On the design of a dual potential reduction solver

HDSDP Developers' Group

November 26, 2022

In this note we describe the implementation of a dual potential reduction solver that exploits either the embedding or the big-M potential reduction method to solve

$$\begin{aligned} \min_{x} \quad & \langle c, x \rangle \\ \text{subject to} \quad & \mathcal{A}x = b \\ & x \succeq_{\mathcal{K}} 0 \end{aligned}$$

via its dual

$$\begin{aligned} \max_{y,s} \quad & b^\top y \\ \text{subject to} \quad & \mathcal{A}^* y + s = c \\ & s \succeq_{\mathcal{K}^*} 0. \end{aligned}$$

The main solver contains the following components

- Solver/data

  Interface: set data, set parameter, set solution, optimize, get solution

  Sparse, dense, rank-one

- Algorithm

  HSD/infeasible start/dual potential reduction

  Presolve, phase A, phase B

  Schur complement setup*, potential line-search, barrier line-search

- Linear algebra

  sparse, dense, low rank; eigen, trace, decomposition (Cholesky)

  Lanczos, conjugate gradient, block buffer computation*

- Other utilities

  Parameter tuner, IO and things like that

1 Algorithm

1.1 Notations

$\langle \cdot, \cdot \rangle$ denotes inner product; $\succeq_{\mathcal{K}}$ denotes partial order defined over cone $\mathcal{K}$; the operator $\mathcal{A}$: $\bigotimes_{i=1}^{k} \mathbb{R}^{n_i} \to \mathbb{R}^m$ and its adjoint $\mathcal{A}^*$: $\mathbb{R}^m \to \bigotimes_{i=1}^{k} \mathbb{R}^{n_i}$ are respectively defined. They have different expressions for different cones. The dual barrier is defined by $\log\det s$. e is generalized to express the unit vector in the conic space.

1.2 Basic dual algorithm from a KKT view

In the dual method, we are interested in applying the dual algorithm to solve the conic problem, whose KKT conditions give

$$\begin{aligned} \mathcal{A}x &= b \\ \mathcal{A}^* y + s &= c \\ xs &= 0(\mu e) \\ x, s &\succeq 0, \end{aligned}$$

where $xs = \mu e$ denotes the perturbed KKT system often used in the analysis of the interior point method.

From the Newton's perspective towards the condition, given (x, y, s) such that (y, s) satisfies $\mathcal{A}^* y + s = c$, we wish to find $\Delta x, \Delta s$ such that

$$\begin{aligned} \mathcal{A}(x + \Delta x) &= b \\ \mathcal{A}^* \Delta y + \Delta s &= 0 \\ (x + \Delta x)(s + \Delta s) &= \mu e. \end{aligned}$$

Since we assume a dual feasible solution, the second term has no related residual. Dual method modifies the third condition and

$$x + \Delta x = \mu (s + \Delta s)^{-1}.$$

Linearizing $(s + \Delta s)^{-1} \approx s^{-1} - s^{-1} \Delta s s^{-1}$ gives

$$\mu s^{-1} \Delta s s^{-1} + \Delta x = \mu s^{-1} - x.$$

Combined with the above relations, we have

$$\begin{aligned} \mathcal{A}\Delta x &= \mu \mathcal{A} s^{-1} - \mathcal{A}x - \mu \mathcal{A} s^{-1} \Delta s s^{-1} \\ &= \mu \mathcal{A} s^{-1} - \mathcal{A}x + \mu \mathcal{A} s^{-1} (\mathcal{A}^* \Delta y) s^{-1} \\ &= \mu \mathcal{A} s^{-1} - \mathcal{A}x + \mu \mathcal{A} s^{-2} \mathcal{A}^* \Delta y \\ &= b - \mathcal{A}x \end{aligned}$$

and we arrive at

$$\mathcal{A} s^{-2} \mathcal{A}^* \Delta y = \frac{1}{\mu} b - \mathcal{A} s^{-1}.$$

Solving the system provides $(\Delta y, \Delta s = -\mathcal{A}^* \Delta y)$ we need.

1.3 Homogeneous dual method

Homogeneous dual method works almost the same as dual method but applies simplified embedding trick

$$\begin{aligned} \mathcal{A}x - b\tau &= 0 \\ -\mathcal{A}^* y - s + c\tau &= 0 \\ \langle b, y \rangle - \langle c, x \rangle - \kappa &= 0 \\ xs &= \mu e \\ \kappa \tau &= \mu \end{aligned}$$

By treating τ as dual variable, we have, similarly

$$\begin{aligned} \mathcal{A}(x + \Delta x) - b(\tau + \Delta \tau) &= 0 \\ -\mathcal{A}^*(y + \Delta y) - (s + \Delta s) + c(\tau + \Delta \tau) &= 0 \\ \langle b, y + \Delta y \rangle - \langle c, x + \Delta x \rangle - (\kappa + \Delta \kappa) &= 0 \\ (x + \Delta x)(s + \Delta s) &= \mu e \\ (\kappa + \Delta \kappa)(\tau + \Delta \tau) &= \mu. \end{aligned}$$

Since we do not assume a dual feasible solution this time, $r^d := -\mathcal{A}^* y - s + c\tau$ may be nonzero and we have

$$\begin{aligned} \mathcal{A}\Delta x - b\Delta \tau &= b\tau - \mathcal{A}x \\ -\mathcal{A}^* \Delta y - \Delta s + c\Delta \tau &= \mathcal{A}^* y + s - c\tau \\ \langle b, \Delta y \rangle - \langle c, \Delta x \rangle - \Delta \kappa &= -\langle b, y \rangle + \langle c, x \rangle + \kappa \\ \mu s^{-1} \Delta s s^{-1} + \Delta x &= \mu s^{-1} - x \\ \mu \tau^{-2} \Delta \tau + \Delta \kappa &= \mu \tau^{-1} - \kappa. \end{aligned}$$

After some tedius simplification we arrive at

$$\begin{aligned} \mathcal{A}s^{-2}\mathcal{A}^*\Delta y - (b+\mu\mathcal{A}s^{-1}cs^{-1})\Delta\tau &= b\tau - \mu\mathcal{A}s^{-1} + \mu\mathcal{A}s^{-1}r^d s^{-1} \\ (b-\mu\mathcal{A}s^{-1}cs^{-1})\Delta y + (\mu\langle c, s^{-1}cs^{-1}\rangle + \kappa\tau^{-1})\Delta\tau &= -\langle b, y\rangle + \mu\tau^{-1} + \mu\langle c, s^{-1}\rangle - \mu\langle c, s^{-1}r^d s^{-1}\rangle \end{aligned}$$

and we can update $(\Delta y, \Delta s, \Delta\tau)$ in the same fashion.

1.4 Conic operations and mixed cones

It is clear that we need $\langle c, x\rangle, \mathcal{A}s^{-1}cs^{-1}, \langle c, s^{-1}cs^{-1}\rangle$ and nutorious $\mathcal{A}s^{-2}\mathcal{A}^*$ in our computation. In different conic contexts we have

Operation/Cone	LP $(x \succeq 0)$	SDP $(X \succeq 0)$	...
e	$(1, \ldots, 1)^\top$	I	
s^{-1}	$(1/s_1, \ldots, 1/s_n)$	S^{-1}	
$s^{-1}c$	$(c_1/s_1, \ldots, c_n/s_n)$	$S^{-1}C$	
$\langle c, x\rangle$	$c^\top x$	$\langle C, X\rangle$	
$\mathcal{A}x$	Ax	$(\langle A_1, X\rangle, \ldots, \langle A_m, X\rangle)$	
$\mathcal{A}^*y$	$A^\top y$	$\sum_i y_i A_i$	
$\mathcal{A}s^{-1}cs^{-1}$	$As^{-2}c$	$(\langle A_1, S^{-1}CS^{-1}, \ldots, A_m, S^{-1}CS^{-1}\rangle)$	
$\langle c, s^{-1}cs^{-1}\rangle$	$c^\top s^{-2}c$	$\langle C, S^{-1}CS^{-1}\rangle$	
$M = \mathcal{A}s^{-2}\mathcal{A}^*$	$As^{-2}A^\top$	$M_{ij} = \langle A_i, S^{-1}A_j S^{-1}\rangle$	

Table 1. Conic operations

When more than one cone is present, by linearity of $\mathcal{A}$, we simply add the quantities, e.g.,

$$\begin{aligned} M &= M_{\text{LP}} + M_{\text{SDP}} \\ \mathcal{A}s^{-1}cs^{-1} &= As^{-2}c + (\langle A_1, S^{-1}CS^{-1}, \ldots, A_m, S^{-1}CS^{-1}\rangle)^\top \\ \langle c, s^{-1}cs^{-1}\rangle &= c^\top s^{-2}c + \langle C, S^{-1}CS^{-1}\rangle. \end{aligned}$$

When other cones are added, we only need to define their conic operations and add them together.

2 SDP Data structures

2.1 Factorized data

The following structure stores the eigen-decomposition of a data matrix $A = \sum_{i=1}^r \lambda_i u_i u_i^\top$. The structure should support the following operations.

- $\langle A, B\rangle = \sum_{i=1}^r \lambda_i u_i^\top B u_i$
- $B = S^{-1}AS^{-1} = \sum_{i=1}^r \lambda_i (S^{-1}u_i)(S^{-1}u_i)^\top$

  In this case the LHS serves as a buffer

```
typedef struct {

    int     rank;
    double *evals;
    double *evecs;

} eigFactor;
```

2.2 SDP coefficient matrix

NOTE: Only lower triangular is stored.

We use the following structures to store A and C matrices from SDP coefficients. They should support the following functionalities

- $B \leftarrow \alpha A + B$
- $\langle A_i, A_j \rangle$ (TODO)
- $\|A\|_F$
- $\sum_{ij} |a_{ij}|$
- $A \leftarrow \alpha A$
- `[V, e] = eig(A)` (TODO)
- `full(A)`

```
typedef struct {

    int       nCol;
    void      *dataMat;
    eigFactor *eig;
    void      (*dataMataApB)  (void *, double, void *);
    void      (*dataMatScal)  (void *, double);
    double    (*dataMatNorm)  (void *, int);
    int       (*dataMatEig)   (void *, void **);
    int       (*dataMatGetNnz)(void *);
    void      (*dataMatDump)  (void *, double *);

} sdpCoeffMat;
```

2.2.1 Sparse matrix

```
typedef struct {

  int     nSDPCol;
  int     nTriMatElem;
  int    *triMatCol;
  int    *triMatRow;
  double *triMatElem;

} sdpSparseData;
```

2.2.2 Dense matrix

```
typedef struct {

  int     nSDPCol;
  double *dsMatElem;

} sdpDenseData;
```

2.2.3 Rank-one sparse matrix

```
typedef struct {

  int      nSDPCol;
  int      nSpR1FactorElem;
  int     *spR1MatIdx;
  double  *spR1MatElem;

} sdpRankOneSpData;
```

2.2.4 Rank-one dense matrix

```
typedef struct {

  int     nSDPCol;
  double *r1MatFactor;

} sdpRankOneSpData;
```

2.3 SDP variable and step

We use the following structures to store S.

- S^{-1}
- `L = chol(S)`
- `L \ z, L' \ z`
- $y \leftarrow \alpha S x + y$

coming...

2.4 Schur complement matrix

3 Contribution and formats

- Indentation, bracket

  Default as in Xcode, following the samples below
- Doxygen string and comments

  Using `@file, @brief, /* */`
- Function with `void` return value should `return;`
- Name style

  Bottom-level routine: `extern void csp_Axpby`

  Medium-level routine: `hdsdpSpMatTrace`
- Use `assert` whenever necessary
- Static before extern
- ...

```c
static int pdsCreate( void **pldl, int n ) {

    int retcode = RETCODE_OK;
    pds_linsys *pds = NULL;

    POTLP_INIT(pds, pds_linsys, 1);

    if ( !pds ) {
        retcode = RETCODE_FAILED;
        goto exit_cleanup;
    }

     pds->n = n;
    *pldl = pds;

    /* Initialize pardiso */
    POTLP_ZERO(pds->pt, void *, 64);
    POTLP_ZERO(pds->iparm, int, 64);

    int mtype = PARDISO_SYM_INDEFINITE;
    pardisoinit(pds->pt, &mtype, pds->iparm);

    set_pardiso_param(pds->iparm, PARDISO_PARAM_NONDEFAULT, 1);
    set_pardiso_param(pds->iparm, PARDISO_PARAM_SYMBOLIC, PARDISO_PARAM_SYMBOLIC_MMD);
    set_pardiso_param(pds->iparm, PARDISO_PARAM_PERTURBATION, 3);
    set_pardiso_param(pds->iparm, PARDISO_PARAM_INPLACE, 1);
    set_pardiso_param(pds->iparm, PARDISO_PARAM_INDEX, PARDISO_PARAM_INDEX_C);

exit_cleanup:
    return retcode;
}
```

```c
extern void potVecScal( pot_vec *pVexX, double sVal ) {

    scal(&pVexX->n, &sVal, pVexX->x, &potIntConstantOne);

    if ( pVexX ->nrm != -1.0 ) {
        pVexX->nrm = pVexX->nrm * fabs(sVal);
    }

    return;
}
```