



Alfonso: Matlab Package for Nonsymmetric Conic Optimization

Dávid Papp,^a Sercan Yildiz^b

^aDepartment of Mathematics, North Carolina State University, Raleigh, North Carolina 27695; ^bProduct Analytics, Qontigo, New York, New York 10004

Contact: dpapp@ncsu.edu,  <https://orcid.org/0000-0003-4498-6417> (DP); syildiz@qontigo.com,  <https://orcid.org/0000-0003-2004-1101> (SY)

Received: August 4, 2020

Revised: August 19, 2020; January 18, 2021; January 31, 2021

Accepted: February 5, 2021

Published Online in Articles in Advance: September 8, 2021

<https://doi.org/10.1287/ijoc.2021.1058>

Copyright: © 2021 INFORMS

Abstract. We present *alfonso*, an open-source Matlab package for solving conic optimization problems over nonsymmetric convex cones. The implementation is based on the authors' corrected analysis of a method of Skajaa and Ye. It enables optimization over any convex cone as long as a logarithmically homogeneous self-concordant barrier is available for the cone or its dual. This includes many nonsymmetric cones, for example, hyperbolicity cones and their duals (such as sum-of-squares cones), semidefinite and second-order cone representable cones, power cones, and the exponential cone. Besides enabling the solution of problems that cannot be cast as optimization problems over a symmetric cone, algorithms for nonsymmetric conic optimization also offer performance advantages for problems whose symmetric cone programming representation requires a large number of auxiliary variables or has a special structure that can be exploited in the barrier computation. The worst-case iteration complexity of *alfonso* is the best known for nonsymmetric cone optimization: $\mathcal{O}(\sqrt{\nu} \log(1/\epsilon))$ iterations to reach an ϵ -optimal solution, where ν is the barrier parameter of the barrier function used in the optimization. *Alfonso* can be interfaced with a Matlab function (supplied by the user) that computes the Hessian of a barrier function for the cone. A simplified interface is also available to optimize over the direct product of cones for which a barrier function has already been built into the software. This interface can be easily extended to include new cones. Both interfaces are illustrated by solving linear programs. The oracle interface and the efficiency of *alfonso* are also demonstrated using an optimal design of experiments problem in which the tailored barrier computation greatly decreases the solution time compared with using state-of-the-art, off-the-shelf conic optimization software.

Summary of Contribution: The paper describes an open-source Matlab package for optimization over nonsymmetric cones. A particularly important feature of this software is that, unlike other conic optimization software, it enables optimization over any convex cone as long as a suitable barrier function is available for the cone or its dual, not limiting the user to a small number of specific cones. Nonsymmetric cones for which such barriers are already known include, for example, hyperbolicity cones and their duals (such as sum-of-squares cones), semidefinite and second-order cone representable cones, power cones, and the exponential cone. Thus, the scope of this software is far larger than most current conic optimization software. This does not come at the price of efficiency, as the worst-case iteration complexity of our algorithm matches the iteration complexity of the most successful interior-point methods for symmetric cones. Besides enabling the solution of problems that cannot be cast as optimization problems over a symmetric cone, our software can also offer performance advantages for problems whose symmetric cone programming representation requires a large number of auxiliary variables or has a special structure that can be exploited in the barrier computation. This is also demonstrated in this paper via an example in which our code significantly outperforms Mosek 9 and SCS 2.

History: Accepted by Ted Ralphs, Area Editor for Software Tools.

Funding: This material is based upon work supported by the National Science Foundation [Grants DMS-1719828 and DMS-1847865] (D. Papp). The material is also based upon work partially supported by the National Science Foundation [Grant DMS-1638521] to the Statistical and Applied Mathematical Sciences Institute (S. Yildiz).

Supplemental Material: The software that supports the findings of this study is available within the paper and its Supplementary Information [<https://doi.org/10.1287/ijoc.2021.1058>] or is available from the IJOC GitHub software repository (<https://github.com/INFORMSjoc>) at [<http://dx.doi.org/10.5281/zenodo.4422116>].

Keywords: conic optimization • interior-point method • self-concordant barrier • nonsymmetric cone • software

1. Introduction

We present *alfonso*, an open-source, Octave-compatible Matlab package for solving optimization problems over (not necessarily symmetric) convex cones. More precisely, *alfonso* can be used to solve primal-dual pairs of optimization problems of the form

$$\begin{aligned} & \underset{\mathbf{x} \in \mathbb{R}^n}{\text{minimize}} && \mathbf{c}^\top \mathbf{x} \\ & \text{subject to} && \mathbf{A}\mathbf{x} = \mathbf{b} \\ & && \mathbf{x} \in K \\ & \underset{\mathbf{s} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^m}{\text{maximize}} && \mathbf{b}^\top \mathbf{y} \\ & \text{subject to} && \mathbf{A}^\top \mathbf{y} + \mathbf{s} = \mathbf{c} \\ & && \mathbf{s} \in K^*, \end{aligned} \quad (\text{P}) \quad (\text{D})$$

where K is a full-dimensional, pointed, closed, convex cone and K^* is its dual. The only additional assumption K needs to satisfy is that *an efficient algorithm to compute the gradient and the Hessian of some logarithmically homogeneous self-concordant barrier function of K is available* (see Section 1.1). As an automatic generalization, it is also sufficient to have such a barrier function for only the dual cone K^* , because we can apply *alfonso* to the dual problem. A feasible initial point is not required, only an initial point in the interior of K .

We may assume without loss of generality that $\text{rank}(\mathbf{A}) = m$. If $\text{rank}(\mathbf{A}) < m$, then depending on whether $\mathbf{b} \in \text{range}(\mathbf{A})$ or not, either the equality constraints in the primal problem are inconsistent or some of the equalities are redundant and can be removed.

The set of problems *alfonso* can solve includes optimization over many nonsymmetric cones of great interest, for example, hyperbolicity cones of efficiently computable hyperbolic polynomials (Renegar 2004) and their duals, sum-of-squares cones (Blekherman et al. 2013, chapter 3), \mathcal{L}_p cones (Glineur and Terlaky 2004) and other flavors of (generalized) power cones (Roy and Xiao 2018), and the exponential cone (Chares 2009).

To maximally take advantage of this level of generality, *alfonso* can be interfaced directly with a function handle to a *membership and barrier function oracle*, a Matlab function that computes whether a given point is in the interior of the cone and for interior points computes the gradient and Hessian of an appropriate barrier function. For convenience, we have also created a simplified interface that allows the user to specify K as the direct product of known cones for which *alfonso* already has oracles implemented. Through this interface, *alfonso* is easily extensible: the barrier function of any cone may be implemented and then added to the list of cones accepted by this interface, by adding only a few additional lines to *alfonso*'s source code.

To our knowledge, there are very few other software for nonsymmetric conic optimization and even fewer that can be extended by the user to allow for

optimization over new cones. SCS (O'Donoghue et al. 2016) is a first-order, operator splitting method that can solve conic optimization problems over any cone that is easy to orthogonally project to and currently supports the exponential and power cones in addition to symmetric cones. ECOS (Domahidi et al. 2013) is a second-order cone programming software whose latest version also handles exponential cone constraints. DDS (Karimi and Tunçel 2019) is a recent convex optimization solver that can solve a wide variety of problems that have no known representations as symmetric cone programs but with an entirely different approach to domain definition. Hypatia is a recently announced nonsymmetric cone optimization solver written in Julia, based on a similar algorithm as *alfonso* (Coey et al. 2020); but at the time of writing this paper, the code does not appear to be publicly available.¹ In the commercial domain, Mosek 9 is capable of solving conic optimization problems with any combination of symmetric, exponential, and power cone constraints (Mosek ApS 2019); but it is neither open source nor extensible with new cones.

Besides enabling the solution of problems that cannot be cast as optimization over a symmetric cone, algorithms for nonsymmetric conic optimization can also offer performance advantages for problems that can be written as optimization problems over symmetric cones. This is the case, for example, when the equivalent representation as an optimization problem over a symmetric cone requires an extended formulation with a large number of auxiliary variables or when the representation has some special structure that all-purpose optimization software often do not take advantage of, such as Hankel, Toeplitz, or low-rank structures in semidefinite programming. An example of a family of optimization problems that greatly benefit from a nonsymmetric cone optimization approach is *sum-of-squares optimization*; this application was the initial motivation for the development of *alfonso* (Papp and Yıldız 2019). In Section 3, we demonstrate another application in which *alfonso* is several orders of magnitude more efficient than the straightforward semidefinite programming approach. Additional complex examples with extensive computational results comparing an earlier version of the code with state-of-the-art off-the-shelf interior-point solvers can be found in the authors' recent work on polynomial optimization (Papp 2019, Papp and Yıldız 2019).

The implementation is based on an interior-point method (IPM) originally proposed by Skajaa and Ye (2015) and subsequently improved by the authors (Papp and Yıldız 2017). The iteration complexity of the method matches the iteration complexity of popular algorithms for symmetric cone optimization.

The source code can be found in the supplemental material and is available for download at <https://github.com/INFORMSJoC/2021.1058>.

1.1. Mathematical Background

Definition 1 (Proper Cone). We say that a set $K \subseteq \mathbb{R}^n$ is a *cone* provided that for every $\mathbf{x} \in K$ and $\lambda \geq 0$ we also have $\lambda \mathbf{x} \in K$. A cone is *proper* if it satisfies all of the following: it is closed, convex, has a nonempty interior, and does not contain a line.

Aside from closedness and convexity, the remaining assumptions on K are essentially without loss of generality in the sense that every finite-dimensional closed convex optimization problem can be equivalently written in the form of (P) with an appropriate choice of the full-row-rank matrix \mathbf{A} , vectors \mathbf{b}, \mathbf{c} , and the proper cone K .

Definition 2 (Logarithmically Homogeneous Self-Concordant Barrier Function, LHSCB). Let K° denote the interior of K . A function $f : K^\circ \mapsto \mathbb{R}$ is a *barrier function* if $f(\mathbf{x}_i) \rightarrow \infty$ for every sequence $\mathbf{x}_1, \mathbf{x}_2, \dots$ of points $\mathbf{x}_i \in K^\circ$ converging to a boundary point of K . A barrier is *self-concordant* if it is convex, three times continuously differentiable, and if for every $\mathbf{x} \in K^\circ$ and $\mathbf{h} \in \mathbb{R}^n$ the inequality $|D^3f(\mathbf{x})[\mathbf{h}, \mathbf{h}, \mathbf{h}]| \leq 2D^2f(\mathbf{x})[\mathbf{h}, \mathbf{h}]^{3/2}$ holds. The self-concordant barrier f is called *logarithmically homogeneous* if there exists a scalar ν such that for every $\mathbf{x} \in K^\circ$ and $t > 0$ we have $f(t\mathbf{x}) = f(\mathbf{x}) - \nu \ln t$. The scalar ν is called the *barrier parameter* of K .

As a shorthand, we say that f is a ν -LHSCB for K if f is a logarithmically homogeneous self-concordant barrier whose domain is the interior of a proper convex cone K and if the barrier parameter of f is ν . For the interested reader, the monograph from Nesterov and Nemirovskii (1994) provides a comprehensive treatment of LHSCBs. Renegar’s treatment (Renegar 2001) of the subject is also excellent. Many fundamental cones in the application of convex optimization have known and easily computable LHSCBs.

Example 1. The following examples are proper convex cones with the additional property that either the cone or its dual has a known LHSCB with easily computable derivatives. Only the first three cones are symmetric; the remaining ones are not.

Example 1.1. The function $f(x) = -\ln x$ is an LHSCB for \mathbb{R}_+ , and more generally $f(\mathbf{x}) = -\sum_{i=1}^n \ln x_i$ is an n -LHSCB for \mathbb{R}_+^n .

Example 1.2. The function

$$f(\mathbf{x}) = -\ln\left(x_0^2 - \sum_{i=1}^n x_i^2\right)$$

is a two-LHSCB for the *second-order cone*

$$\mathcal{Q}_{n+1} \stackrel{\text{def}}{=} \{(x_0, \dots, x_n) \mid x_0 \geq \|(x_1, \dots, x_n)\|\}.$$

Note that its barrier parameter is independent of the dimension n .

Example 1.3. The function

$$f(\mathbf{X}) = -\ln \det \mathbf{X}$$

is an n -LHSCB for the cone of $n \times n$ positive semidefinite real symmetric matrices.

Example 1.4. The *exponential cone* is the three-dimensional cone

$$\mathcal{E} \stackrel{\text{def}}{=} \text{cl}(\{\mathbf{x} \in \mathbb{R}_+^2 \times \mathbb{R} \mid x_1 > x_2 e^{x_3/x_2}\}).$$

The function

$$f(\mathbf{x}) = -\ln(x_1) - \ln(x_2) - \ln(x_2 \ln(x_1/x_2)) - x_3$$

is a three-LHSCB for this cone (Chares 2009, chapter 2).

Example 1.5. Suppose $\lambda = (\lambda_1, \dots, \lambda_n) \in \mathbb{R}^n$ satisfies $\lambda_i > 0$ for each i and $\sum_{i=1}^n \lambda_i = 1$. Then the (*generalized*) *power cone* with *signature* λ is the convex cone defined as

$$\mathcal{P}_\lambda \stackrel{\text{def}}{=} \left\{ (\mathbf{x}, z) \in \mathbb{R}_+^n \times \mathbb{R} \mid |z| \leq \prod_{i=1}^n x_i^{\lambda_i} \right\}. \quad (1)$$

The function

$$f(\mathbf{x}, z) = -\ln\left(\prod_{i=1}^n x_i^{2\lambda_i} - z^2\right) - \sum_{i=1}^n (1 - \lambda_i) \ln(x_i)$$

is an $(n + 1)$ -LHSCB for this cone. This was first proven by Roy and Xiao (2018), who also study a number of related cones. The dual cone of \mathcal{P}_λ is identical to the cone known in algebraic geometry as the *sum of nonnegative circuit polynomials cone* (Iliman and de Wolff 2016).

Example 1.6. A homogeneous n -variate polynomial h of degree d is said to be *hyperbolic with respect to the point* $\mathbf{e} \in \mathbb{R}^n$ if $h(\mathbf{e}) > 0$ and if for every $\mathbf{x} \in \mathbb{R}^n$, the univariate polynomial $h(\mathbf{x} + t\mathbf{e})$ has only real roots. The corresponding *hyperbolicity cone* is the set

$$\Lambda_{h,\mathbf{e}}^+ \stackrel{\text{def}}{=} \{\mathbf{x} \in \mathbb{R}^n \mid h(\mathbf{x} + t\mathbf{e}) > 0 \quad \forall t > 0\}.$$

It can be shown that $\Lambda_{h,\mathbf{e}}^+$ is a proper convex cone for which $-\ln h(\cdot)$ is a d -LHSCB (Güler 1997). Because the determinant is a hyperbolic polynomial (with respect to the identity matrix) whose hyperbolicity cone is the semidefinite cone, optimization over hyperbolicity cones is a generalization of semidefinite programming.

Example 1.7. If K_1, \dots, K_k are proper convex cones in \mathbb{R}^n whose interiors have a nonempty intersection and f_i is a ν_i -LHSCB for K_i ($i = 1, \dots, k$), then $\sum_{i=1}^k f_i$ is a ν -LHSCB for the intersection $\bigcap_{i=1}^k K_i$, with barrier parameter $\nu = \sum_{i=1}^k \nu_i$.

Example 1.8. If $K_1 \subseteq \mathbb{R}^{n_1}, \dots, K_k \subseteq \mathbb{R}^{n_k}$ are proper convex cones and f_i is a ν_i -LHSCB for K_i ($i = 1, \dots, k$), then

$\sum_{i=1}^k f_i$ is a ν -LHSCB for the product cone $K_1 \times \dots \times K_k$, with barrier parameter $\nu = \sum_{i=1}^k \nu_i$.

Example 1.9. Let $K \subseteq \mathbb{R}^n$ be a proper convex cone with an LHSCB f and L be a linear subspace of \mathbb{R}^n that intersects K° . Then $f|_L$, the restriction of f to the subspace L , is an LHSCB for $K \cap L$. Denoting the orthogonal projection matrix onto L by \mathbf{P}_L , the gradient and Hessian of this barrier function are $\mathbf{P}_L \nabla f(\mathbf{x})$ and $\mathbf{P}_L \nabla^2 f(\mathbf{x}) \mathbf{P}_L$, respectively.

In a different notation, if $K \subseteq \mathbb{R}^m$ is a proper convex cone with an LHSCB f_K and $\mathbf{A} \in \mathbb{R}^{m \times n}$ is a matrix whose range space intersects K° , then the cone

$$C = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{A}\mathbf{x} \in K\}$$

is a proper convex cone and $f_C(\mathbf{x}) = f_K(\mathbf{A}\mathbf{x})$ is an LHSCB for C whose gradient and Hessian are easily computable from the gradient and Hessian of f_K . A notable special case is when K is the positive semidefinite cone; sets C that can be written in this form are called *spectrahedral cones* (Blekherman et al. 2013, chapter 2).

Examples of spectrahedral cones that benefit from a nonsymmetric cone optimization approach include the epigraph of the spectral norm (Nesterov and Nemirovskii 1994, proposition 5.4.6), also known as the *spectral norm cone*, and the cone of *sum-of-squares polynomials*, for which the LHSCB inherited from semidefinite programming is particularly efficiently computable when the polynomials are represented in an interpolant basis (Papp and Yıldız 2019).

Example 1.9 is particularly notable because even though we have easily computable LHSCBs for these cones, there does not appear to be any straightforward way to construct easily computable barrier functions for their dual cones. Additional techniques to construct LHSCBs for convex cones from known LHSCBs of simpler cones can be found in Nesterov and Nemirovskii (1994, chapter 5).

1.2. The Algorithm and Its Complexity

The implementation is based on an IPM applied to a homogeneous self-dual embedding of (P)-(D) that was originally proposed by Skajaa and Ye (2015) and subsequently improved by the authors. We refer the reader to Papp and Yıldız (2017) for the details of the algorithm and its analysis and Papp and Yıldız (2019, section 2) for a brief summary.

This method is one of the theoretically most efficient algorithms applicable to nonsymmetric cone programming; its worst-case iteration complexity matches the iteration complexity of successful IPMs for symmetric cones. The main convergence and complexity result from our analysis (Papp and Yıldız 2019, proposition 2.1) can be summarized as follows: the number of iterations and number of calls to the

membership and barrier function oracle required to reduce the primal and dual infeasibility and complementarity metrics to ε times their initial value are $\mathcal{O}(\sqrt{\nu} \log(1/\varepsilon))$, where ν is the barrier parameter of the barrier function. For most cones, this means $\mathcal{O}(\sqrt{n} \log(1/\varepsilon))$ iterations and oracle calls.

2. Interfaces

2.1. Installation

Alfonso is entirely written in Matlab m-code and is thus portable and easy to install: unzip the downloaded files in any directory and add the *alfonso* directory to the Matlab (or Octave) path. Some of the examples that are not detailed in this paper require additional packages.

2.2. Input Interfaces

An instance of the optimization Problem (P) can be described by the problem data $(\mathbf{A}, \mathbf{b}, \mathbf{c})$ and the cone K . Because of the level of generality *alfonso* is aimed at, there are two ways to specify the cone when interfacing with the code.

2.2.1. The Oracle Interface. The cone K can be specified using a *membership and barrier function oracle*, which is a subroutine with the following signature:

```
1 function [in, g, H, L] = oracle(x, bParams)
```

The first input argument \mathbf{x} represents the primal vector \mathbf{x} , which is the oracle's input.

The second argument *bParams* is an optional one that can be used to specify other parameters for the barrier function. For example, if *oracle* implements an LHSCB for the generalized power cone with signature λ (recall Example 1.5), then it is convenient to pass λ as a parameter. If necessary, multiple parameters that cannot be conveniently passed as a single vector can be passed using a *struct* for *bParams*.

If f denotes the LHSCB implemented in *oracle*, then the four outputs of the oracle are

- *in*: a Boolean flag that is *true* if $\mathbf{x} \in K^\circ$ and *false* otherwise.
- *g*: a vector whose value is the gradient $\nabla f(\mathbf{x})$ if $\mathbf{x} \in K^\circ$. Its value is ignored otherwise.
- *H*: a matrix whose value is the Hessian $\nabla^2 f(\mathbf{x})$ if $\mathbf{x} \in K^\circ$. Its value is ignored otherwise.
- *L*: a lower triangular Cholesky factor of the Hessian.

Alfonso frequently calls the oracle with only the first or the first two output arguments. Unless all output parameters can be computed very efficiently, it is highly recommended that the oracle only computes

the necessary output arguments, using Matlab’s `nargout` feature.

If \mathbf{H} or \mathbf{L} is sparse, they should be computed as sparse matrices. The Cholesky factor can often be determined in closed form; otherwise one may always resort to the following generic code snippet to compute \mathbf{L} from \mathbf{H} :

```
if nargout > 3
    [L, err] = chol(H, 'lower');
    if err > 0
        in = false; g = NaN; H = NaN; L = NaN;
        return;
    end
end
end
```

Lastly, `alfonso` needs a starting point for the optimization. Only a primal initial point is needed, in the interior of K ; `alfonso` automatically computes an initial primal-dual iterate on the central path.

Having all of this ready, the optimization problem can be solved by calling

```
alfonso(probData, x0, @oracle,
        bParams, opts)
```

The first argument is a Matlab `struct` with three mandatory fields, A , b , and c ; it contains the problem data. The second argument is the initial point. The third is a function handle to the membership and barrier function `oracle`, whereas the fourth (optional) argument is the parameter to be passed to the oracle as its second argument.

The last optional argument `opts` is a structure specifying the optimization options. See Section 2.4 for more details on algorithmic and other options and Section 2.5 for a complete example of how an optimization problem can be set up and solved using this interface.

2.2.2. The Simple Interface. The goal of the simple interface is to facilitate the reuse of previously implemented barrier functions. In the simple interface, the cone is specified as a Cartesian product $K_1 \times \dots \times K_k$ of known cones K_i passed to `alfonso` as a Matlab cell array of structures whose i th element describes K_i .

In the cone array K , each element $K\{i\}$ has two mandatory fields: $K\{i\}.type$, a string that specifies the cone K_i , and $K\{i\}.dim$, a string that specifies the dimension of the cone. The already built-in cones include

- `type = 'l'` or `'lp'` represents the nonnegative orthant.
- `type = 'soc'` or `'socp'` represents the second-order cone.

- `type = 'exp'` represents the exponential cone.
- `type = 'gpow'` represents a generalized power cone (defined in Example 1.5). The parameter λ must be specified in the field $K\{i\}.lambda$ as an additional vector.

Deviating slightly from the theory, variables in `alfonso` are allowed to be *free*, that is, not to be a member of any cone. This can be specified using $K\{i\}.type = 'free'$. Free variables are handled by placing them in a second-order cone using a single additional dummy variable, which is a common strategy in conic optimization attributed to Andersen (2002) and is also used, for instance, in SeDuMi.

For example, the cell array

```
K{1}.type = 'socp'; % second-order cone
K{1}.dim = 10;
K{2}.type = 'free'; % free variables
K{2}.dim = 6;
K{3}.type = 'lp'; % nonnegative orthant
K{3}.dim = 10;
K{4}.type = 'exp'; % exponential cone,
always 3-dimensional
```

defines the cone $K = \mathcal{Q}_{10} \times \mathbb{R}^6 \times \mathbb{R}_+^{10} \times \mathcal{E}$.

When K is the Cartesian product of known cones, it is not necessary to provide an initial point; `alfonso` defaults to the concatenation of known, “central,” interior points of these cones. The syntax of the simple interface is

```
alfonso_simple(c, A, b, K, x0, opts)
```

where the first four arguments are as described above, $x0$ is the optional initial point (that can be set to `[]` for the default value) and the also optional `opts` argument is the same options structure as used in the oracle interface. (See Section 2.4 for more details on the options.)

2.3. Outputs

Regardless of which interface is used, `alfonso` returns a single structure as a result with over 20 fields that contain various diagnostic elements and information about the optimization process in addition to the primal and dual solutions. The comments in the header of `alfonso.m` contain a detailed description of all of them; here we only summarize the most important ones:

- `status`: an integer representing the solver status when the solver stopped. Its value is one if an approximately optimal solution was found;
- `statusString`: the same information as `status` but in a human-readable format
- `x`, `s`, and `y`: the final primal and dual iterates;

Figure 1. (Color online) A Membership and Barrier Function Oracle for Solving Linear Programs in Standard Form

```

1 function [in, g, H, L] = gH_lp(x, ~)
2     n = length(x);
3     in = min(x)>0;
4     if in
5         g = -1./x;
6         H = sparse(1:n,1:n,x.^(-2),n,n,n);
7         L = sparse(1:n,1:n,-g,n,n,n);
8     else
9         g = NaN; H = NaN; L = NaN;
10    end
11 end

```

- `pObj` and `dObj`: final primal and dual objective function values;
- `time`: the solution (wall-clock) time in seconds.

2.4. Algorithmic and Other Options

Options for `alfonso` can be set using the optional last argument to the `alfonso()` or `alfonso_simple()` function. This argument is a structure (`struct`) with fields set to their desired values. Any options not specified this way will take their default values, which are detailed in the header of `alfonso.m`. The options that the users are most likely to want to change are the following:

- `optimTol`: optimality tolerance ϵ ; default value: $1e-6$;
- `verbose`: a Boolean flag controlling the output level; default value: `1`.

The remaining options adjust various parameters of the algorithm (such as the line search procedure); these are documented in the header of `alfonso.m` and are omitted here, as changing them is only recommended in very particular situations.

2.5. A Minimal Example: Solving Linear Programs

In this section, we use the toy example of solving linear programs in standard form to illustrate how problem data are structured for each of the two interfaces. This example (with additional comments) is also included in the package in the files `random_lp.m` and `random_lp_simple.m`. Additional examples can be found in the `examples` subdirectory of the code.

2.5.1. The Oracle Interface. To solve a linear program using the oracle interface, the user must implement a Matlab function that solves the membership problem and (for points in the interior) computes the gradient and factors the Hessian of an LHSCB for the nonnegative orthant $K = \mathbb{R}_+^n$. For the nonnegative orthant, we use the logarithmic barrier f given by $f(\mathbf{x}) = -\sum_{i=1}^n \ln(x_i)$. A straightforward implementation

is shown on Figure 1. For efficiency, we use sparse matrices. The second input argument of the barrier function (that allows the passing of parameters) is not used.

With the oracle `gH_lp()` ready, a linear program in standard form, with problem data \mathbf{A} , \mathbf{b} , and \mathbf{c} as in (P), can be solved by simply calling

```

probData=struct('c',c,'A',A,'b',b);
results=alfonso(probData,x0,@gH_lp);

```

where \mathbf{x}_0 is any componentwise positive initial point, for example, the all-ones vector `ones(n,1)`. The optimal solution will be returned in `results.x`.

If any options are to be changed, the second line needs to include the options structure. In the following example, we decrease the optimality tolerance:

```

opts.optimTol=1e-7;
results=alfonso(probData,x0,@gH_lp, [],opts);

```

The empty list in the fourth argument is a placeholder for the optional parameters to pass to the function `gH_lp`, which is not used in this example.

2.5.2. The Simple Interface. Using the simple interface, the user only needs to represent the n -dimensional nonnegative orthant in a cone structure (cell array) as follows:

```

K{1}=struct('type','lp','dim',n);
results=alfonso_simple(c,A,b,K,x0,opts);

```

Note that using the simple interface, the fourth argument `x0` may be replaced by `[]`, in which case `alfonso` will choose the default value (in this example, the all-ones vector).

Figure 2. (Color online) A Membership and Barrier Function Oracle for the E-Optimal Design Example

```

1 function [in, g, H, L] = ex_oracle_design(tx, pars)
2 % This function implements a membership and barrier function oracle for
3 % the E-optimal design example e_design.m
4 %
5 % INPUT
6 % tx:           column vector representing [t; x(1); ...; x(n)]
7 % pars:        structure with a single field pars.v
8 %             pars.v is a two-dimensional array whose ith column
9 %             v(:,i) represents the ith design vector (i=1,...,p)
10 t = tx(1);
11 x = tx(2:end);
12 [n,p] = size(pars.v);
13
14 % in the cone?
15 if any(x <= 0)
16     in = false;  g = NaN;  H = NaN;  L = NaN;  return
17 end
18
19 Ax = -t*eye(n) + pars.v*diag(x)*pars.v';
20
21 [L,err] = chol(Ax,'lower');
22 if err > 0
23     in = false;  g = NaN;  H = NaN;  L = NaN;  return
24 else
25     in = true;
26 end
27
28 % compute g and H if required
29 if nargin > 1
30     g = [0; -1./x];
31
32     Li = inv(L);
33     g(1) = Li(:)'*Li(:);
34     w = L \ pars.v;
35     for i=1:p
36         g(i+1) = -w(:,i)'*w(:,i);
37     end
38
39     % compute H and L if required
40     if nargin > 2
41         H = diag([0; x.^(-2)]);
42
43         invAx = Li'*Li;
44         H(1,1) = H(1,1) + invAx(:)'*invAx(:);
45         Lws = L' \ w;
46         for i=2:p+1
47             H(i,1) = H(i,1) - Lws(:,i-1)'*Lws(:,i-1);
48         end
49         H(1,2:p+1) = H(2:p+1,1)';
50         H(2:end,2:end) = H(2:end,2:end) + (w'*w).^2;
51
52         if nargin > 3
53             [L,err] = chol(H,'lower');
54             if err > 0
55                 in = false;  g = NaN;  H = NaN;  L = NaN;  return
56             end
57         end
58     end
59 end
60 return
    
```

3. Numerical Illustration: Design of Experiments

In this section, we illustrate the potential benefit of customizable barrier computation for a semidefinite representable problem using the example of *optimal design of experiments*, comparing the performance of *alfonso* to SCS 2.1.1 and Mosek 9.2.16. For the sake of brevity, we shall forego the detailed description of the statistical problem in order to focus on the formulation of the relevant convex optimization problem, which is stated as follows (Boyd and Vandenberghe 2004, section 7.5).

In the optimal design problem, the input data are a (usually dense) matrix $\mathbf{V} \in \mathbb{R}^{n \times p}$; we seek a vector $\mathbf{x} \in \mathbb{R}^p$ that solves the following optimization problem:

$$\begin{aligned} & \underset{\mathbf{x} \in \mathbb{R}^n}{\text{maximize}} && \Phi(\mathbf{V}\text{diag}(\mathbf{x})\mathbf{V}^T) \\ & \text{subject to} && \mathbf{1}^T \mathbf{x} = 1 \\ & && \mathbf{x} \geq 0 \end{aligned} \quad (2)$$

for some *optimality criterion* Φ that maps positive definite matrices to reals. (Implicit is the constraint that the argument of Φ is a positive definite matrix.) Most optimality criteria that are interesting from a statistical perspective are semidefinite representable in the sense of Ben-Tal and Nemirovski (2001), implying that these problems are solvable using semidefinite programming. For example, the choice of $\Phi(\mathbf{M}) = \lambda_{\min}(\mathbf{M})$ leads to an *E-optimal* design; see Boyd and Vandenberghe (2004, section 7.5.2) for a statistical interpretation.

Lower bound constraints on the smallest eigenvalue of a matrix \mathbf{M} can be cast in terms of a linear matrix inequality using the fact that

$$t \leq \lambda_{\min}(\mathbf{M}) \iff \mathbf{M} \succeq t\mathbf{I}_n.$$

Because in our application $\mathbf{M} = \mathbf{V}\text{diag}(\mathbf{x})\mathbf{V}^T$ is a linear function of our decision variables \mathbf{x} , Equation (2) can be readily translated to a semidefinite program with the help of an additional decision variable t and subsequently solved by any semidefinite programming solver. This is the formulation that we use with Mosek and SCS.

Instead of this semidefinite programming approach, *alfonso*, equipped with a custom barrier oracle implementation, can be used to solve Equation (2) directly as an optimization problem over the nonsymmetric cone

$$K_{\mathbf{V}} \stackrel{\text{def}}{=} \{(t, \mathbf{x}) \in \mathbb{R} \times \mathbb{R}_+^n \mid t \leq \lambda_{\min}(\mathbf{V}\text{diag}(\mathbf{x})\mathbf{V}^T)\}.$$

Figure 2 shows our implementation of the n -LHSCB for this cone inherited from the semidefinite formulation. This example is also included with *alfonso* in the file `e_design.m`; it has been slightly reformatted here to fit the page.

Table 1. Solver Statistics (Number of Iterations and Total Solver Time in Seconds) from Alfonso, Mosek 9, and SCS 2 Solving the E-Optimal Design Problem (2)

n	Alfonso		Mosek		SCS	
	Iter	Time	Iter	Time	Iter	Time
50	48	0.46	10	0.52	3,080	2.30
100	55	1.37	11	2.80	9,340	45.23
150	49	1.51	11	12.24	18,800	270.51
200	46	2.15	11	33.71	20,540	640.21
250	51	5.36	13	92.88	40,320	2,387.81
300	44	4.41	10	155.84		>1 hr
350	50	9.25	11	304.98		
400	46	7.94	11	521.45		
450	57	20.70	12	908.56		
500	51	12.85	12	1,420.50		

Notes. Alfonso and Mosek returned solutions with tolerance $\varepsilon = 10^{-8}$; the accuracy of the SCS solutions is $\varepsilon = 10^{-3}$. Missing values indicate that the solver exceeded one hour. Iter, iteration.

Table 1 shows the numerical results from a set of synthetic instances of (2) with $p = 2n$ and $n \in \{50, 200, \dots, 500\}$, using randomly generated matrices \mathbf{V} . Mosek and SCS were interfaced via Matlab. All computational results were obtained on a standard desktop computer equipped with 32GB RAM and a 4-GHz Intel Core i7 processor with four cores running using Matlab R2017b for Windows 10. Alfonso's optimality tolerance was reduced to $\varepsilon = 10^{-8}$ from the default 10^{-6} to match the accuracy of Mosek's solutions. Mosek and SCS were run using their default options except for increasing the maximum number of iterations for SCS to avoid early termination, tacitly acknowledging that as a first-order method, SCS is designed and expected to yield solutions with substantially lower accuracy than the interior-point methods. The solutions returned by SCS with its default tolerance settings correspond to $\varepsilon \approx 10^{-3}$ in our stopping criterion. The complete code of this example can be found in `e_design.m`.

In spite of returning lower-accuracy solutions, SCS exceeded one hour in the solution of the larger problems. Alfonso was significantly faster than both Mosek and SCS.

4. Discussion

Alfonso provides an easily usable and customizable, yet efficient, open-source tool for conic optimization. Using its oracle interface, researchers and practitioners can solve optimization problems over nonsymmetric cones that do not have a convenient representation in terms of symmetric cone constraints. Additionally, as our last example shows, it can even provide a significant speed-up over state-of-the-art solvers in problems with a straightforward, semidefinite programming formulation by exploiting the problem's structure and avoiding the introduction of a large number of auxiliary variables. A key feature of the underlying algorithm is that

all of its parameters are generic, applicable to any convex cone. Therefore, the user only needs to provide the code to compute the derivatives of the barrier function and a point in the interior of the cone.

4.1. Extending the Simple Interface

The simple interface currently supports a limited number of nonsymmetric cones (mostly the same ones as SCS and Mosek). New cones can be easily added with minimal changes to the code, limited to a single file `alfonso_simple.m`. Specifically, once the membership and barrier function oracle is prepared (as a separate Matlab file), the simple interface only needs a pointer to the cone and an interior point, both added in the form of a new line in a `switch-case` structure.

Endnote

¹ As of October 2020, Hypatia has been released.

References

- Andersen ED (2002) Handling free variables in primal-dual interior-point methods using a quadratic cone. *Proc. SIAM Conf. Optim.* (SIAM, Philadelphia).
- Ben-Tal A, Nemirovski A (2001) *Lectures on Modern Convex Optimization* (SIAM, Philadelphia).
- Blekherman G, Parrilo PA, Thomas RR, eds. (2013) *Semidefinite Optimization and Convex Algebraic Geometry* (SIAM, Philadelphia).
- Boyd SP, Vandenberghe L (2004) *Convex Optimization* (Cambridge University Press, Cambridge, UK).
- Chares R (2009) Cones and interior-point algorithms for structured convex optimization involving powers and exponentials. Unpublished PhD thesis, Université Catholique de Louvain, Louvain-la-Neuve, Belgium.
- Coeys C, Kapelevich L, Vielma JP (2020) Toward practical generic conic optimization. Preprint, submitted May 3, <https://arxiv.org/abs/2005.01136>.
- Domahidi A, Chu E, Boyd S (2013) ECOS: An SOCP solver for embedded systems. *Proc. Eur. Control Conf. (ECC)* (IEEE, Piscataway, NJ), 3071–3076.
- Glineur F, Terlaky T (2004) Conic formulation for l_p -norm optimization. *J. Optim. Theory Appl.* 122(2):285–307.
- Güler O (1997) Hyperbolic polynomials and interior point methods for convex programming. *Math. Oper. Res.* 22(2):350–377.
- Iliman S, de Wolff T (2016) Amoebas, nonnegative polynomials and sums of squares supported on circuits. *Res. Math. Sci.* 3(1):1–35.
- Karimi M, Tunçel L (2019) Domain-driven solver (DDS): A MATLAB-based software package for convex optimization problems in domain-driven form. Preprint, submitted August 7, <https://arxiv.org/abs/1908.03075>.
- Mosek ApS (2019) Mosek Optimization Suite release 9.0.105. Accessed April 1 2021, <https://docs.mosek.com/9.0/releasenotes/index.html>.
- Nesterov Y, Nemirovskii A (1994) *Interior-Point Polynomial Algorithms in Convex Programming*, SIAM Studies in Applied Mathematics, vol. 13 (Society for Industrial and Applied Mathematics, Philadelphia).
- O’Donoghue B, Chu E, Parikh N, Boyd S (2016) Operator splitting for conic optimization via homogeneous self-dual embedding. *J. Optim. Theory Appl.* 169:1042–1068.
- Papp D (2019) Duality of sum of nonnegative circuit polynomials and optimal SONC bounds. Preprint, submitted December 10, <https://arxiv.org/abs/1912.04718>.
- Papp D, Yildiz S (2017) On “A homogeneous interior-point algorithm for non-symmetric convex conic optimization.” Preprint, submitted December 1, <https://arxiv.org/abs/1712.00492>.
- Papp D, Yildiz S (2019) Sum-of-squares optimization without semidefinite programming. *SIAM J. Optim.* 29(1):822–851.
- Renegar J (2001) *A Mathematical View of Interior-Point Methods in Convex Optimization*, MOS-SIAM Series on Optimization (Society for Industrial and Applied Mathematics, Philadelphia).
- Renegar J (2004) Hyperbolic programs, and their derivative relaxations. Technical report, Cornell University Operations Research and Industrial Engineering, Ithaca, NY.
- Roy S, Xiao L (2018) On self-concordant barriers for generalized power cones. Technical Report MSR-TR-2018-3, Microsoft Research, Redmond, WA.
- Skajaa A, Ye Y (2015) A homogeneous interior-point algorithm for nonsymmetric convex conic optimization. *Math. Programming* 150(2):391–422.