SUITEOPT 2.0.0, FEBRUARY 15, 2022

WILLIAM W. HAGER
UNIVERSITY OF FLORIDA

1. Introduction.

`SuiteOPT` is a software package that currently solves problems of the form

$$\min\ f(\mathbf{x}) \quad \text{subject to} \quad \mathbf{bl} \le \mathbf{Ax} \le \mathbf{bu}, \quad \mathbf{lo} \le \mathbf{x} \le \mathbf{hi}. \tag{P}$$

Here $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{bl}$ and $\mathbf{bu} \in \mathbb{R}^m$, and $\mathbf{lo}$ and $\mathbf{hi} \in \mathbb{R}^n$. There are four different packages contained in SuiteOPT.

1. `PPROJ:` Given $\mathbf{y} \in \mathbb{R}^n$, PPROJ uses the algorithm of [20] along with the Dual Active Set Algorithm [5, 6, 10, 11, 12, 13, 14, 15], and techniques for updating and downdating a sparse Cholesky factorization [1, 2, 3, 4, 7], to solve the projection problem
$$\min\ \|\mathbf{y} - \mathbf{x}\|^2 \quad \text{subject to} \quad \mathbf{bl} \le \mathbf{Ax} \le \mathbf{bu}, \quad \mathbf{lo} \le \mathbf{x} \le \mathbf{hi}.$$
2. `NAPHEAP:` Given $\mathbf{a} \in \mathbb{R}^n$ and a diagonal matrix $\mathbf{D}$ with nonnegative diagonal, `NAPHEAP` uses the Newton/heap-based algorithm of [8] to solve the problem
$$\min\ \left(\frac{1}{2}\right)\mathbf{x}^{\mathsf{T}}\mathbf{Dx} - \mathbf{c}^{\mathsf{T}}\mathbf{x} \quad \text{subject to} \quad \text{bl} \le \mathbf{a}^{\mathsf{T}}\mathbf{x} \le \text{bu}, \quad \mathbf{lo} \le \mathbf{x} \le \mathbf{hi}.$$
3. `CG_DESCENT:` The algorithms of [16, 17, 18] are used to solve an unconstrained optimization problem by the conjugate gradient method.
4. `PASA:` The polyhedral active set algorithm uses the framework of [19] to combine the previous algorithms and solve a general problem of the form (P).

Since the update and downdate techniques used by PPROJ are contained in the CHOLMOD package of Timothy A. Davis' SuiteSparse, the relevant parts of SuiteSparse were extracted to form the directory SuiteSparseX, which is included in the SuiteOPT software. Although there are four different solvers, they can all be accessed through PASA; that is, the PASA software will analyze the problem structure and determine which of the solvers, or combination of solvers, should be used to solve the given problem.

There are three different interfaces to the software:

(a) Since the codes are written in C, they can be invoked inside a C code.
(b) Problems can be formulated and solved using MATLAB.
(c) There is an interface through the CUTEst (Constrained and Unconstrained Testing Environment with Safe Threads) platform of Gould, Orban, and Toint [9].

We now explain how to install and use the software. Regardless of the software interface, an important parameter when setting up a problem is the number infinity. Floating point infinity is defined in `SuiteOPTconfig/SuiteOPTconfig.h` based on ANSI C99 and C90 standards. This definition is consistent with MATLAB's infinity, "`inf`". If the variable x_i has no upper bound, then in C, the user would set "`hi [i] = SuiteOPTinf`", while in MATLAB "`hi(i) = inf`". If all the components of $\mathbf{x}$ have no upper bound, then leave **hi** undefined. Lower bounds are treated in a similar fashion, but with `hi` replaced by `lo` and with infinity replaced by minus infinity. As explained below, the user can override the default definition of infinity in the `Userconfig.mk` file,

2. Installation and Use in MATLAB. Startup MATLAB in the directory `SuiteOPT/MATLAB`. In the command window, type "`make`" and follow the instructions. To use a solver in MATLAB, a structure must be constructed that contains the problem data and any changes to the default parameter values. If `Data` denotes an input data structure associated with a solver, then the command to solve a problem is simply

```
x = solver_name (Data) ;
```

where `solver_name` is either `pasa`, `pproj`, `cg_descent`, or `napheap`.

The important elements of the input data structure that the user may wish to modify are described in a `readme.m` file in the solver's MATLAB subdirectory. In MATLAB, type "`help readme`" to view the file. By default, when an element of the problem data structure is not present, the code assumes that the associated constraint does not exist. In other words, if the constraint $\texttt{x} \le \texttt{hi}$ does not exist and `data` denotes the input data structure for a solver, then do not specify a value for the data element `data.hi`. Often, there is no need to specify dimensional information such as `data.nrow` and `data.ncol`, the number of rows and columns for the matrix `A` in (P), since MATLAB usually learns the dimensions when the matrix is created.

For the definitions of the default parameters associated with a solver, see the `default.c` file in the `Source` directory for the solver. For any of the solvers except `pasa`, the value of a parameter is changed using a statement of the form

```
data.parameter = (new value) ;
```

where `data` is the input data structure for the solver and `parameter` is the name of the parameter whose value is to be changed. On the other hand, since `pasa` uses all the solvers, the corresponding statement is

```
pasadata.solver.parameter = (new value) ;
```

where `solver` is either `pasa`, `pproj`, `napheap`, or `cg`. See the demo files in the subdirectories

SuiteOPT/PASA/MATLAB,
SuiteOPT/PPROJ/MATLAB,
SuiteOPT/CGDESCENT/MATLAB, and
SuiteOPT/NAPHEAP/MATLAB

for examples showing how to set up a problem.

3. The User Configuration File, and Compilation and Use in C. If `SuiteOPT` is used in C or CUTEst, then a file

`SuiteOPTconfig/Userconfig.mk`

must be provided with the following information:

(1) The C compiler denoted CC.
(2) The optimization flags denoted OPTFLAGS.
(3) A specification of the BLAS and LAPACK routines, and possibly the path to the BLAS.
(4) Possibly a new definition for floating point infinity.

The file `Userconfig.mk` contains examples showing how to set up the User's configuration file. Since the BLAS are crucial for the efficiency of the algorithms, a word of caution if OpenBLAS are used. SuiteSparse Version 5.4 employs OpenMP threading, and the OpenBLAS routine DGEMM, which is used in supernodal Cholesky factorizations, can be very slow with OpenMP threading. Hence, if the OpenBLAS are used, it is recommended to not use the supernodal routines; this is achieved by adding to OPTFLAGS the option "-DNSUPER". Alternatively, the user can employ a different version of the BLAS that may work better with OpenMP threading, such as Intel's MKL (Math Kernel Library) BLAS. The Userconfig.mk file contains examples showing how to set up this file. Note that "`cmake`" is needed to compile SuiteSparse, and the BLAS may also require gfortran and pthread.

Once the `Userconfig.mk` file is set up, the `SuiteOPT` C codes can be compiled from the top level `SuiteOPT` directory by typing "`make`". A specific `solver` from `SuiteOPT` can be compiled from the top level directory by typing "`make solver`", where `solver` is either `pasa`, `cg`, `napheap`, or `pproj`. Alternatively, navigate to the subdirectory for the solver, and type "`make`".

The codes in the Demo subdirectories all have the layout shown below. Here, {`solver`} should be replaced by either `pasa`, `cg`, `napheap`, or `pproj`. The "`solver_name`" is either `pasa`, `pproj`, `napheap`, or `cg_descent`.

```
/* Create the data structure for the solver */
    Data = solver_setup () ;
/* where ``solver'' is pasa, pproj, napheap, or cg */

/* Store problem information in Data structure */
    Data.lo = ... ;

/* Solve the problem */
    solver_name (Data) ;
/* replace solver_name by pasa, pproj, napheap, or cg_descent */

/* Extract solution from Data structure */
    for (j = 0; j < ncol; j++) printf ("x [%i] = %e\n", j, Data.x [j]) ;

/* Terminate the problem */
    {solver}_terminate (&Data) ;
```

4. Installation and Use in CUTEst.

To install either `pasa` or `cg_descent` in CUTEst, the user must first download and install the CUTEst package. If an existing version of CUTEst is already installed, be sure to update to a version after August, 2019, since CUTEst was modified at that date to enable use with SuiteOPT. Currently, CUTEst can be downloaded from

https://github.com/ralna/CUTEst/wiki

Install the packages archdefs, cutest, and sifdecode, as well as the CUTE library of test problems. After installing CUTEst, be sure to set the environment variables that are highlighted during the installation process. The two variables needed by `SuiteOPT` are CUTEST, which is the full path to the location of your cutest directory, and MYARCH, which is a string providing information about your operating system and computer architecture. After setting up CUTEst, add two new lines to the file $CUTEST/bin/sys/$MYARCH to provide your BLAS and LAPACK information. This same information appears in the Userconfig.mk file except that quotation marks should be inserted around the libraries:

```
BLAS="... -lgfortran -lpthread"
LAPACK="..."
```

Another environment variable that needs to be set is LD_LIBRARY_PATH. This variable should include the full path to the `ldlibs` directory at the top level of `SuiteOPT`. The command for setting up the environment variable is something like the following (the full path name should be in quotes):

```
setenv LD_LIBRARY_PATH 'path'
```

If there are multiple paths in the load library, then each path is separated by a colon.

Once this setup is complete, the installation of `pasa` and `cg_descent` in CUTEst is completed by typing "`make cute`" in the top level directory of `SuiteOPT`. As the codes are compiled and installed into CUTEst, the following two files are created:

```
SuiteOPT/CGDESCENT/CUTEst/runcutest
SuiteOPT/PASA/CUTEst/runcutest.
```

These files contain aliases for running either `pasa` or `cg_descent` in CUTEst. If these aliases are placed in a file such as ".cshrc" or ".bashrc" that is executed at startup, then the aliases can be used in any window that is opened subsequently. The command for solving a polyhedral constrained optimization problem PROB.SIF using the PASA alias is "`pasarun PROB`". If the problem is unconstrained, then it could also be solved using the command "`cgrun PROB`"; in either case, when the problem is unconstrained, it is solved using `cg_descent`.

New values for the pasa parameters can be inserted in

```
SuiteOPT/PASA/CUTEst/pasa_main.c
```

right before execution of the call to `pasa`. New values for the CG_DESCENT parameters can be inserted right before the call to "`cg_descent`" in

```
SuiteOPT/CGDESCENT/CUTEst/cg_descent_main.c.
```

When new parameter values are set, the `pasa_main.c` or the `cg_descent_main.c` codes should be recompiled by typing "`make`" in their respective CUTEst subdirectories.

5. Parameters. The names of the parameters, their definitions, and their default values can be found in the source directory file whose name ends in `default.c`. When the `setup` code for any solver is executed, it not only creates the input data structure for the solver, but it also initializes the parameters to their default values. Thus after running the `setup` code and before executing the associated solver, any of the default parameter values can be changed. Examples showing how to set parameters in C appear in the `Demo` directory for any solver. As an illustration, suppose we wish to change the convergence tolerance for `cg_descent` from its default value `1.e-06` to `1.e-08`. If `cgdata` denotes the output of `cg_setup`, then the statement

```
cgdata->Parm->grad_tol = 1.e-08 ;
```

resets the convergence tolerance to `1.e-08`.

The solver `pasa` could potentially utilize any of the solvers `pproj`, `cg_descent`, or `napheap`. Some parameters, such as `grad_tol`, are propagated from `pasa` to the other solvers. Other parameters, however, are uniquely associated with a solver. For example, if `cg_descent` is used in the subspace optimization by `pasa`, then the user may wish to change the value of LBFGS memory from its default value 11, to say 15. If `pasadata` denotes the output of the `pasa_setup` code, then the statement

```
pasadata->Parms->cg->LBFGSmemory = 15 ;
```

resets the LBFGS memory to 15. The four elements of the `Parms` structure are `pasa`, `cg`, `pproj`, and `napheap`.

Resetting parameters in MATLAB is different and easier when compared to C. In MATLAB, one builds a structure with the non-default elements of the solver's data structure, and parameters are treated like any other element of the data structure. If `pasadata` denotes a structure being built in MATLAB to solve a problem using `pasa`, then the following statement resets the LBFGS memory to 15:

```
pasadata.cg.LBFGSmemory = 15 ;
```

Essentially, the `Parms` structure in the C example above is omitted in MATLAB. If `cg_descent` is used to solve an unconstrained optimization problem in MATLAB and cgdata denotes the data structure being built, then the statement

```
cgdata.LBFGSmemory = 15 ;
```

resets the LBFGS memory to 15.

6. Appendix: Elements of Input Data Structures.

This appendix provides a list of the elements in the solver input data structure that the user may wish to specify. All of the solvers in `SuiteOPT` use floating point variables of size `SuiteOPTfloat`, which is "`double`" by default. The integer variables are of size `SuiteOPTint`, which is "`int`" by default. For really big problem, "`int`" would need to be changed to "`long`". The size of `SuiteOPTfloat` and `SuiteOPTint` can be changed in the file

```
SuiteOPT/SuiteOPTconfig/SuiteOPTconfig.h
```

Throughout this appendix, `SuiteOPTfloat` and `SuiteOPTint` are denoted `SF` and `SI` respectively.

Structure: PASAdata

The PASAdata structure is input to `pasa` when solving an optimization problem of the form:

$$\min_{\mathbf{x}} \; f(\mathbf{x}) \quad \text{subject to} \quad \mathbf{bl} \le \mathbf{Ax} \le \mathbf{bu}, \quad \mathbf{lo} \le \mathbf{x} \le \mathbf{hi}.$$

Special Cases Treated:

1. Unconstrained optimization, the constraints are not present.
2. Bound constrained optimization, the constraint is $\mathbf{lo} \le \mathbf{x} \le \mathbf{hi}$.
3. Linear programming, the objective is linear $\mathbf{c}^{\mathsf{T}}\mathbf{x}$.
4. A quadratic program, the objective is $0.5\mathbf{x}^{\mathsf{T}}\mathbf{Qx} + \mathbf{c}^{\mathsf{T}}\mathbf{x}$.
5. A projection, the objective is $\|\mathbf{y} - \mathbf{x}\|^2$, and $\mathbf{y}$ is a given vector to project on the polyhedron.
6. A separable quadratic, where the objective is min $0.5\mathbf{x}^{\mathsf{T}}\mathbf{Dx} + \mathbf{c}^{\mathsf{T}}\mathbf{x}$ subject to $\mathbf{lo} \le \mathbf{x} \le \mathbf{hi}$ and $\text{bl} \le \mathbf{a}^{\mathsf{T}}\mathbf{x} \le \text{bu}$, where $\mathbf{D}$ is a diagonal matrix with nonnegative diagonal $\mathbf{d}$ and $\mathbf{a}$ is a column vector.

NOTE: When the setup code for any solver is run, all the fields of the data structure are set to either NULL or EMPTY. Elements of the data structure that are not needed to describe a problem do not need to be touched; the NULL or EMPTY default values indicate that the data element is not present in the problem.

Name	Type	Description
nrow	SI	Number of rows in $\mathbf{A}$ if it exists.
ncol	SI	Number of components in x (= number of cols in $\mathbf{A}$ if it exists).
lambda	SF *	Size nrow. If parameter use_lambda is TRUE, then lambda points to a starting guess for the multiplier. If lambda is NULL, then PASA allocates memory for lambda, and returns in lambda the multiplier associated with the constraint $\mathbf{bl} \le \mathbf{Ax} \le \mathbf{bu}$. Any allocated memory is freed by pasa_terminate.
x	SF *	Size ncol. Points to a starting guess for routines that require one (NAPHEAP, PPROJ, and the LP solver do not require a starting guess for x). If NULL, then PASA allocates memory for x, and if a starting guess is needed, then it is set to zero. The problem solution is returned in x. Any allocated memory is freed by pasa_terminate.
lo	SF *	Size ncol. Lower bound in the constraint $\mathbf{lo} \le \mathbf{x}$. NULL implies use $-\infty$ for lower bound.
hi	SF *	Size ncol. Upper bound in the constraint $\mathbf{x} \le \mathbf{hi}$. NULL implies use $+\infty$ for upper bound.
bl	SF *	Size nrow. Lower bound in the constraint $\mathbf{bl} \le \mathbf{Ax}$. NULL implies use $-\infty$ for lower bound.

bu	SF *	Size nrow. Upper bound in the constraint $\mathbf{Ax} \le \mathbf{bu}$. NULL implies use $+\infty$ for upper bound.
y	SF *	Size ncol. A vector to be projected onto polyhedron $\mathbf{bl} \le \mathbf{Ax} \le \mathbf{bu},\ \mathbf{lo} \le \mathbf{x} \le \mathbf{hi}$.
A_by_rows	SF *	Size nrow*ncol. Numerical entries in $\mathbf{A}$ (when constraint matrix input as a dense matrix).
A_by_cols	SF *	Size nrow*ncol. Numerical entries in $\mathbf{A}$ (when constraint matrix input as a dense matrix).
Ti	SI *	Size Tnz. Row indices of nonzero elements of $\mathbf{A}$ (when constraint matrix input in triples format).
Tj	SI *	Size Tnz. Column indices of nonzeros in $\mathbf{A}$.
Tx	SF *	Size Tnz. Numerical values of nonzeros in $\mathbf{A}$.
Tnz	SI	Number of entries in Ti, Tj, and Tx.
sym	int	Only used in triples format. TRUE implies matrix is symmetric and only element on main diagonal and on one side are given
Ap	SI *	Size ncol + 1. Column pointers for $\mathbf{A}$ (when constraint matrix input using sparse matrix format).
Ai	SI *	Size Ap [ncol]. Row indices in increasing order in each column for the nonzeros in $\mathbf{A}$.
Ax	SF *	Size Ap [ncol]. Numerical entries in $\mathbf{A}$ coresponding to the row indices in Ai.
a	SF *	Size ncol. The coefficient matrix in the case where nrow = 1 (dense vector).
d	SF *	Size ncol. Hessian diagonal when the objective Hessian is a diagonal matrix.
value	function	value (SF *f, SF *x, SI ncol) should put the value of the objective at x in *f (not needed for a QP)
grad	function	grad (SF *g, SF *x, SI ncol) should put the gradient of the objective at x in g (not needed when for a QP)
valgrad	function	valgrad (SF *f, SF *g, SF *x, SI ncol) put the value of the objective at x in *f and the gradient in g (this routine is optional, but it can speed up the computations when the objective and its gradient are computed faster together than than they are computed separately.
c	SF *	Size ncol. Linear term when objective is quadratic or linear.
Hdense	SF *	Size ncol*ncol. Numerical entries in Hessian of a QP when Hessian input as a dense matrix.

HTi	SI *	Size Hnz. Row indices of nonzero elements in Hessian of a QP when Hessian input in triples format. There is one row index entry for each nonzero in the matrix.
HTj	SI *	Size Hnz. Column indices of nonzeros in Hessian. There is one column index entry for each nonzero in the matrix.
HTx	SF *	Size Hnz. Numerical values of nonzeros in Hessian.
Hnz	SI	Number of entries in HTi, HTj, and HTx.
Hsym	int	Only used in triples format. TRUE implies only element on main diagonal and on one side are given. FALSE implies that all nonzeros in the Hessian are given.
Hp	SI *	Size ncol + 1. Column pointers for Hessian (when Hessian input using sparse matrix format).
Hi	SI *	Size Hp [ncol]. Row indices in increasing order in each column for the nonzeros in Hessian.
Hx	SF *	Size Hp [ncol]. Numerical entries in Hessian coresponding to the indices in Hi.
hprod	function	hprod (SF *p, SF *d, SI *F, SI ncol, SI n) computes p = H(:, F)*d where H is the ncol by ncol Hessian of a QP, d has n elements, and F denotes a collection of n indices contained in 1:ncol (not needed if the nonzero matrix elements of the Hessian are provided).
cg_prod	function	cg_hprod (SF *p, SF *x, SI ncol) computes p = H*x where H is the ncol by ncol Hessian of a QP (not needed if the nonzero matrix elements of the Hessian are provided).
Parms	PASAparms *	Contains pointers to four parameter structures corresponding to solver S = pasa, cg, pproj, or napheap. Parms->S is the parameter structure associated with solver S.
Stats	PASAstats *	Contains pointers to four statistics structures corresponding to solver S = pasa, cg, pproj, or napheap. Stats->use_S is TRUE if solver S was used during the run and the statistics for solver S are found in Stats->S.
xWork	SF *	NULL (default set in pasa_setup) implies that the code should allocate the floating point work area, otherwise the user provides a pointer xWork to the floating point work area.
iWork	SI *	NULL (default set in pasa_setup) implies that the code should allocate the integer work area, otherwise the user provides a pointer iWork to the integer work area.

The are three ways to input either the constraint matrix **A** or the Hessian of a QP:

1. Using a dense packed array containing the matrix entries. When inputting the constraint matrix in dense format, both the `nrow` and `ncol` elements of the PASAdata structure should be given.
2. Using a set of three arrays Ti, Tj, and Tx (triples), where Ti stores the row indices of the nonzero elements, Tj stores the corresponding column indices, and Tx stores the corresponding nonzero matrix entries.
3. Using a set of three arrays Ap, Ai, and Ax, where Ax is an array containing the nonzero numerical entries ordered by column of the matrix, Ai is an array containing the row indices of each nonzero in Ax with the row indices in increasing order for each column of the matrix, and Ap stores the column pointers (Ap[j] is the location of the first nonzero in Ax associated with column j).

The following elements of the PASAdata structure are used internally, and should not be touched by the user.

Name	Type	Description
cgdata	CGdata *	Input data for CG_DESCENT
ppdata	PPdata *	Input data for PPROJ.
napdata	NAPdata *	Input data for NAPHEAP.
x_created	SF *	Size ncol. Pointer to memory created for x.
lambda_created	SF *	Size nrow. Pointer to memory created for lambda.
A_created	int *	TRUE if the constraint matrix **A** is input either using triples or dense matrix formats and Ap, Ai, and Ax were malloc'd by pasa.
H_created	int *	TRUE if the Hessian of a QP is input either using triples or dense matrix formats and Hp, Hi, and Hx were malloc'd by pasa.
LP	int *	TRUE if the problem is an LP.

Structure: PPdata

The PPdata structure is input to `pproj` when solving an optimization problem of the form:

$$\min_{\mathbf{x}} \|\mathbf{y}-\mathbf{x}\|^2 \quad \text{subject to} \quad \mathbf{bl} \le \mathbf{Ax} \le \mathbf{bu}, \quad \mathbf{lo} \le \mathbf{x} \le \mathbf{hi},$$

where $\|\cdot\|$ is the Euclidean norm.

Name	Type	Description
nrow	SI	Number of rows in $\mathbf{A}$ if it exists.
ncol	SI	Number of components in x (= number of cols in $\mathbf{A}$ if it exists).
lambda	SF *	Size nrow. If parameter start_guess = 3, then lambda stores the starting guess for the constraint multiplier. If NULL, then pproj allocates memory for lambda and returns in lambda the multiplier for the constraint $\mathbf{bl} \le \mathbf{Ax} \le \mathbf{bu}$. Any allocated memory is freed by pproj_terminate.
x	SF *	Size ncol. Projection of y on the polyhedral constraint is stored in x. If NULL, then PPROJ allocates memory for x. Any allocated memory is freed by pproj_terminate.
lo	SF *	Size ncol. Lower bound in the constraint $\mathbf{lo} \le \mathbf{x}$. NULL implies use $-\infty$ for lower bound.
hi	SF *	Size ncol. Upper bound in the constraint $\mathbf{x} \le \mathbf{hi}$. NULL implies use $+\infty$ for upper bound.
bl	SF *	Size nrow. Lower bound in the constraint $\mathbf{bl} \le \mathbf{Ax}$. NULL implies use $-\infty$ for lower bound.
bu	SF *	Size nrow. Upper bound in the constraint $\mathbf{Ax} \le \mathbf{bu}$. NULL implies use $+\infty$ for upper bound.
y	SF *	Size ncol. A vector to be projected onto polyhedron $\mathbf{bl} \le \mathbf{Ax} \le \mathbf{bu}$, $\mathbf{lo} \le \mathbf{x} \le \mathbf{hi}$.
A_by_rows	SF *	Size nrow*ncol. Numerical entries in $\mathbf{A}$ (when constraint matrix input as a dense matrix).
A_by_cols	SF *	Size nrow*ncol. Numerical entries in $\mathbf{A}$ (when constraint matrix input as a dense matrix).
Ti	SI *	Size Tnz. Row indices of nonzero elements of $\mathbf{A}$ (when constraint matrix input in triples format).
Tj	SI *	Size Tnz. Column indices of nonzeros in $\mathbf{A}$.
Tx	SF *	Size Tnz. Numerical values of nonzeros in $\mathbf{A}$.
Tnz	SI	Number of entries in Ti, Tj, and Tx.
sym	int	Only used in triples format. TRUE implies matrix is symmetric and only element on main diagonal and on one side are given

Ap	SI *	Size ncol + 1. Column pointers for **A** (when constraint matrix input using sparse matrix format).
Ai	SI *	Size Ap [ncol]. Row indices in increasing order in each column for the nonzeros in **A**.
Ax	SF *	Size Ap [ncol]. Numerical entries in **A** coresponding to the row indices in Ai.
Parm	PPparm *	Parameter structure for pproj.
Stat	PPstat *	Statistics structure for pproj.
ni	SI	Number of strict inequalities where bl[i] < bu[i] (set to EMPTY in pproj_setup and pproj evaluates ni during execution).
nsing	SI	Number of column singletons in A
row_sing	SI *	Size nrow + 1. Pointers connected with column singletons.
singlo	SF *	Size nsing. Lower bounds associated with column singletons.
singhi	SF *	Size nsing. Upper bounds associated with column singletons.
singc	SF *	Size nsing. Cost vector associated with column singletons.

NOTE: The three methods for inputting the constraint matrix **A** for `pproj` are identical to the three methods used by `pasa` for input of the constraint matrix.

The following elements of the PPdata structure are used internally, and should not be touched by the user.

Name	Type	Description
priordata	PPcom	Data from a prior run of PPROJ. Used when computing a series of projections where y changes, but not the polyhedron. The prior projection can be used to get a starting guess for the active constraints in the new projection.
x_created	SF *	Size ncol. Pointer to memory created for x.
lambda_created	SF *	Size ncol. Pointer to memory created for lambda.
A_created	int *	TRUE if Ap, Ai, and Ax created by pasa_read_problem.
pproj_init_done	int *	FALSE initially, TRUE after running pproj_init.

Structure: CGdata

The CGdata structure is input to `cg_descent` when solving an unconstrained optimization problem:

$$\min_{\mathbf{x}} \ f(\mathbf{x})$$

Name	Type	Description
n	SI	Problem dimension.
x	SF *	Size n. Points to a starting guess. If NULL, then CG_DESCENT allocates memory for x and uses zero for the starting guess. The problem solution is returned in x. Any allocated memory is freed by cg_terminate.
value	function	value (SF *f, SF *x, SI n) should put the value of the objective at x in *f (not required for a QP).
grad	function	grad (SF *g, SF *x, SI n) should put the gradient of the objective at x in g (not needed required for a QP).
valgrad	function	valgrad (SF *f, SF *g, SF *x, SI n) should put the value of the objective at x in *f and the gradient in g (this routine is optional, but it can speed up the computations when the objective and its gradient are computed faster together than than they are computed separately.
c	SF *	Size n. Linear term when objective is quadratic.
Hdense	SF *	Size ncol*ncol. Numerical entries in Hessian of a QP when Hessian input as a dense matrix.
HTi	SI *	Size Hnz. Row indices of nonzero elements in Hessian of a QP when Hessian input in triples format. There is one row index entry for each nonzero in the matrix.
HTj	SI *	Size Hnz. Column indices of nonzeros in Hessian. There is one column index entry for each nonzero in the matrix.
HTx	SF *	Size Hnz. Numerical values of nonzeros in Hessian.
Hnz	SI	Number of entries in HTi, HTj, and HTx.
Hsym	int	Only used for triples; TRUE implies only element on main diagonal and on one side are given. FALSE implies that all nonzeros in the Hessian are given.
Hp	SI *	Size ncol + 1. Column pointers for Hessian (when Hessian input using sparse matrix format).

Hi	SI *	Size Hp [ncol]. Row indices in increasing order for each column of the Hessian corresponding to the nonzero matrix elements.
Hx	SF *	Size Hp [ncol]. Numerical entries in Hessian coresponding to each element of Hi.
hprod		hprod (SF *p, SF *x, SI n) computes p = H*x where H is the n by n Hessian of the objective (not needed if the nonmatrix elements are provided).
Parm	CGparm *	Parameter structure.
Stat	CGstat *	Statistics structure.
Work	SF *	NULL (default set in cg_setup) implies that the code should allocate the floating point work area, otherwise the user provides a pointer Work to the floating point work area.

The following elements of the CGdata structure are used internally, and should not be touched by the user.

Name	Type	Description
x_created	SF *	Size n. Pointer to memory created for x.
H_created	int *	TRUE if the Hessian of a QP is input either using triples or dense matrix formats and Hp, Hi, and Hx were malloc'd by cg_descent.

Structure: NAPdata

The NAPdata structure is input to `napheap` when solving an optimization problem of the form:

$$\min_{\mathbf{x}} \left(\frac{1}{2}\right) \mathbf{d}^{\mathsf{T}}\mathbf{x}^2 - \mathbf{c}^{\mathsf{T}}\mathbf{x} \quad \text{subject to} \quad \text{bl} \le \mathbf{a}^{\mathsf{T}}\mathbf{x} \le \text{bu}, \quad \mathbf{lo} \le \mathbf{x} \le \mathbf{hi},$$

where $\mathbf{d} \ge \mathbf{0}$ and $\mathbf{x}^2$ is the vector whose entries are the squares of the entries in $\mathbf{x}$.

Special Cases Treated by NAPHEAP:

1. $\mathbf{d} = \mathbf{0}$ (the objective is linear).
2. $\mathbf{d} > \mathbf{0}$ (diagonal of Hessian strictly positive).
3. $\mathbf{d} \ge \mathbf{0}$ (semidefinite Hessian).
4. $\mathbf{d} = \mathbf{1}$ (all the diagonal element are 1).

NOTE: As always, unused elements of the data structure do not need to be defined.

Name	Type	Description
n	SI	Problem dimension (number of components in x).
x	SF *	Size n. If NULL, then NAPHEAP allocates memory for x, and returns the problem solution in x. Any allocated memory is freed by napheap_terminate.
lambda	SF	Size 1. If lambda is finite, then it is treated as a starting guess for the multiplier. By default, lambda = ∞ and the starting guess is generated internally by the code. At completion, the computed multiplier is stored in lambda.
c	SF *	Size n. Vector for the linear term in the objective (NULL by default, which implies that $\mathbf{c} = \mathbf{0}$).
d	SF *	Size n. Vector for the quadratic term in the objective (NULL by default, which implies $\mathbf{d} = \mathbf{0}$).
a	SF *	Size n. Numerical entries in **a**, the linear constraint vector.
blo	SF	Size 1. Lower bound in the linear constraint.
bhi	SF	Size 1. Upper bound in the linear constraint.
lo	SF *	Size n. Lower bound in the constraint $\mathbf{lo} \le \mathbf{x}$. NULL implies use $-\infty$ for lower bound.
hi	SF *	Size n. Upper bound in the constraint $\mathbf{x} \le \mathbf{hi}$. NULL implies use $+\infty$ for upper bound.
xWork	SF *	NULL (default set in napheap_setup) implies that the code should allocate the floating point work area, otherwise the user provides a pointer xWork to the floating point work area (max size 5n).

iWork	SI *	NULL (default) implies that the code should allocate the integer work area, otherwise the user provides a pointer iWork to the integer work area (max size 4n + 1).
Parm	NAPparm *	Parameter structure for napheap.
Stat	NAPstat *	Statistics structure for napheap.

The following elements of the NAPdata structure are used internally, and should not be touched by the user.

Name	Type	Description
x_created	SF *	Pointer to memory created for x.
c_created	SF *	Pointer to memory created for c.
xWork_created	SF *	Pointer to the floating point work area created during a prior run.
iWork_created	SI *	Pointer to the integer work area created during a prior run.
akakdk	SF	sum ak*ak/dk, k = 1:n, from a previous run.

REFERENCES

[1] Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickam, *Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate*, ACM Trans. Math. Software, 35 (2009), pp. 22:1–14.

[2] T. A. Davis and W. W. Hager, *Modifying a sparse Cholesky factorization*, SIAM J. Matrix Anal. Appl., 20 (1999), pp. 606–627.

[3] ———, *Multiple-rank modifications of a sparse Cholesky factorization*, SIAM J. Matrix Anal. Appl., 22 (2001), pp. 997–1013.

[4] ———, *Row modifications of a sparse Cholesky factorization*, SIAM J. Matrix Anal. Appl., 26 (2005), pp. 621–639.

[5] ———, *Dual multilevel optimization*, Math. Program., 112 (2008), pp. 403–425.

[6] ———, *A sparse proximal implementation of the LP Dual Active Set Algorithm*, Math. Program., 112 (2008), pp. 275–301.

[7] ———, *Dynamic supernodes in sparse Cholesky update/downdate and triangular solves*, ACM Trans. Math. Software, 35 (2009), pp. 27:1–23.

[8] T. A. Davis, W. W. Hager, and J. T. Hungerford, *An efficient hybrid algorithm for the separable convex quadratic knapsack problem*, ACM Trans. Math. Software, 42 (2016), pp. 22:1–22:25.

[9] N. I. M. Gould, D. Orban, and P. L. Toint, *CUTEst: a constrained and unconstrained testing environment with safe threads for mathematical optimization*, Comput. Optim. Appl., 60 (2015), pp. 545–557.

[10] W. W. Hager, *The dual active set algorithm*, in Advances in Optimization and Parallel Computing, P. M. Pardalos, ed., North Holland, Amsterdam, 1992, pp. 137–142.

[11] ———, *Analysis and implementation of a dual algorithm for constrained optimization*, J. Optim. Theory Appl., 79 (1993), pp. 427–462.

[12] ———, *The LP dual active set algorithm*, in High Performance Algorithms and Software in Nonlinear Optimization, R. D. Leone, A. Murli, P. M. Pardalos, and G. Toraldo, eds., Dordrecht, 1998, Kluwer, pp. 243–254.

[13] ———, *The dual active set algorithm and its application to linear programming*, Comput. Optim. Appl., 21 (2002), pp. 263–275.

[14] ———, *The dual active set algorithm and the iterative solution of linear programs*, in Novel Approaches to Hard Discrete Optimization, P. M. Pardalos and H. Wolkowicz, eds., vol. 37, Fields Institute Communications, 2003, pp. 95–107.

[15] W. W. Hager and D. W. Hearn, *Application of the dual active set algorithm to quadratic network optimization*, Comput. Optim. Appl., 1 (1993), pp. 349–373.

[16] W. W. Hager and H. Zhang, *A new conjugate gradient method with guaranteed descent and an efficient line search*, SIAM J. Optim., 16 (2005), pp. 170–192.

[17] ———, *Algorithm 851: CG_DESCENT, a conjugate gradient method with guaranteed descent*, ACM Trans. Math. Software, 32 (2006), pp. 113–137.

[18] ———, *The limited memory conjugate gradient method*, SIAM J. Optim., 23 (2013), pp. 2150–2168.

[19] ———, *An active set algorithm for nonlinear optimization with polyhedral constraints*, Sci. China Math., 59 (2016), pp. 1525–1542.

[20] ———, *Projection onto a polyhedron that exploits sparsity*, SIAM J. Optim., 29 (2016), pp. 1773–1798.