On the design of a dual potential reduction solver

HDSDP Developers' Group

November 24, 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

$$\begin{array}{rl} \min_x & \langle c, x\rangle \\ \text{subject to} & \mathcal{A}x = b \\ & x \succeq_{\mathcal{K}} 0 \end{array}$$

via its dual

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

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 SDP Data structures

1.1 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.

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

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

  In this case the LHS serves as a buffer

```
typedef struct {

    int     nCol;
    int     rank;
    double *evals;
    double *evecs;

} eigFactor;
```

1.2 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 *);
    double        (*dataMatDot)   ( void *, double * );
    void          (*dataMatScal)  (void *, double);
    double        (*dataMatNorm)  (void *, int);
    hdsdp_retcode (*dataMatEig)   (void *, void **);
    int           (*dataMatGetNnz)(void *);
    void          (*dataMatDump)  (void *, double *);

} sdpCoeffMat;
```

1.2.1 Sparse matrix

```
typedef struct {

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

} sdpSparseData;
```

1.2.2 Dense matrix

```
typedef struct {

  int     nSDPCol;
  double *dsMatElem;

} sdpDenseData;
```

1.2.3 Rank-one sparse matrix

```
typedef struct {

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

} sdpRankOneSpData;
```

1.2.4 Rank-one dense matrix

```
typedef struct {

  int     nSDPCol;
  double *r1MatFactor;

} sdpRankOneSpData;
```

1.3 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…

1.4 Schur complement matrix

2 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
- ...

```
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;
}
```

```
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;
}
```