`SUITEOPT 3.0.0` USER GUIDE, JUNE 10, 2023

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 [22] along with the Dual Active Set Algorithm [7, 8, 12, 13, 14, 15, 16, 17], and techniques for updating and downdating a sparse Cholesky factorization [3, 4, 5, 6, 9], 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 [10] to solve the problem
$$\min\ \left(\frac{1}{2}\right)\mathbf{x}^{\mathsf{T}}\mathbf{D}\mathbf{x} - \mathbf{c}^{\mathsf{T}}\mathbf{x} \quad \text{subject to} \quad \text{bl} \le \mathbf{a}^{\mathsf{T}}\mathbf{x} \le \text{bu}, \quad \mathbf{lo} \le \mathbf{x} \le \mathbf{hi}.$$
3. `CG_DESCENT:` The algorithms of [18, 19, 20] are used to solve an unconstrained optimization problem by the conjugate gradient method.
4. `PASA:` The polyhedral active set algorithm uses the framework of [21] to combine the previous algorithms and solve a general problem of the form (P).

The update and downdate techniques used by `PPROJ` are extracted from Timothy A. Davis' `SuiteSparse` and placed in the directory `SuiteSparseX` of `SuiteOPT`. The four different solvers in `SuiteOPT` can all be accessed through `PASA`; the `PASA` software analyzes its input data structure to determine which of the solvers, or combination of solvers, should be used to solve the given problem.

`SuiteOPT 1.0.0`, released on November 5, 2019, contained a gradient-based implementation of `PASA`, which is documented in the ACMTOMS article [23], found in the same directory as the UserGuide. Version 3.0.0 of `SuiteOPT` (June 10, 2023) includes routines to accelerate the convergence of `PASA` when the objective Hessian is available. The Hessian-based routines require an approximate solution of a symmetric linear system of equations. This is done either by either an iterative method based on the conjugate gradient method, or by a direct solver. `SuiteOPT` comes equipped with an interface to the direct solver `MUMPS` [1, 2] as well as a copy of the software itself in the `MUMPS` subdirectory. Hopefully, an interface to Harwell's MA57 will be available soon. Symmetric positive definite linear systems are handled using Davis' `SuiteSparse` software. `SuiteSparse`, `MUMPS`, and `SuiteOPT` each have their own licenses; for an overview, see the file LICENSE.txt in the top level `SuiteOPT` directory.

2. General Instructions for Using `SuiteOPT`. There are three different ways to run `SuiteOPT`:

(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 platform CUTEst (Constrained and Unconstrained Testing Environment with Safe Threads) of Gould, Orban, and Toint [11].

Regardless of the software interface, an important parameter when setting up a problem is the number infinity. `SuiteOPT` employs a definition of floating point infinity 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 set "`hi[i] = inf`", while in MATLAB set "`hi(i) = inf`". If all the components of $\mathbf{x}$ have no upper bound, then leave **hi** undefined; by default, all the components of **hi** are $+\infty$. Lower bounds are treated in a similar fashion. The main user setup file is `Userconfig.mk`, found in the directory `SuiteOPTconfig`. As explain in the `Userconfig.mk` comments, the user can override the default definition of infinity by adding a flag to the parameter `OPTFLAGS`.

If the Hessian-based `PASA` is used, then symmetric solver should be specified as another flag of `OPTFLAGS`. Currently, the only option is the `MUMPS` solver, which is specified by the flag "`-DUSE_MUMPS`". `MUMPS` utilizes the `SCOTCH` ordering routines when they are available, in which case their location is also specified in Userconfig.mk. MATLAB has its own built-in versions of the BLAS and LAPACK, and automatically employs 64 bit integers. When working in C or CUTEst, Userconfig.mk should specify the location of the user's BLAS and LAPACK; to compile `SuiteOPT` with 64 bit integers, add the flag "`-DDLONG`" to `OPTFLAGS`, and be sure to choose versions of the BLAS and LAPACK libraries with the "ilp64" routines, rather than "lp64" routines. Also, when working in C or CUTEst, the user should specify their C and Fortran compilers as indicated at the top of `Userconfig.mk`.

3. Installation and Use in MATLAB.

To compile the `SuiteOPT` solvers for 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, change your directory to the MATLAB directory that is found inside the solver's directory and startup MATLAB. Your MATLAB code for solving a problem should create a structure that contains the problem data. 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`. Note that all the solvers can be accessed through `pasa`; if the problem input to `pasa` is an unconstrained problem or a knapsack problem or a projection problem, then `pasa` uses `cg_descent`, `napheap`, or `pproj` respectively to solve the problem.

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 `x` $\leq$ `hi` does not exist and `data` denotes the input data structure for a solver, then do not specify a value for `data.hi`. Often, there is no need to specify dimensional information such as `data.nrow` (number of linear inequality constraints) or `data.ncol` (the dimension of $\mathbf{x}$) since MATLAB may deduce this information from the input data. On the other hand, in some weird cases, it could be necessary to provide the dimension of $\mathbf{x}$. For example, if $\mathbf{A}$ is a sparse matrix with some trailing columns of zeros, there is a potential for `pasa` to conclude erroneously that the dimension of $\mathbf{x}$ is equal to the number of nonzero columns in $\mathbf{A}$. By specifying the dimension of $\mathbf{x}$ in `data.ncol`, this confusion is eliminated.

The definitions of the default parameters for a solver appear in the file ending in `default.c` 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 following 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. When using `pasa`, on the other hand, the corresponding statement is `pasadata.solver.parameter = (new value) ;`

where `solver` is either `pasa`, `pproj`, `napheap`, or `cg`. See the demo files in the MATLAB subdirectories of each solver for examples showing how to set up a problem.

If the problem objective is not quadratic or linear, then the user needs to provide functions to evaluate the objective and its gradient. The names of these functions should be stored in `pasadata.value` and `pasadata.grad`. If the user wishes to provide the objective Hessian at a given `x`, then the Hessian function should evaluate the nonzero elements of the Hessian and store them in triples format; that is, if `nnz` nonzeros are on the main diagonal and on one side, then a `nnz` by 3 matrix is constructed in which each row contains a row index, a column index, and the corresponding Hessian element, in this order. The name of the Hessian function should be stored in `pasadata.hessian`.

If the objective is either quadratic or linear, then the objective is fully specified by giving the cost vector associated with the linear term along with the square, symmetric Hessian matrix. The cost vector should be dense, while the Hessian matrix should be either a square sparse matrix, created using MATLAB's `sparse` function, or a dense matrix. All the elements of the Hessian, both above and below the diagonal, should be provided. If the user wishes to use triples to input the elements of Hessian on the main diagonal and one side, then he could put the name of the Hessian function in `pasadata.hessian` while also specifying that the problem is a QP by setting `pasadata.pasa.QP = 1`. Again, see the demo files in the MATLAB subdirectories of a solver for examples.

4. Installation and Use in C. Once the `Userconfig.mk` file is set up, all the `SuiteOPT` C codes can be compiled at once from the top level `SuiteOPT` directory by typing "`make`". Once `SuiteSparse` and `MUMPS` have been compiled from the top level `SuiteOPT` directory, then after any changes in a directory associated with a `solver`, it can be recompiled 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 of a solvers all have the layout shown below:

```
/* Create pointer to the data structure for a solver */
    SOLVERdata *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) ;
/* where solver_name is 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) ;
```

See the `Demo` subdirectory of a solver for examples showing how to set up a problem.

Similar to the strategy for using `SuiteOPT` in MATLAB, if the problem objective is neither quadratic nor linear, then the user needs to provide functions to evaluate the objective and its gradient; the names of these functions should be inputs to `pasadata->value` and `pasadata->grad` respectively. On the other hand, the C setup provides greater flexibility for input of the Hessian when compared to MATLAB. The Hessian function should build a `SOPT_matrix` structure, denoted here as `H`, containing the nonzero matrix elements. The structure accepts accepts either a sparse matrix, a dense matrix, or a matrix in triples format. When using the sparse or dense formats, the entire matrix must be provided, including matrix elements both below and above the diagonal. When using the triples format, three arrays `H->rows`, `H->cols`, and `H->vals` are constructed with the row and column indices of the nonzero matrix elements and their associated `vals` (numerical values). The rows and columns can be specified either with Fortran indexing, where the first row and column are 1 (`H->fortran = TRUE` by default), or with C indexing (`H->fortran = FALSE`), where the first row and column are 0. If `H->sym = TRUE` (default for the Hessian), then the matrix is symmetric and only elements on the diagonal and one side are given, while if `H->sym = FALSE`, then the entire matrix is should be given. Again, see the `Demo` subdirectory of a solver for examples.

If the objective is either quadratic or linear, then the objective is fully specified by giving the cost vector associated with the linear term along with the square, symmetric Hessian matrix. The linear cost vector `data->c` is an array of scalars, while the Hessian matrix `data->H`, an `SOPT_matrix`, should be either a sparse matrix, a dense matrix, or a matrix in triples format.

5. Installation and Use in CUTEst.

Both `pasa` or `cg_descent` can be installed in CUTEst, provided the user has downloaded and installed CUTEst on his computer. 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="..."
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 'full_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 from the `runcutest` file 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 the call to `pasa`. Similarly, 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.
```

After adding new parameter values in `pasa_main.c` or `cg_descent_main.c`, the CUTE installation should be recompiled by typing "`make`" in the CUTEst subdirectories where the parameters were changed.

6. Parameters. The names of the parameters, their definitions, and their default values can be found in the solver 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, when `cg_descent` is invoked from `pasa`, then the user could change the value of `cg_descent` LBFGS memory from its default value 11 to 15 using the statement

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

Note that 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.

7. The Input Data Structures. The input data structures for the solvers are defined and documented in the Include directories for each solver, in the file solver.h. Search for "data_struct" in the ".h" file. Note that the structure `SOPT_matrix` is defined in the file `SuiteOPTconfig/sopt.h`. 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 matrices, "`int`" would need to be changed to "`long`". This is a done by adding the flag "`-DDLONG`" to `OPTFLAGS` in the `Userconfig.mk` file. This flag will make all the integers throughout `SuiteOPT` long integers, including integers in `MUMPS`, `SuiteSparse`, and `SSM`.

REFERENCES

[1] P. R. Amestoy, A. Buttari, J.-Y. L'Excellent, and T. Mary, *Performance and scalability of the block low-rank multifrontal factorization on multicore architectures*, 45 (2019), pp. 2:1–2:26.

[2] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent, *A fully asynchronous multifrontal solver using distributed dynamic scheduling*, SIAM J. Matrix Anal. Appl., 23 (2001), pp. 15–41.

[3] 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.

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

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

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

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

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

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

[10] 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.

[11] 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.

[12] 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.

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

[14] ———, *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.

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

[16] ———, *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.

[17] 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.

[18] 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.

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

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

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

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

[23] ———, *A gradient-based implementation of the polyhedral active set algorithm*, ACM Trans. Math. Software, (2023), p. https://doi.org/10.1145/3583559.