HDSDP 2.0

Software for Semidefinite Programming

User Manual

December 5, 2024

Wenzhi Gao,
Stanford University

Dongdong Ge
Shanghai Jiaotong University

Yinyu Ye
Stanford University

Table of contents

1 Introduction

HDSDP is a numerical software for general-purpose semidefinite programming problems (SDP).

$$
\begin{aligned}
\min_{X} \quad & \langle C, X \rangle \\
\text{subject to} \quad & \mathcal{A}X = b \\
& X \succeq 0
\end{aligned}
$$

The solver implements the dual interior point method [1] and incorporates several important extensions that enhance the performance of the dual method. HDSDP has been tested over massive datasets and proves a robust and efficient SDP solver [2].

1.1 Release note

- Sep 2022

  Initial release of HDSDP 1.0

- June 2023

  The whole repo rewritten based on new design structure

- March 2024

  Added a dedicated LP solver

- July 2024

  Added a primal SDP solver and hybrid primal-primal-dual LP solver

- Oct 2024

  Added new linear system support

2 Installation and requirement

HDSDP is written in ANSI C and is freely available. Since version 2.0, HDSDP is maintained as personal repository

https://github.com/Gwzwpxz/hdsdp

and all the source files for HDSDP are directly available.

2.1 Package dependence

HDSDP is a standalone subroutine library, and the only external dependence is LAPACK and BLAS, which are easily accessible for most platforms. Alternatively, HDSDP can be linked with Intel MKL and leverage its multi-thread computing features.

Linking with LAPACK and BLAS. Different distributions of BLAS and LAPACK sometimes have different ways of naming. For example, `ddot` can appear as `ddot`, `ddot_`, `DDOT`, or `DDOT_`. Users can enable the following macros in `external/lapack_names` to ensure that the symbols are correctly detected:

- `#define UNDERBLAS`

  For `ddot_`

- `#define CAPBLAS`

  For `DDOT`

- `#define UNDERCAPBLAS`

  For `DDOT_`

2.2 Building the HDSDP library and SDPA/MPS solver

HDSDP can be used in two ways: as a subroutine library or as an SDPA solver. The subroutine library allows users to call HDSDP within their customized applications, while the SDPA solver mainly serves for benchmarking purpose. Both the library and the SDPA solver can be built using the CMAKE build system.

The CMakeLists.txt file located in the repository should be modified based on the user's local environment.

- Line 2. Target architecture

  If a MacOS user is using HDSDP linked with MKL, uncomment the following line

  `# set(CMAKE_OSX_ARCHITECTURES x86_64)`

- Line 30. MKL path

  If a user wants to use HDSDP linked with MKL, set the path to the MKL libraries according to the instructions in CMakeLists.txt

By default HDSDP is compiled without MKL, and the only requirements are LAPACK and BLAS routines. Then the library can be built with

```
mkdir build
cd build
cmake ..  # Add "-DLINSYS_PARDISO=ON" if MKL is linked
make
```

and the build system will generate three files

- Static library `hdsdp`
- Dynamic library `hdsdp`
- Executable binary `sdpasolve`

2.3 Using the executable binary

The `sdpasolve` executable is used for benchmarking purpose and accepts two types of input

```
./sdpasolve data.dat-s # For SDP problems
./sdpasolve data.mps   # For standard-form LP problems
```

3 Data and solver interface

HDSDP is designed as a stand-alone optimization solver and provides a self-contained interface. After the data is input, the solver will initiate the subsequent solution phases automatically until the solution procedure ends or fails. With the compiled library, the user can call the solver by including the header `interface/hdsdp.h` and the data interface `interface/hdsdp_user_data.h`. See `test/test_file_io.c` for more examples of calling `HDSDP`.

The header `hdsdp.h` defines several utility macros, including return code, status code, parameters, constants and the solver interface.

The header `hdsdp_user_data.h` defines the user input data interface of `HDSDP`.

3.1 Notations

`HDSDP` internally solves the following mixed SDP-LP problem

$$\begin{array}{rll} \min\limits_{X,x} & \displaystyle\sum_{j=1}^{p} \langle C_j, X_j\rangle + \langle c, x\rangle & \\ \text{subject to} & \displaystyle\sum_{j=1}^{p} \langle A_{ij}, X_j\rangle + \langle a_i, x\rangle = b_i, & i=1,\ldots,m \\ & X_j \in \mathbb{S}_+^{n_j}, & j=1,\ldots,p \\ & x \in \mathbb{R}_+^{n_l}. & \end{array}$$

We define the following notations

- m is the number of constraints
- p is the number of SDP cones
- n_j is the cone dimension of the j-th SDP cone
- $n_s = \sum_{j=1}^{p} n_j$ is the total SDP cone dimension
- n_l is the LP cone dimension.

3.2 Return code

HDSDP subroutines use return code to handle error and exceptions. Subroutines with return code `hdsdp_retcode` returns one of the codes below

- `HDSDP_RETCODE_OK`

  The subroutine successfully exits

- `HDSDP_RETCODE_FAILED`

  The subroutine exits due to exception or error

- `HDSDP_RETCODE_MEMORY`

  The subroutine exits due to memory allocation issues

Users can use the `HDSDP_CALL(func)` macro from `interface/hdsdp_utils.h` when calling HDSDP routines, and the macro will automatically jump to `exit_cleanup` label when a non-OK status code is detected: for example

```
HDSDP_CALL(HDSDPOptimize(hdsdp, 1));
exit_cleanup:
    HDSDPDestroy(&hdsdp);
```

3.3 Data format

The data interface of HDSDP is defined in `interface/hdsdp_user_data.h` and allows users to input $\{C_j\}, \{A_{ij}\}, \{a_i\}, c$ using sparse matrix input.

3.3.1 SDP cone data

Each SDP cone in HDSDP is determined by m constraint matrices and an objective matrix

$$\{A_1, \ldots, A_m, C\},$$

where $\{A_i\}$ and C are all symmetric matrices. Given a symmetric matrix $A \in \mathbb{S}^n$:

$$A = \begin{pmatrix} \downarrow a_{11} & & & \\ \downarrow a_{21} & \downarrow a_{22} & & \\ \vdots & \vdots & \ddots & \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix},$$

we define its vectorization $\operatorname{vec}(A) : \mathbb{S}^n \to \mathbb{R}^{\frac{n \times (n+1)}{2}}$ as follows

$$\operatorname{vec}(A) := (a_{11}, a_{21}, \cdots a_{n1}, \ldots, a_{22}, a_{23} \ldots, a_{nn})^\top.$$

The vectorized version of A only contains the lower triangular part of A. Then, for each SDP block we can concatenate the coefficient matrices by

$$V_{\text{SDP}} := \begin{pmatrix} | & | & | & & | \\ \operatorname{vec}(C) & \operatorname{vec}(A_1) & \operatorname{vec}(A_2) & \cdots & \operatorname{vec}(A_m) \\ | & | & | & & | \end{pmatrix} \in \mathbb{R}^{\frac{n \times (n+1)}{2} \times (m+1)}.$$

Then the matrix V_{SDP} is compressed using standard compressed sparse column (`CSC`) format.

Example 1. Consider the following 2D SDP

$$\begin{aligned} \min_X \quad & \left\langle \begin{pmatrix} 6 & 2 \\ 2 & 3 \end{pmatrix}, X \right\rangle \\ \text{subject to} \quad & \left\langle \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, X \right\rangle = 1 \\ & \left\langle \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, X \right\rangle = 2 \\ & X \succeq 0. \end{aligned}$$

The problem has 2 constraints and an SDP cone. Then the SDP cone can be represented by

$$V_{\text{SDP}} = \begin{pmatrix} 6 & 1 & 0 \\ 2 & 0 & 2 \\ 3 & 1 & 0 \end{pmatrix}$$

and the cone can be represented by the following sparse matrix format.

```
Vp[4] = {0, 3, 5, 6};
Vi[6] = {0, 1, 2, 0, 2, 1};
Vx[6] = {6, 2, 3, 1, 1, 2};
```

3.3.2 LP cone data

LP cone data is similar to SDP, where we concatenate $\{a_i\}$ by

$$V_{\mathrm{LP}} = (\, c \quad a_1 \quad a_2 \quad \cdots \quad a_m \,) \in \mathbb{R}^{n \times (m+1)}$$

and V_{LP} is also compressed by `CSC` format.

Scalar bound cone. HDSDP also provides a scalar bound cone for constraints on the dual variables $l \cdot e \leq y \leq u \cdot e$. HDSDP will detect in the presolve phase whether an LP cone is a scalar bound cone, and will switch to bound cone when it is detected.

3.4 Data interface

3.4.1 Create user data structure

Function

```
extern hdsdp_retcode HUserDataCreate( user_data **pHdata );
```

Explanation

Create an HDSDP user data pointer

Argument

- `pHdata`

  Type: `user_data **`

  Address of the pointer to which the user data structure is to be allocated

Return

Status of execution

3.4.2 Set cone data

Function

```
extern void HUserDataSetConeData( user_data *Hdata,
                                  cone_type cone,
                                  int nRow,
                                  int nCol,
                                  int *coneMatBeg,
                                  int *coneMatIdx,
                                  double *coneMatElem );
```

Explanation

Set cone data in an existing user data structure

Argument

- `Hdata`

Type: `user_data *`

Pointer of the user data structure

- `cone`

  Type: `cone_type *`

  Type of the cone specified by the user. Possible choices are

  - `HDSDP_CONETYPE_LP`

    LP cone

  - `HDSDP_CONETYPE_DENSE_SDP`

    SDP cone

  - `HDSDP_CONETYPE_SPARSE_SDP`

    SDP cone where most of the $\{A_i\}$ are all-zero matrices

- `nRow`

  Type: `int`

  Number of constraints of the cone. Should be the same for all the cones

- `nCol`

  Type: `int`

  Dimension of the cone, n_l or n_j

- `coneMatBeg`

  Type: `int *`

  Column pointer for the `CSC` sparse matrix

- `coneMatIdx`

  Type: `int *`

  Row indices for the `CSC` sparse matrix

- `coneMatElem`

  Type: `double *`

  Elements for the `CSC` sparse matrix

Return

No return

3.4.3 Determine cone type

Function

```
extern cone_type HUserDataChooseCone( user_data *Hdata );
```

Explanation

Automatically detect and adjust the cone type based on problem data. The routine can determine **1)** whether a dense SDP cone should instead be specified as a sparse SDP cone **2)** whether an LP cone is a bound cone. This routine will be automatically invoked within the HDSDP solver routine

Argument

- Hdata

  Type: user_data *

  Pointer of the user data structure with cone data available

Return

Type of the cone detected by the solver

3.4.4 Clear user data

Function

```
extern void HUserDataClear( user_data *Hdata );
```

Explanation

Clear the user data

Argument

- Hdata

  Type: user_data *

  Pointer of the user data

Return

No return

3.4.5 Destroy user data structure

Function

```
extern void HUserDataDestroy( user_data **pHdata );
```

Explanation

Clear and release all the memory of user data

Argument

- pHdata

  Type: user_data **

  Address of the pointer to which the user data structure is allocated

Return

No return

3.5 Solver interface

3.5.1 Create the solver structure

Function

```
extern hdsdp_retcode HDSDPCreate( hdsdp **pHSolver );
```

Explanation

Create an HDSDP solver data structure

Argument

- `pHSolver`

  Type: `hdsdp **`

  Address of the pointer to which the solver is to be allocated

Return

Status of execution

3.5.2 Solver initialization

Function

```
extern hdsdp_retcode HDSDPInit( hdsdp *HSolver, int nRows, int nCones );
```

Explanation

Initialize solver and allocate solver memory

Argument

- `HSolver`

  Type: `hdsdp *`

  Pointer of the solver

- `nRows`

  Type: `int`

  Number of constraints of the problem

- `nCones`

  Type: `int nCones`

  Number of cones of the problem

Return

Status of execution

3.5.3 Set conic data

Function

```
extern hdsdp_retcode HDSDPSetCone( hdsdp *HSolver, int iCone, void *userCone );
```

Explanation

Set cone data in the solver

Argument

- `HSolver`

Type: `hdsdp *`

Pointer of the solver

- `iCone`

  Type: `int`

  Index of the cone. Range between 0 and `nCones` specified when calling `HDSDPInit`

- `userCone`

  Type: `void *`

  Pointer to user cone data

Return

Status of execution

3.5.4 Set dual objective

Function

```
extern void HDSDPSetDualObjective( hdsdp *HSolver, double *dObj );
```

Explanation

Set the primal constraint right-hand-side and the dual objective b

Argument

- `HSolver`

  Type: `hdsdp *`

  Pointer of the solver

- `dObj`

  Type: `double *`

  Dual objective. A vector of length m

Return

No return

3.5.5 Set dual starting point

Function

```
extern void HDSDPSetDualStart( hdsdp *HSolver, double *dStart );
```

Explanation

Set the starting point for the dual problem y_0.

Argument

- `HSolver`

  Type: `hdsdp *`

  Pointer of the solver

- `dStart`

  Type: `double *`

  Dual variable starting point. A vector of length m

Return

No return

3.5.6 Set integer parameter

Function

```
extern void HDSDPSetIntParam( hdsdp *HSolver,
                              int intParam,
                              int intParamVal );
```

Explanation

Set an integer parameter (see the next section for the available parameters)

Argument

- `HSolver`

  Type: `hdsdp *`

  Pointer of the solver

- `intParam`

  Type: `int`

  Integer parameter name

- `intParamVal`

  Type: `int`

  Value of the integer parameter

Return

No return

3.5.7 Set double parameter

Function

```
extern void HDSDPSetDblParam( hdsdp *HSolver,
                              int dblParam,
                              double dblParamVal );
```

Explanation

Set a double parameter (see the next section for the available parameters)

Argument

- `HSolver`

  Type: `hdsdp *`

  Pointer of the solver

- `dblParam`

  Type: `int`

  Double parameter name

- `dblParamVal`

  Type: `double`

  Value of the double parameter

Return

No return

3.5.8 Optimization

Function

```
extern hdsdp_retcode HDSDPOptimize( hdsdp *HSolver, int dOptOnly );
```

Explanation

Invoke HDSDP optimization routine

Argument

- `HSolver`

  Type: `hdsdp *`

  Pointer of the solver

- `dOptOnly`

  Type: `int`

  Whether HDSDP should only optimize the dual problem. If `dOptOnly` is nonzero, then the algorithm will perform Phase 1 (infeasible-start embedding phase) and will not go into primal solution extraction.

Return

Status of execution

3.5.9 Get dual solution

Function

```
extern hdsdp_retcode HDSDPGetRowDual( hdsdp *HSolver,
                                      double *pObjVal,
                                      double *dObjVal,
                                      double *dualVal );
```

Explanation

Extract dual optimal value after optimization finishes

Argument

- `HSolver`

  Type: `hdsdp *`

  Pointer of the solver

- `pObjVal`

  Type: `double *`

  Optimal primal objective *value*. Can be `NULL` if objective value is not needed

- `dObjVal`

  Type: `double *`

  Optimal dual objective *value*. Can be `NULL` if objective value is not needed

- `dualVal`

  Type: `double *`

  Optimal dual objective solution $y^\star$. Can be `NULL` if solution is not needed

Return

Status of execution

3.5.10 Get cone solution

Function

```
extern void HDSDPGetConeValues( hdsdp *HSolver,
                                int iCone,
                                double *conePrimal,
                                double *coneDual,
                                double *coneAuxi );
```

Explanation

Extract the cone primal optimal solution. Auxiliary memory is required for extraction.

Argument

- `HSolver`

  Type: `hdsdp *`

  Pointer of the solver

- `iCone`

  Type: `int`

  Index of the cone

- `conePrimal`

  Type: `double *`

  Optimal primal solution $X^\star$

- `coneDual`

  Type: `double *`

  Optimal dual slack $S^\star$

- `coneAuxi`

  Type: `double *`

  Auxiliary memory for extraction of size n_j for SDP) or n_l (for LP).

Return

No return

3.5.11 Check solution and compute solution metrics

Function

```
extern hdsdp_retcode HDSDPCheckSolution( hdsdp *HSolver, double diErrors[6] );
```

Explanation

Check the solution quality and compute the DIMACS errors

Argument

- `HSolver`

  Type: `hdsdp *`

  Pointer of the solver

- `diErrors`

  Type: `double *`

  The six DIMACS errors indicating the quality of the solution

Return

Status of execution

3.5.12 Release solver memory

Function

```
extern void HDSDPClear( hdsdp *HSolver );
```

Explanation

Release all the internal memory allocated by HDSDP

Argument

- `HSolver`

  Type: `hdsdp *`

  Pointer of the solver

Return

No return

3.5.13 Destroy solver structure

Function

```
extern void HDSDPDestroy( hdsdp **pHSolver );
```

Explanation

Clear and release all the memory of HDSDP.

Argument

- `pHSolver`

  Type: `hdsdp **`

  Pointer of the solver

Return

No return

4 Parameters

4.1 Integer parameters

4.1.1 INT_PARAM_CORRECTORA

Explanation: Number of corrector steps in HDSDP infeasible embedding phase. More corrector steps reduce the number of iterations but can increase time spent per iteration

Range: ≥ 0

Default: Adjusted by problem feature

4.1.2 INT_PARAM_CORRECTORB

Explanation: Number of corrector steps in HDSDP feasible dual potential reduction phase. More corrector steps reduce the number of iterations but can increase time spent per iteration

Range: ≥ 0

Default: Adjusted by problem feature

4.1.3 INT_PARAM_MAXITER

Explanation: Maximum number of potential reduction iterations

Range: ≥ 1

Default: 500

4.1.4 INT_PARAM_PSDP

Explanation: *Experiment feature*. Whether primal IPM solver is turned on when dual SDP is close to convergence

Range: $\{0, 1\}$

Default: 0

4.1.5 INT_PARAM_PRELEVEL

Explanation: Presolving level for problem data and KKT solver. In general no benefit using low level presolving

Range: $\{0, 1, 2\}$

Default: 2

4.1.6 INT_PARAM_THREADS

Explanation: Number of threads for linear algebra subroutines. Only effective when Intel MKL is linked to HDSDP.

Range: ≥ 1

Default: 12

4.2 Double parameters

4.2.1 DBL_PARAM_RELFEASTOL

Explanation: Relative feasibility tolerance for $\max\left\{\frac{\|\mathcal{A}X-b\|}{1+\|b\|_1}, \frac{\|\mathcal{A}^*y+S-C\|_F}{1+\|C\|_{\text{sum}}}\right\}$

Range: ≥ 0

Default: 10^{-8}, also adjusted by problem feature

4.2.2 DBL_PARAM_RELOPTTOL

Explanation: Relative optimality tolerance for $\frac{|\langle C,X\rangle-\langle b,y\rangle|}{|\langle C,X\rangle|+|\langle b,y\rangle|+1}$

Range: ≥ 0

Default: 10^{-8}, also adjusted by problem feature

4.2.3 DBL_PARAM_ABSFEASTOL

Explanation: Absolute feasibility tolerance for $\max\{\|\mathcal{A}X-b\|, \|\mathcal{A}^*y+S-C\|_F\}$

Range: ≥ 0

Default: 10^{-8}, also adjusted by problem feature

4.2.4 DBL_PARAM_ABSOPTTOL

Explanation: Absolute optimality tolerance for $|\langle C,X\rangle-\langle b,y\rangle|$

Range: ≥ 0

Default: 10^{-8}, also adjusted by problem feature

4.2.5 DBL_PARAM_TIMELIMIT

Explanation: Solver running time limit

Range: ≥ 0

Default: 3600.0

4.2.6 DBL_PARAM_POTRHOVAL

Explanation: Potential function parameter

Range: ≥ 0

Default: Adjusted by problem feature

4.2.7 DBL_PARAM_HSDGAMMA

Explanation: Infeasibility elimination aggressiveness in embedding phase

Range: $[0,1]$

Default: Adjusted by problem feature

4.2.8 DBL_PARAM_DUALBOX_LOW

Explanation: Scalar lower bound on the dual variables $y \geq l \cdot e$

Range: $\mathbb{R}$

Default: Adjusted by problem feature

4.2.9 DBL_PARAM_DUALBOX_UP

Explanation: Scalar upper bound on the dual variables $y \leq u \cdot e$

Range: $\mathbb{R}$

Default: Adjusted by problem feature

4.2.10 DBL_PARAM_BARMUSTART

Explanation: Starting barrier parameter

Range: ≥ 0

Default: 10^5

4.2.11 DBL_PARAM_DUALSTART

Explanation: Starting dual slack size: θ in $S_0 = C - \mathcal{A}^* y_0 + \theta I$

Range: ≥ 0

Default: 10^5

4.2.12 DBL_PARAM_POBJSTART

Explanation: Initial primal objective guess

Range: $\mathbb{R}$

Default: 10^{10}

4.2.13 DBL_PARAM_TRXESTIMATE

Explanation: Primal optimal solution trace estimate

Range: ≥ 0

Default: 10^8

4.2.14 DBL_PARAM_PRECORDACC

Explanation: Accuracy for preparing for primal solution recovery. When the dual solution is too accurate, extraction of primal solution can fail due to significant numerical errors. Therefore, this parameter should not be set too small.

Range: ≥ 0

Default: 10^{-8}

5 Logging and examples

HDSDP has a logging system that allows users to keep track of the optimization progress.

5.1 SDP logging

```
Filename: ../examples/mcp100.dat-s
Reading SDPA file in 0.000535 seconds

HDSDP: software for semi-definite programming

Wenzhi Gao, Dongdong Ge, Yinyu Ye, 2024
---------------------------------------------
Pre-solver starts
  Processing the cones
    M1: 0    M2: 100  M3: 0    M4: 0    M5: 0
  Starting KKT analysis
    Using dense Schur complement
  Collecting statistics
  Making adjustments
    Scale cone objective by 1.0e+00
    Scale rhs by 1.0e+00
    Hardware has 1 thread(s)
Pre-solver ends. Elapsed time: 0.0 seconds

Statistics
  Number of rows: 100
  Number of cones: 1
  Number of sparse SDP cones: 0
...
  Norm of objective: 2.69e+02
  Norm of SDP data: 1.00e+02
  Norm of RHS: 1.00e+02

Parameters
  Maximum iteration   : 500
  Infeasible corrector: 4
  Feasible corrector  : 0
  ...
  Dual box low        : -1.0e+06
  Dual box up         : 1.0e+06
  Starting primal     : 1.0e+08

This is a trace-implied SDP problem
Optimizing over 1 thread(s)
Initialize with dual residual 1.0e+05
HDSDP starts. Using infeasible dual method

    nIter             pObj             dObj      dInf         Mu   Step     |P|   T [D]
        1  +7.33331918e+07  -7.35913354e+07  0.00e+00  3.56e+04   0.76  7.2e+00   0.0

Infeasible method finds a dual feasible solution
HDSDP re-starts. Using feasible dual method

    nIter             pObj             dObj      pInf         Mu   Step     |P|   T [P]
        1  +7.33331918e+07  -4.81595948e+07  5.23e-02  4.26e+05   1.00  1.0e+01   0.0
...
       34  -2.26157351e+02  -2.26157351e+02  0.00e+00  5.28e-12   0.12  1.2e+01   0.1

Optimization time: 0.1 seconds
DIMACS error metric:
    1.10e-13 0.00e+00 0.00e+00 0.00e+00 6.50e-09 6.50e-09

SDP Status: Primal dual optimal
  pObj -2.2615734854e+02
  dObj -2.2615735148e+02
PD Gap +2.9464339377e-06
  Time 0.1 seconds
```

5.2 LP logging

```
Reading specialized standard form mps ../examples/afiro.mps
Optimizing an LP of 23 variables and 15 constraints
Data statistics: |A| = 1.06e+02 |b| = 9.82e+03 |c| = 2.08e+00 Nnz = 49
Using Hybrid Primal-Primal-Dual solver

    nIter            pObj             dObj      pInf      dInf        Mu    P/D Step   T [PD]
        0  -2.76027927e+02  -1.04931952e+02  8.59e-01  1.91e+00   0.00  0.00   0.0
        1  -2.54976427e+02  -4.55135868e+02  1.83e-02  3.93e-01   0.98  0.79   0.0
        2  -4.33824047e+02  -5.31380377e+02  4.30e-03  4.39e-03   0.76  0.99   0.0
        3  -4.63764901e+02  -4.83113579e+02  5.05e-04  9.48e-04   0.88  0.78   0.0
        4  -4.64643942e+02  -4.65030078e+02  3.98e-15  1.41e-05   1.00  0.99   0.0
        5  -4.64752596e+02  -4.64754528e+02  4.93e-16  7.06e-08   0.99  0.99   0.0
        6  -4.64753140e+02  -4.64753150e+02  2.70e-15  3.53e-10   1.00  1.00   0.0
        7  -4.64753143e+02  -4.64753143e+02  3.22e-15  1.77e-12   1.00  1.00   0.0

LP Status: Primal dual optimal

LP Solution statistic
pObj: -4.648e+02   dObj: -4.648e+02
Abs. pInf:  1.152e-11    Rel. pInf:  3.221e-15
Abs. dInf:  4.266e-12    Rel. dInf:  1.765e-12
Abs. Gap :  4.830e-08    Rel. gap :  5.191e-11

Linear solver statistic
Num. Factor:   7 (+1)    Factor Time:  0.00
Num. Solves:  15 (+1)    Solve  Time:  0.00

Elapsed Time:  0.000 seconds
```

References

[1] Steven J Benson and Yinyu Ye. Algorithm 875: dsdp5—software for semidefinite programming. *ACM Transactions on Mathematical Software (TOMS)*, 34(3):1–20, 2008.

[2] Wenzhi Gao, Dongdong Ge, and Yinyu Ye. Hdsdp: software for semidefinite programming. *ArXiv preprint arXiv:2207.13862*, 2022.