ISSN 1342-2804

Research Reports on Mathematical and Computing Sciences

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

Sunyoung Kim, Masakazu Kojima, and Hayato Waki

August 2008, B–449

Department of Mathematical and Computing Sciences
Tokyo Institute of Technology

SERIES B: Operations Research

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[†], and Hayato Waki[‡]

August 2008

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 an 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 some typical anchor locations. The function test_SFSDP.m is for numerical experiments on SFSDPplus.m applied to 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 Women's University, 11-1 Dahyun-dong, Sudaemoon-gu, Seoul 120-750 Korea. S. Kim's research was supported by KRF 2007-313-C00089. *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*

1 Introduction

A sensor network localization problem is to locate m sensors that fit the distances when a subset of distances and some sensors of known position (called anchors) are provided in a sensor network of n sensors, where $n > m$. Various approaches [1, 6, 7, 9, 10, 17] 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, 14, 18].

We introduce a Matlab package SFSDP for solving sensor network localization problems by SDP relaxation. The main function SFDPS.m of the package is an implementation of the SDP relaxation proposed in the recent paper by Kim, Kojima and Waki [11]. SFSDP.m is intended to improve the efficiency of Biswas and Ye's FSDP [2] by exploiting sparsity, the aggregated and correlative sparsity [8, 13, 12], 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 4000 sensors in 2-dimensional case, than FSDP.

SFSDP.m can solve the problem with exact and noisy distance. 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 anchors. Let $\rho > 0$ be a radio range, which determines the set $\widetilde{\mathcal{N}}_x$ of pairs of sensors p and q such that their unknown (Euclidean) distance d_{pq} is not larger than ρ, and the set $\widetilde{\mathcal{N}}_a$ of pairs of a sensor p and an anchor r such that their distance d_{pr} is not longer than ρ;

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

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 $\widetilde{\mathcal{N}}_x$ and and $\mathcal{N}_a$ a subset of $\widetilde{\mathcal{N}}_a$. 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 distance.

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

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

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

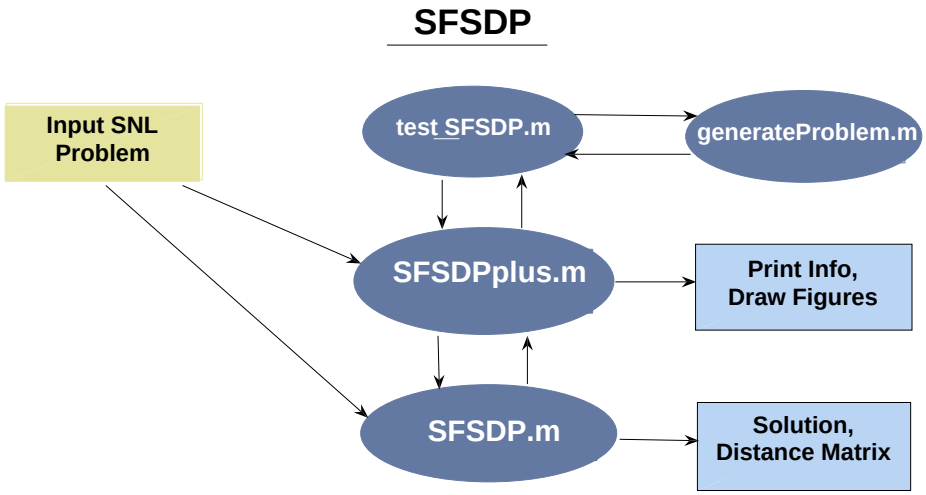

Figure 1: The structure of SFSDP

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

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 in solving a large scale sensor network localization problem efficiently by SFSDP.m. See Section 4.1 of [11] for more details. We assume that either (i) (noisy) distance 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 sparsity (the aggregated and correlative sparsity) or (ii) the original problem itself is sparse. If we take $\widetilde{\mathcal{N}}_x$ and $\widetilde{\mathcal{N}}_a$ (or their subsets large enough) for $\mathcal{N}_x$ and $\mathcal{N}_a$, respectively, the assumption (i) is usually satisfied. It should be remarked, 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 [18] 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 [11].

The structure of the package SFSDP is shown in Figure 1. Besides 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 SeDuMi [16], which is available at [15], to solve SDP relaxation problems.

The other two functions generateProblem.m and test_SFDP.m are for users interested in numerical experiment 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 input and output for the functions SFSDP.m, SFSDPplus.m, generateProblem.m and test_SFSDP.m in detail in Section 3.

2 Sample Run

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 with 3 sensors and 4 anchors is generated with the following xMatrix0 and distanceMatrix0. 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 $(1, 1)$. Input data and parameters are prepared 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 elements of xMatrix0 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 location of sensors, can be omitted for general case with unknown location of sensors.

The corresponding distanceMatrix0 has the following values. That is, nonzero (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    0.2000    0.4472    0.5000         0    0.8062         0
         0         0    0.4000         0    0.5000         0         0
         0         0         0         0         0    0.6708    0.5000
```

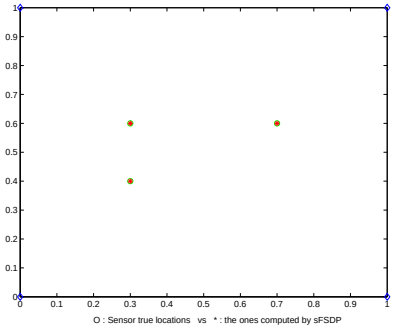

Figure 2: An example with three sensors and four anchors.

Then, issue a command:

```
>> [xMatrix,info] = SFSDPplus(sDim,noOfSensors,noOfAnchors,...
                    xMatrix0,distanceMatrix0,pars);
## sDim = 2, noOfSensors = 3, noOfAnchors = 4
## the number of dist. equations between two sensors  = 3
## the number of dist. equations between a sensor & an anchor = 5
## the minimum, maximum and average degrees over sensor nodes = 3, 4,   3.67
## +0.0000e+00 <= x(1) <= +1.0000e+00
## +0.0000e+00 <= x(2) <= +1.0000e+00
## the max. radio range = 8.0623e-01, the estimated noisy factor = 2.0250e-05

SFSDP --- A Sparse version of FSDP (Biswas and Ye)
Sunyoung Kim, Masakazu Kojima and Hayato Waki
Version 1.01, July 28, 2008

## pars: eps = 1.00e-05, free = 0, minDegree = 4, objSW = 1, noisyFac = 0
## the number of dist. equations used in SFSDP between two sensors  = 3
## the number of dist. equations used in SFSDP between a sensor & an anchor = 5
## the minimum, maximum and average degrees over sensor nodes = 3, 4,   3.67
## cpu time for generating an SDP relaxation problem =     0.03
## cpu time for retrieving an optimal solution =     0.00
## cpu time for SeDuMi =     0.18
## mean error in dist. eq. = 2.64e-06, max. error in dist. eq. = 2.11e-05
## rmsd = 1.98e-05
## see Figure 101
## cpu time for a gradient method =     0.04
## mean error in dist. eq. = 2.86e-06, max. error in dist. eq. = 1.27e-05
## rmsd = 1.82e-05
## 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 are stored in the file examples/example1.mat of the package, and can be loaded as

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

instead of specifying them from the command window.

Now consider a 2-dimensional problem with 500 sensors and 100 anchors placed randomly in the region $[0, 1] \times [0, 1]$ and noisy distance. As in practical applications, we assume that the location of sensors is not known. Thus, 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,info] = SFSDPplus(sDim,noOfSensors,noOfAnchors,...
                     xMatrix0,distanceMatrix0,pars);
## only anchor locations are given
## sDim = 2, noOfSensors = 500, noOfAnchors = 100
## the number of dist. equations between two sensors  = 8171
## the number of dist. equations between a sensor & an anchor = 3000
## the minimum, maximum and average degrees over sensor nodes = 24, 167,  38.68
## no location for sensors is given

SFSDP --- A Sparse version of FSDP (Biswas and Ye)
Sunyoung Kim, Masakazu Kojima and Hayato Waki
Version 1.01, July 28, 2008

## pars: eps = 1.00e-05, free = 0, minDegree = 4, objSW = 1, noisyFac = 1.00e-01
## the number of dist. equations used in SFSDP between two sensors  = 989
## the number of dist. equations used in SFSDP between a sensor & an anchor = 1500
## the minimum, maximum and average degrees over sensor nodes = 5, 138,   6.96
## cpu time for generating an SDP relaxation problem =     4.03
## cpu time for retrieving an optimal solution =     0.11
## cpu time for SeDuMi =    10.26
## mean error in dist. eq. = 1.30e-03, max. error in dist. eq. = 8.51e-02
## see Figure 101
## cpu time for a gradient method =     3.62
## mean error in dist. eq. = 1.25e-03, max. error in dist. eq. = 6.18e-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 a gradient method provided by Prof. Kim-Chuan Toh. The figure on the right-hand-side of Figure 3 is attained after applying the function.

Now we solve the same problem with information on the location of sensors to see how accurately the computed location of sensors approximates the true location of sensors.

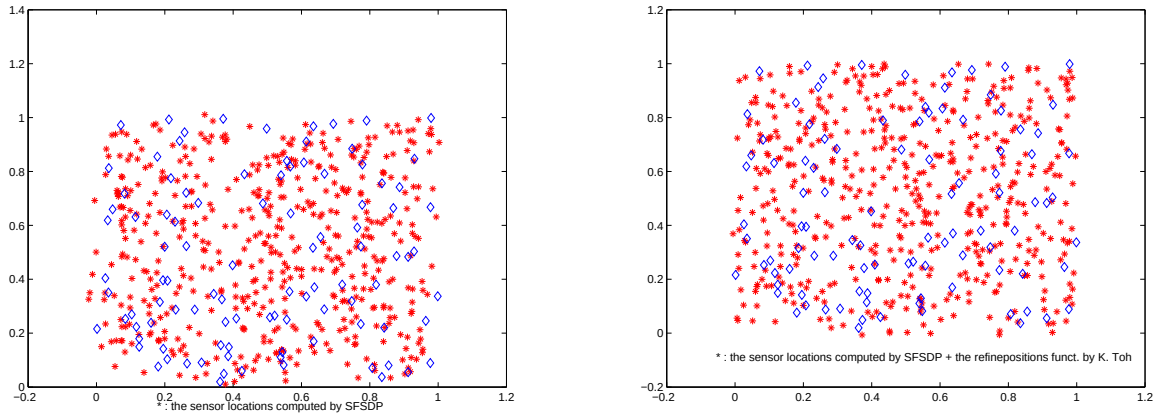

Figure 3: A 2-dimensional problem with 500 sensors (no information on their location) and 100 anchors and noisy distance. Before and after the refinement using a gradient method.

```
>> load 'd2n01s500a100.mat';
>> [xMatrix,info] = SFSDPplus(sDim,noOfSensors,noOfAnchors,...
                  xMatrix0,distanceMatrix0,pars);
## sDim = 2, noOfSensors = 500, noOfAnchors = 100
## the number of dist. equations between two sensors  = 8171
## the number of dist. equations between a sensor & an anchor = 3000
## the minimum, maximum and average 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 and Hayato Waki
Version 1.01, July 28, 2008

## pars: eps = 1.00e-05, free = 0, minDegree = 4, objSW = 1, noisyFac = 1.00e-01
## the number of dist. equations used in SFSDP between two sensors  = 989
## the number of dist. equations used in SFSDP between a sensor & an anchor = 1500
## the minimum, maximum and average degrees over sensor nodes = 5, 138,   6.96
## cpu time for generating an SDP relaxation problem =     5.18
## cpu time for retrieving an optimal solution =     0.19
## cpu time for SeDuMi =    15.42
## mean error in dist. eq. = 1.30e-03, max. error in dist. eq. = 8.51e-02
## rmsd = 4.38e-02
## see Figure 101
## cpu time for a gradient method =     5.08
## mean error in dist. eq. = 1.25e-03, max. error in dist. eq. = 6.18e-02
## rmsd = 7.76e-03
## see Figure 103
```

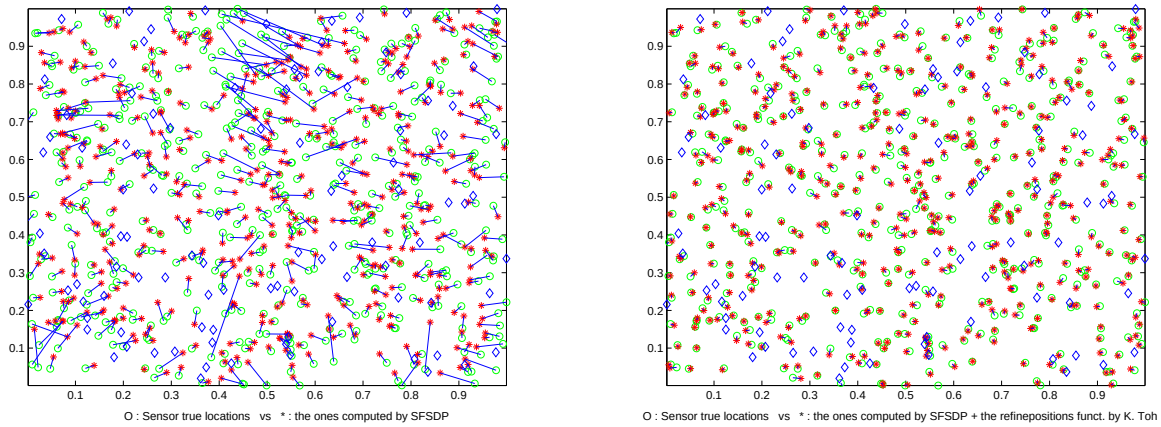

Figure 4: A 2-dimensional problem with 500 sensors (information available on their location) and 100 anchors and noisy distance. Before and after the refinement using a gradient method.

Figure 4 is displayed at the end of execution.

We note the difference in output and Figures 3 and 4 obtained from solving the same problem with and without the information on the location of sensors.

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,info,distanceMatrix] = SFSDP(sDim,noOfSensors,noOfAnchors,...
                   xMatrix0,distanceMatrix0,pars);

SFSDP --- A Sparse version of FSDP (Biswas and Ye)
Sunyoung Kim, Masakazu Kojima and Hayato Waki
Version 1.01, July 28, 2008

## pars: eps = 1.00e-05, free = 0, minDegree = 4, objSW = 1, noisyFac = 1.00e-01
## the number of dist. equations used in SFSDP between two sensors  = 989
## the number of dist. equations used in SFSDP between a sensor & an anchor = 1500
## the minimum, maximum and average degrees over sensor nodes = 5, 138,   6.96
## cpu time for generating an SDP relaxation problem =     5.20
## cpu time for retrieving an optimal solution =     0.17
```

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,info] = 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');
```

We will describe each of input data and parameters in detail in Section 3.

2.4 test_SFSDP.m

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

```
>> 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=2001, which is the same problem as the second example in Section 2.1,

```
>> test_SFSDP(2,0.1,0.3,500,2,100,2001);
## cpu time for generating a sensor network problem =     1.10
## sDim = 2, noOfSensors = 500, anchorType = 2, noOfAnchors = 100
## radiorange = 3.00e-01, noisyFac = 1.00e-01, randSeed = 2001
```

```
## the number of dist. equations between two sensors  = 8171
## the number of dist. equations between a sensor & an anchor = 3000
## the minimum, maximum and average degrees over sensor nodes = 24, 167,  38.68

SFSDP --- A Sparse version of FSDP (Biswas and Ye)
Sunyoung Kim, Masakazu Kojima and Hayato Waki
Version 1.01, July 28, 2008

## pars: eps = 1.00e-05, free = 0, minDegree = 4, objSW = 1, noisyFac = 1.00e-01
## the number of dist. equations used in SFSDP between two sensors  = 989
## the number of dist. equations used in SFSDP between a sensor & an anchor = 1500
## the minimum, maximum and average degrees over sensor nodes = 5, 138,   6.96
## cpu time for generating an SDP relaxation problem =     5.05
## cpu time for retrieving an optimal solution =     0.18
## cpu time for SeDuMi =    15.11
## mean error in dist. eq. = 1.30e-03, max. error in dist. eq. = 8.51e-02
## rmsd = 4.38e-02
## see Figure 101
## cpu time for a gradient method =     5.14
## mean error in dist. eq. = 1.25e-03, max. error in dist. eq. = 6.18e-02
## rmsd = 7.76e-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=2001, we issue a command:

```
 >> test_SFSDP(3,0.1,0.5,500,2,50,2001);
```

Then, on the screen the following is displayed.

```
>> test_SFSDP(3,0.1,0.5,500,2,50,2001);
## cpu time for generating a sensor network problem =     0.38
## sDim = 3, noOfSensors = 500, anchorType = 2, noOfAnchors = 50
## radiorange = 5.00e-01, noisyFac = 1.00e-01, randSeed = 2001
## the number of dist. eq. between two sensors  = 11015
## the number of dist. eq. between a sensor & an anchor = 3831
## the min., max. and ave. degrees over sensor nodes = 24, 259,  51.72

SFSDP --- A Sparse version of FSDP (Biswas and Ye)
Sunyoung Kim, Masakazu Kojima and Hayato Waki
Version 1.01, July 28, 2008

## pars: eps = 1.00e-05, free = 0, minDegree = 5, objSW = 1, noisyFac = 1.00e-01
## the number of dist. eq. used in SFSDP between two sensors  = 993
## the number of dist. eq. used in SFSDP between a sensor & an anchor = 1997
```

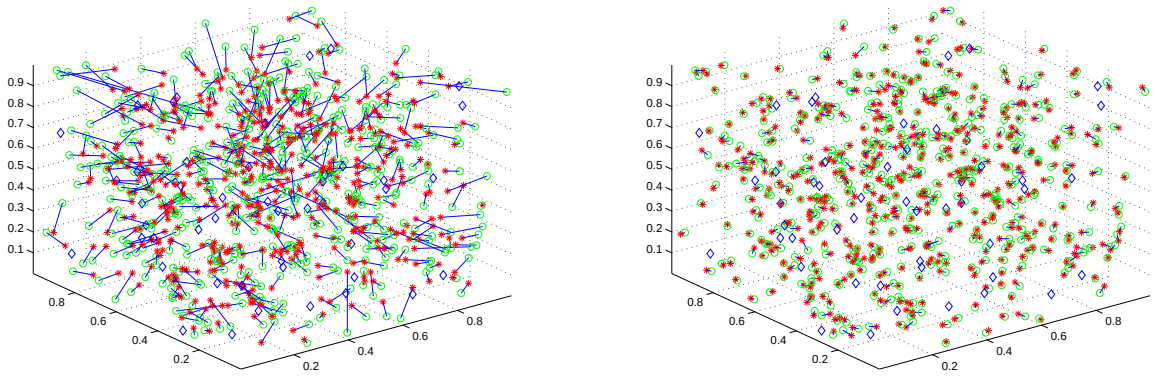

Figure 5: Before and after the refinement using a gradient method

```
## the min., max. and ave. degrees over sensor nodes = 5, 176,   7.97
## cpu time for generating an SDP relaxation problem =     3.45
## cpu time for retrieving an optimal solution =     0.13
## cpu time for SeDuMi =     9.03
## mean error in dist. eq. = 5.82e-03, max. error in dist. eq. = 1.88e-01
## rmsd = 9.83e-02
## see Figure 101
## cpu time for a gradient method =     6.39
## mean error in dist. eq. = 3.33e-03, max. error in dist. eq. = 1.06e-01
## rmsd = 1.91e-02
## see Figure 103
```

Figure 5 is shown at the end of execution.

3 Input, Output and Parameters

3.1 Input

As we can see in the following commands,

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

input for SFSDPplus.m and SFSDP.m is the dimension of the space where sensors and anchors are placed, the number of sensors, the number of anchors, the location matrix of sensors and anchors, and the distance matrix, pars involving some of parameters, which are described in Table 1.

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. See Section 3.3 for more detail.

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

When using test_SFSDP.m as

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

the required input is the dimension of the space where sensors and anchors are placed, noisy factor, radio range, the number of sensors, anchor type, and the number of anchors, and a random seed. The dimension of the space is called sDim. If sDim= 2, sensors and anchors will be located in $[0,1] \times [0,1]$. If sDim= 3, sensors and anchors will be located 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 distance. Otherwise, a value $\sigma > 0$ indicates that noise with the standard normal distribution $N(0,\sigma)$ exists in estimated distance. More precisely, noisy distance $\hat{d}_{pq}$ and $\hat{d}_{pr}$ are given such that

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

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. All sensors are placed in $[0,1]^{\mathrm{sDim}}$ randomly. The 4th argument noOfSensors in 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 randSeed is a random seed for a random distribution of sensors and anchors if anchorType = 2. For instance,

AnchorType	Position
0	anchors placed at grid points on the boundary and interior of $[0,1]^{\text{sDim}}$
1	anchors placed at grid points in the interior of $[0,1]^{\text{sDim}}$
2	anchors placed randomly in $[0,1]^{\text{sDim}}$
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

```
>> test_SFSDP(2,0.0,0.2,500,0,4,2001);
```

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 = 2001.

3.2 Output

As we can see in the following commands,

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

the output of SFSDPplus.m is xMatrix and info, which are described in Table 3.

xMatrix	sDim × 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	info from SeDuMi output. See SeDuMi user guide [15].

Table 3: Output of SFSDPplus.m

The output of SFSDP.m is xMatrix, info, as SFSDPplus.m, and distanceMatrix.

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

The description of output distanceMatrix is similar to that of input distnceMatrix0 given in Table 1. However, some values of the output distanceMatrix differs from the corresponding values of the input distanceMatrix0. 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)). As we mentioned in the Introduction, SFSDP.m replaces $\mathcal{N}_x$ and $\mathcal{N}_a$ by subsets of them, $\mathcal{N}'_x$ and $\mathcal{N}'_a$, respectively, to

reduce the size of the problem and extract sparsity from the problem. The values of output distanceMatrix represent the distances d_{pq} ($(p,q) \in \mathcal{N}'_x$) and d_{pr} ($(p,r) \in \mathcal{N}'_a$) (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 reduced problem. Thus,

$$\begin{aligned}
\text{distanceMatrix}(p,q) &= \text{distanceMatrix0}(p,q) > 0 \\
&\quad \text{if distanceMatrix}(p,q) > 0 \text{ or } (p,q) \in \mathcal{N}'_x, \\
\text{distanceMatrix}(p,q) &= 0 \text{ if } (p,q) \in \mathcal{N}_x \backslash \mathcal{N}'_x, \\
\text{distanceMatrix}(p,r) &= \text{distanceMatrix0}(p,r) > 0 \\
&\quad \text{if distanceMatrix}(p,r) > 0 \text{ or } (p,r) \in \mathcal{N}'_a, \\
\text{distanceMatrix}(p,r) &= 0 \text{ if } (p,r) \in \mathcal{N}_a \backslash \mathcal{N}'_a.
\end{aligned}$$

3.3 Parameters

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

4 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 [11], 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 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.

Parameters for SeDuMi	
pars.eps, pars.free, pars.fid	See SeDuMi user guide [15].
Parameters for SFSDP.m	
pars.minDegree	A positive integer greater than sDim, which is used in choosing 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 done. The default value is sDim + 2. See Section 4.1 of [11] for more details.
pars.objSW	= 0 if #anchors $\geq$ sDim+1 and no noise to solve the problem (2). = 1 if #anchors $\geq$ sDim+1 and noise to solve the problem (3). = 2 if #anchors = 0 and no noise to minimize a regularization term subject to the constraint of the problem (2). = 3 if #anchors = 0 and noise to solve the problem (3) with an additional regularization term in its objective function.
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.
pars.moreDistanceSW	= 1 to add all distances between sensors and anchors within the radio range before applying FSDP.m. This option is valid only when locations of sensors are given. = 0 to solve the given problem (default).

Table 4: Parameters

[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, and P. N. Belhumeur (2004) "Rigidity, computation, and randomization in network localization," *in Proceedings of IEEE Infocom.*

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

[9] 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.

[10] 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.

[11] S. Kim, M. Kojima and H. Waki (2008) " Exploiting sparsity in SDP relaxation for sensor network localization," Research Report B-447, Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology, Oh-Okayama, Meguro, Tokyo 152-8552, Japan.

[12] K. Kobayashi, S. Kim and M. Kojima, Correlative sparsity in primal-dual interior-point methods for LP, SDP and SOCP, to appear in *Applied Mathematics and Optimization.*

[13] 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.

[14] J. Nie (2006) "Sum of squares method for sensor network localization," preprint.

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

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

[17] P. Tseng, (2007) "Second order cone programming relaxation of sensor network localization," to appear in *SIAM Journal on Optimization*.

[18] Z. Wang, S. Zheng, S. Boyd, and Y. Ye (2007) "Further relaxations of the SDP approach to sensor network localization," preprint.