relaxation problem =     0.39
-SeDuMi Wrapper for SDPA Start-
Note: pars information [4th argument] is not used
Free Variables are divided into positive and negative part of LP cone
Converted to SDPA internal data / Starting SDPA main loop
Converting optimal solution to Sedumi format
-SeDuMi Wrapper for SDPA End-
## elapsed time for retrieving an optimal solution =     0.06
## elapsed time for SDP solver =     1.76
## mean error in dist. eq. = 6.78e-05, max. error in dist. eq. = 4.13e-02
## rmsd = 3.00e-02
## see Figure 101
## elapsed time for a gradient method =     0.51
## mean error in dist. eq. = 6.63e-05, max. error in dist. eq. = 4.60e-02
## rmsd = 3.91e-03
## see Figure 103
```

The Figure 4 is displayed at the end.

For 3-dimensional problem, noisyFac = 0.1, radiorange=0.5, 500 sensors, anchorType=2, noOfAnchors=50, and randomSeed=2009, we issue a command:

```
 >> test_SFSDP(3,0.1,0.5,500,2,50,2009);
```

Then, on the screen the following is displayed.

```
## elapsed time for generating a sensor network problem =     0.11
## sDim = 3, noOfSensors = 500, anchorType = 2, noOfAnchors = 50
## radiorange = 5.00e-01, noisyFac = 1.00e-01, randSeed = 2009
## the number of dist. eq. between two sensors  = 33089
## the number of dist. eq. between a sensor & an anchor = 6885
## the min., max. and ave. degrees over sensor nodes = 41, 276, 146.13
## sDim = 3, noOfSensors = 500, noOfAnchors = 50
## the number of dist. eq. between two sensors  = 33089
## the number of dist. eq. between a sensor & an anchor = 6885
## the min., max. and ave. degrees over sensor nodes = 41, 276, 146.13
## +3.0829e-03 <= x(1) <= +9.9401e-01
## +2.7934e-03 <= x(2) <= +9.9940e-01
## +4.8920e-03 <= x(3) <= +9.9575e-01
## the max. radio range = 4.9999e-01, the estimated noisy factor = 9.9806e-02

SFSDP --- A Sparse version of FSDP (Biswas and Ye)
Sunyoung Kim, Masakazu Kojima, Hayato Waki and Makoto Yamashita
Version 1.22, January 2010

## sDim = 3, noOfSensors = 500, noOfAnchors = 50
## pars: SDPsolver = sdpa, eps = 1.00e-07
## pars: sparseSW = 1, minDegree = 5, edgeSelectionSW = 1
## pars: objSW = 1, noisyFac = 1.0e-01, regTermFactor = 0.00
## the number of dist. eq. used in SFSDP between two sensors  = 990
## the number of dist. eq. used in SFSDP between a sensor & an anchor = 3819
## the min., max. and ave. degrees over sensor nodes = 5, 168,  11.60
## elapsed time for generating an SDP relaxation problem =     0.48
-SeDuMi Wrapper for SDPA Start-
Note: pars information [4th argument] is not used
Free Variables are divided into positive and negative part of LP cone
Converted to SDPA internal data / Starting SDPA main loop
Converting optimal solution to Sedumi format
-SeDuMi Wrapper for SDPA End-
## elapsed time for retrieving an optimal solution =     0.07
## elapsed time for SDP solver =     2.24
## mean error in dist. eq. = 2.79e-04, max. error in dist. eq. = 1.49e-01
## rmsd = 6.60e-02
## see Figure 101
## elapsed time for a gradient method =     1.08
## mean error in dist. eq. = 1.81e-04, max. error in dist. eq. = 9.22e-02
## rmsd = 1.06e-02
## see Figure 103
```

Figure 5 is shown at the end of execution.

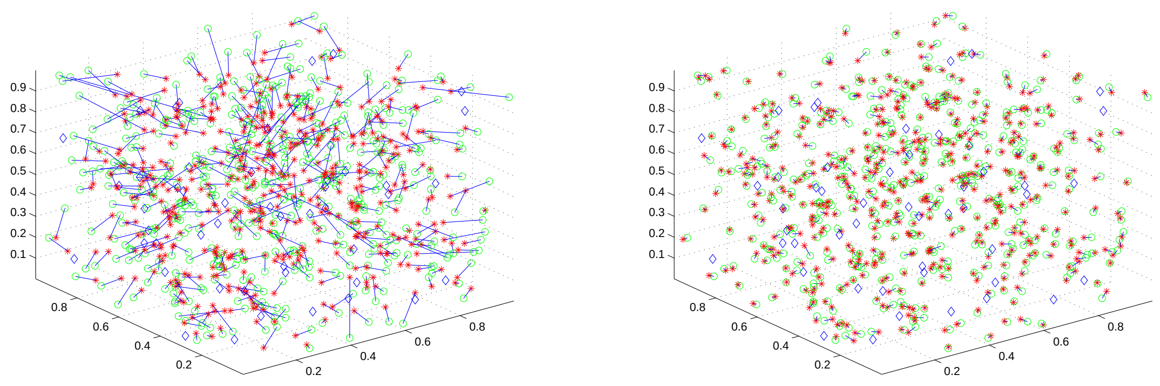


Figure 5: Before and after the refinement using the gradient method

3 Sample Run Using SeDuMi

There are two ways to call SeDuMi [18] from SFSDP instead of SDPA to solve SDP relaxation problems. As an example, we consider '2n01s500a100.mat'.

```
>> load d2n01s500a100.mat;
```

One way is to specify

```
>> pars.SDPsolver = 'sedumi';
```

The other way is to modify the MATLAB program SFSDP.m; replace the line

```
SDPsolverDefault = 'sdpa';
```

by

```
SDPsolverDefault = 'sedumi';
```

In both cases, we issue a command:

```
>>[xMatrix,distanceMatrix,info,pars] = SFSDPplus(sDim,noOfSensors,...
                    noOfAnchors,xMatrix0,distanceMatrix0,pars);
```

4 Input, Output and Parameters

4.1 Input

As we can see in the following commands,

Variable name	Description
sDim	The dimension of the space where sensors and anchors are located (2 or 3).
noOfSensors	The number m of sensors.
noOfAnchors	The number m_a of anchors located in the last m_a columns of xMatrix0.
xMatrix0	sDim$\times n$ matrix of the location of sensors and anchors in the sDim-dimensional space, where n is the total number of sensors and anchors, and anchors are placed in the last m_a columns. Or, sDim$\times m_a$ matrix of anchors in the sDim-dimensional space, where m_a denotes the number of anchors. If noOfAnchors = 0, then xMatrix0 can be [].
distanceMatrix0	The sparse (and noisy) distance matrix between sensors and anchors; distanceMatrix0(p,q) = (noisy) distance between a pair of sensors $(p,q) \in \mathcal{N}_x$ and distanceMatrix0(p,r) = (noisy) distance between a pair of sensor and an anchor $(p,r) \in \mathcal{N}_a$. See (2) and (3). Note that distanceMatrix0 is upper triangular, *i.e.*, distanceMatrix0$(p.q) = 0$ if $p >= q$.
pars	Control parameters in constructing an SDP relaxation problem and solving it by SeDuMi or SDPA. See Section 4.3 for more detail.

Table 1: Input for SFSDPplus.m and SFSDP.m

```
>>[xMatrix,distanceMatrix,info,pars] = SFSDPplus(sDim,noOfSensors,...
                  noOfAnchors,xMatrix0,distanceMatrix0,pars);
>>[xMatrix,distanceMatrix,info,pars] = SFSDP(sDim,noOfSensors,...
                  noOfAnchors,xMatrix0,distanceMatrix0,pars);
```

input arguments for SFSDPplus.m and SFSDP.m are sDim = the dimension of the space where sensors and anchors are placed, noOfSensors = the number of sensors, noOfAnchors = the number of anchors, xMatrix0 = the location matrix of sensors and anchors, distanceMatrix0 = the distance matrix, and pars involving some of parameters described in Table 1.

When using test_SFSDP.m as

```
>> test_SFSDP(sDim,noisyFac,radiorange,noOfSensors,anchorType,...
                   noOfAnchors,randSeed);
```

the required input is sDim = the dimension of the space where sensors and anchors are placed, noisyFac = noisy factor, radiorange = radio range, noOfSensors = the number of sensors, anchorType = anchor type, noOfAnchors = the number of anchors, and randSeed = a random seed. If sDim= 2, sensors and anchors will be located randomly in $[0,1] \times [0,1]$. If sDim= 3, sensors and anchors will be located randomly in $[0,1] \times [0,1] \times [0,1]$. If the value σ of noisyFac is 0, it means that the problem does not contain noise in distances. Otherwise, a value $\sigma > 0$ indicates that noise with the standard normal distribution $N(0,\sigma)$

exists in estimated distances. More precisely, noisy distance $\hat{d}_{pq}$ and $\hat{d}_{pr}$ are given such that

$$\begin{aligned} \hat{d}_{pq} &= \max\{(1+\xi_{pq}),\ 0.1\}d_{pq} \quad ((p,q)\in\mathcal{N}_x), \\ \hat{d}_{pr} &= \max\{(1+\xi_{pr}),\ 0.1\}d_{pr} \quad ((p,r)\in\mathcal{N}_a). \end{aligned}$$

Here ξ_{pq} and ξ_{pr} denote random numbers chosen from the standard normal distribution $N(0,\sigma)$, d_{pq} the true distance between sensors p and q, and d_{pr} the true distance between sensor p and anchor r. The 4th argument noOfSensors in the input field of test_SFSDP.m is the number of sensors. A value for anchorType decides how anchors are located as shown in Table 2. The 6th argument noOfAnchors of input is the number of anchors, and the 7th

AnchorType	Position
0	Anchors placed at the grid points on the boundary and interior of $[0,1]^{\mathrm{sDim}}$
1	Anchors placed at the grid points in the interior of $[0,1]^{\mathrm{sDim}}$
2	Anchors placed randomly in $[0,1]^{\mathrm{sDim}}$
3	sDim+1 anchors on the origin and the coordinate axis
4	sDim+1 anchors near the center
10	No anchor

Table 2: Types of anchors

argument randSeed is a random seed for a random distribution of sensors and anchors if anchorType = 2. For instance,

```
>> test_SFSDP(2,0.0,0.2,500,0,4,2009);
```

The above command has input of the dimension of the space = 2, noisy factor 0.0 (i.e., no noise), radio range = 0.2, the number of sensors = 500, anchor type = 0, the number of anchors = 4, and random seed = 2009.

4.2 Output

As we have seen so far, the common output arguments of SFSDP and SFSDPplus.m are xMatrix, info, pars and distanceMatrix. Among these arguments, xMatrix, info and distanceMatrix are explained in Table 3. The last argument pars involves control parameters used in constructing an SDP relaxation problem and solving it by SeDuMi or SDPA. See Section 4.3 for more detail.

4.3 Parameters

The parameters for SeDuMi, SDPA, SFSDPplus.m, and SFSDP.m are provided in the fields of pars as shown in Table 4.

5 Numerical Results

We report some numerical results to show how large problems SFSDP can solve using SDPA or SeDuMI. Numerical experiments were performed on 2×2.8GHz Quad-Core Intel Xeon with 4GB memory. In Table 5 and 6, "time for building SDP" denotes the elapsed time for building the SDP relaxation problem in seconds, "SDP.time" the elapsed time for solving the SDP relaxation problem by SDPA or SeDuMi and "rmsd" the root mean square distance

$$\left(\frac{1}{n} \sum_{p=1}^{n} \|\boldsymbol{x}_p - \hat{\boldsymbol{x}}_p\|^2 \right)^{1/2},$$

where $\boldsymbol{x}_p$ and $\hat{\boldsymbol{x}}_p$ denote the true and computed location of the sensor p.

6 Concluding Remarks

We have described the structure and usage of the Matlab package SFSDP.

The sensor network localization problem has a number of applications where computational efficiency is an important issue. SDP approach has been known to be effective in locating sensors, however, solving large-scale problems with this approach has been a challenge.

From numerical results in [12], SFSDP demonstrates computational advantages over other methods. These come from utilizing the aggregated and correlative sparsity of the problem, which reduces the size of SDP relaxation. We hope to improve the performance of SDP relaxation, in particular, for the case when the original problem does not provide enough distance information between sensors.

Acknowledgments

The authors would like to thank Professor Yinyu Ye for the original version of FSDP, and Professor Kim Chuan Toh for Matlab programs refineposition.m and procrustes.m, and for helpful comments.

References

[1] A. Y. Alfakih, A. Khandani, and H. Wolkowicz (1999) "Solving Euclidean matrix completion problem via semidefinite programming," *Comput. Opt. and Appl.*, **12**, 13-30.

[2] P. Biswas and Y. Ye (2004) "Semidefinite programming for ad hoc wireless sensor network localization," in *Proceedings of the third international symposium on information processing in sensor networks*, ACM press, 46-54.

[3] P. Biswas and Y. Ye (2006) "A distributed method for solving semidefinite programs arising from Ad Hoc Wireless Sensor Network Localization," in *Multiscale Optimization Methods and Applications*, 69-84, Springer.

[4] P. Biswas, T.-C. Liang, T.-C. Wang, Y. Ye (2006) "Semidefinite programming based algorithms for sensor network localization," *ACM Transaction on Sensor Networks*, **2**, 188-220.

[5] P. Biswas, T.-C. Liang, K.-C. Toh, T.-C. Wang, and Y. Ye (2006) "Semidefinite programming approaches for sensor network localization with noisy distance measurements," *IEEE Transactions on Automation Science and Engineering*, **3**, pp. 360–371.

[6] L. Doherty, K. S. J. Pister, and L. El Ghaoui (2001) "Convex position estimation in wireless sensor networks," *Proceedings of 20th INFOCOM*, **3**, 1655-1663.

[7] T. Eren, D. K. Goldenberg, W. Whiteley, Y. R. Wang, A. S. Morse, B. D. O. Anderson (2004), and P. N. Belhumeur "Rigidity, computation, and randomization in network localization," *in Proceedings of IEEE Infocom.*

[8] K. Fujisawa, M. Fukuda, K. Kobayashi, M. Kojima, K. Nakata, M. Nakata and M. Yamashita (2008), "SDPA (SemiDefinite Programming Algorithm) User's Manual — Version 7.0.5," Research Report B-448, Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology, Oh-Okayama, Meguro, Tokyo 152-8552, Japan.

[9] M. Fukuda, M. Kojima, K. Murota and K. Nakata (2000) "Exploiting sparsity in semidefinite programming via matrix completion I: General framework," *SIAM J. on Optimi.*, **11**, 647-674.

[10] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S.Wicker (2002) "An empirical study of epidemic algorithms in large scale multihop wireless network," March.

[11] A. Howard, M. Matarić and G. Sukhatme (2001) "Relaxation on a mesh: a formalism for generalized localization," In *IEEE/RSJ International conference on intelligent robots and systems*, Wailea, Hawaii, 1055-1060.

[12] S. Kim, M. Kojima and H. Waki (2009) " Exploiting sparsity in SDP relaxation for sensor network localization," *SIAM J. on Optim.*, **20**, (1) 192-215.

[13] K. Kobayashi, S. Kim and M. Kojima, (2008) Correlative sparsity in primal-dual interior-point methods for LP, SDP and SOCP, *Applied Mathematics and Optimization,* **58** (1) 69-88.

[14] K. Nakata, K. Fujisawa, M. Fukuda, M. Kojima and K. Murota (2003) "Exploiting sparsity in semidefinite programming via matrix completion II: Implementation and numerical results," *Mathematical Programming*, **95**, 303-327.

[15] J. Nie (2009) "Sum of squares method for sensor network localization," *Comput. Opt. and Appl.,* 43, No. 2, 151-179.

[16] SeDuMi Homepage, http://sedumi.mcmaster.ca

[17] SDPA Homepage, http://sdpa.indsys.chuo-u.ac.jp/sdpa/

[18] J. F. Strum, "SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones," *Optimization Methods and Software*, **11 & 12** (1999) 625-653.

[19] P. Tseng, (2007) "Second order cone programming relaxation of sensor network localization," *SIAM J. on Optim.*, **18**, 156-185.

[20] Z. Wang, S. Zheng, S. Boyd, and Y. Ye (2008) "Further relaxations of the SDP approach to sensor network localization," *SIAM J. on Optim.*, **19** (2) 655-673.

xMatrix	sDim $\times$ n matrix of the location of sensors and anchors computed in the sDim dimensional space, where n is the total number of sensors and anchors, and anchors are placed in the last m_a columns.
info	information on execution of SFSDP and/or SFSDPplus, which includes eTimeBuildSDP : elapsed time for building the SDP problem, eTimeSolveSDP : elapsed time spent in the SDP solver, eTimeConvSDP : elapsed time for conversion, eTimeAddBounds : elapsed time for adding bounds, eTimeRetSolution : elapsed time for retrieving sensors' locations. eTimeGradMethod : elapsed time for a refinement of the SDP solution by the gradient method, "info" also includes information from SeDuMi or SDPA output. See SeDuMi user guide [16] or SDPA user guide [8].
distanceMatrix	The distance matrix used in the construction of an SDP whose description is similar to that of input distnceMatrix0 given in Table 1. More precisely, the output values represent the distances d_{pq} $((p,q) \in \mathcal{N}_x)$ and d_{pr} $((p,r) \in \mathcal{N}_a)$ in the problem (2) (or the noisy distances $\hat{d}_{pq}$ $((p,q) \in \mathcal{N}_x)$ and $\hat{d}_{pr}$ $((p,r) \in \mathcal{N}_a)$ in the problem (3)).

Table 3: Output of SFSDP.m and SFSDPplus.m

Parameters to choose an SDP solver	
pars.SDPsolver	= 'sdpa' to apply SDPA (default). = 'sedumi' to apply SeDuMi.
Parameters for SeDuMi	
pars.eps, pars.free, pars.fid	See SeDuMi user guide [16].
Parameters for SFSDP.m	
pars.minDegree	A positive integer greater than sDim, which is used for selecting subsets $\mathcal{N}'_x$ and $\mathcal{N}'_a$ from $\mathcal{N}_x$ and $\mathcal{N}_a$ to reduce the size of the problem (2) or (3). If it is increased, a stronger relaxation but longer cpu time is expected. If it is equal to or larger than 100, then no reduction is conducted. The default value is sDim + 2. See Section 4.1 of [12] for more details.
pars.objSW	= 0 to solve the noise-free problem (2). = 1 to solve the problem (3) involving noise. = 2 to solve the noise-free problem (2) with no anchor small number of anchors; a regularization term is minimized subject to the constraint of the noise-free problem (2). = 3 to solve a problem involving noise with no anchor or a small number of anchors; a regularization term is added to the objective function of (3).
pars.noisyFac	= [] if noisyFac σ is not specified or unknown. = σ if noisyFac σ is known; used to bound the error ϵ^+_{pq} and ϵ^-_{pq}.
Parameters for SFSDPplus.m	
pars.analyzeData	= 1 to analyze the input data (default). = 0 no information on the input data.

Table 4: Parameters

#Sensors	Time for	Using SDPA		Using SeDuMi		Time for
	building SDP	SDP.time	rmsd	SDP.time	rmsd	gradient method
1000	0.4	16.4	6.7e-03	38.4	6.7e-03	2.3
2000	1.2	43.3	5.4e-03	126.1	5.3e-03	9.1
4000	3.9	63.3	7.9e-03	316.2	1.5e-02	17.1
6000	8.9	102.8	6.3e-03	1061.8	6.9e-03	24.4

Table 5: Numerical results on 2-dimensional problems with randomly generated n sensors in $[0,1] \times [0,1]$, 4 anchors at the corner of $[0,1] \times [0,1]$, radiorange = 0.1, and noisyFac = 0.1

#Sensors	Time for building SDP	Using SDPA		Using SeDuMi		Time for gradient method
		SDP.time	rmsd	SDP.time	rmsd	
1000	0.5	28.1	2.1e-02	56.1	2.5e-02	9.7
2000	1.4	50.8	2.7e-02	138.1	2.7e-02	5.1
3000	2.6	56.3	2.7e-02	246.6	2.7e-02	7.1
4000	4.1	75.4	2.6e-02	294.1	2.6e-02	8.3

Table 6: Numerical results on 3-dimensional problems with randomly generated n sensors in $[0,1]^3$, 8 anchors at the corner of $[0,1]^3$, radiorange = 0.3, and noisyFac = 0.1