A New Architecture for Optimization Modeling Frameworks

Matt Wytock Steven Diamond Felix Heide Stephen Boyd

***Abstract*—We propose a new architecture for optimization modeling frameworks in which solvers are expressed as computation graphs in a framework like TensorFlow rather than as standalone programs built on a low-level linear algebra interface. Our new architecture makes it easy for modeling frameworks to support high performance computational platforms like GPUs and distributed clusters, as well as to generate solvers specialized to individual problems. Our approach is particularly well adapted to first-order and indirect optimization algorithms. We introduce `cvxflow`, an open-source convex optimization modeling framework in Python based on the ideas in this paper, and show that it outperforms the state of the art.**

I. Introduction

Optimization offers a principled approach to solving problems in a wide variety of application domains, such as machine learning, statistics, control, signal and image processing, networking, engineering design, finance, and many others [BV04]. Instead of designing specialized algorithms for each individual problem, the user describes the problem as the minimization of a cost function and the optimal solution with minimal cost is found by the optimization method.

The wealth of applications for optimization methodology has driven the development of several high-level modeling languages. These languages provide a separation of concerns between the development of mathematical models and the implementation of numerical routines to optimize these models. This is especially useful for rapidly prototyping new applications, allowing practitioners to easily experiment with different objectives and constraints by writing expressions that closely mimic the mathematical optimization problem. Prominent examples of modeling languages and frameworks include AMPL [FGK02], YALMIP [Lof04], and CVX [GB14], as well as several tied closely to particular solvers, such as CPLEX's ILOG [Cpl07] and MathProg from GLPK [Mak00].

Despite the popularity of these modeling frameworks, support for modern large-scale computational environments such as GPUs and distributed clusters is virtually nonexistent. In part, this is due to fundamental challenges in scaling interior point methods, which have historically been the basis for solvers, as these methods require solving sparse linear systems to high accuracy and as such do not benefit greatly from GPU implementation. In addition, scaling such methods beyond a single machine typically requires high bandwidth interconnects such as those available exclusively in HPC environments.

However, there are also highly practical reasons for the lack of support for new environments: mature solvers often require several years to develop and writing entirely new implementations of low-level numerical routines specialized to each environment is unappealing. Traditionally, a degree of platform independence has been provided by implementing on top of low-level linear algebra libraries (BLAS and LAPACK) but as we discuss in this paper this architecture is often insufficient, especially for large problems. In addition, such libraries do not handle memory management and data transfer between GPU and CPU or between multiple machines.

The solution that we explore is a new architecture for optimization modeling frameworks based on solvers represented as computation graphs. This architecture is well-suited for solving large optimization problems by taking advantage of problem-specific structure. In particular, the computation graph abstraction naturally represents the composition of highly efficient linear operators which can be significantly faster than standard sparse or dense matrix representation. We develop such a method in this paper and demonstrate that it outperforms the existing state of the art for solving large convex optimization problems, a GPU-enabled version of SCS [OCPB16], which itself is one of the few GPU-optimized solvers available, POGS [FB15] being another example. A secondary, but not insignificant, benefit of this approach is automatic support for a wide variety of computational environments (CPU, GPU, distributed clusters), leveraging the considerable momentum and engineering effort of existing computation graph frameworks from the deep learning community, in particular TensorFlow [AAB+15].

The outline of the paper is as follows. In §II, we review the traditional architecture for optimization modeling frameworks and discuss its shortcomings. In §III, we describe the new architecture we propose and the computation graph abstraction the architecture is based on. In §IV, we compare our approach with alternative approaches discussed in prior work. In §V, we present `cvxflow`, an open-source implementation of the ideas in this paper, and numerical results comparing `cvxflow` with the state of the art.

II. Traditional architecture

The traditional architecture for optimization modeling frameworks dates back to AMPL [FGK02] and GAMS [BKMR88] in the 1980s. In this architecture, the framework is divided into three components: a modeling language, a solver implementation, and a low-level linear algebra interface. The modeling language provides a high-level syntax for describing optimization problems, which is compiled into a low-level representation called the standard form. The solver implementation takes the description of the optimization problem in the standard form and runs a numerical algorithm to compute the solution. The low-level linear algebra library provides optimized implementations of the core routines needed by the solver, such as sparse matrix factorization and matrix-matrix multiplication. The key feature of this architecture is a rigid and well-defined standard form which allows a given modeling language to interface with multiple solver implementations and vice versa.

A. Convex optimization modeling frameworks

The ecosystem of modeling frameworks for convex optimization is an illustrative example of the traditional architecture. Convex optimization modeling languages are built around the principles of disciplined convex programming, a set of rules for constructing optimization problems that make it easy to verify problem convexity. Implementations include CVX [GB14] and YALMIP [Lof04] in MATLAB, CVXPY [DB16c] in Python, Convex.jl [UMZ+14] in Julia, and the standalone compilers CVXGEN [MB12] and QCML [CPDB13].

The standard form for convex optimization problems is a cone program, an optimization problem of the form

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax + b \in \mathcal{K}, \end{array} \tag{1}$$

where $x \in \mathbf{R}^n$ is the optimization variable; $A \in \mathbf{R}^{m \times n}$, $b \in \mathbf{R}^m$, and $c \in \mathbf{R}^n$ are problem data; and $\mathcal{K}$ is a nonempty closed convex cone [NN92]. Computationally, c and b are represented by arrays and A is represented by a standard sparse matrix format, such as column compressed storage (CCS).

A wide variety of solver implementations have been developed for problems in the cone program standard form. Many solvers are written in pure C, including MOSEK [mos15], SDPA [FFK+08], ECOS [DCB13], and SCS [OCPB16]. Other solvers are written in higher level languages, such as SeDuMi [Stu99] and SDPT3 [TTT99] in MATLAB and CVXOPT [ADV15] in Python. The solvers rely heavily on low-level linear algebra interfaces like BLAS and LAPACK [LHKK79]. Existing cone solvers are almost exclusively restricted to CPU implementations; an exception is SCS which provides GPU support using the cuBLAS library [Nvi08].

B. Drawbacks

The traditional approach to optimization modeling frameworks has been enormously successful, allowing modeling languages and solver implementations to be developed independently in the programming languages best suited to their function. The conventional solver implementation is based on interior point methods, for which the dominant computational effort is solving a sparse linear system. Such a solver can be ported relatively easily to new platforms provided the necessary linear algebra libraries (*i.e.*, BLAS and LAPACK) are available.

However, many optimization problems of interest are too large to be solved with interior point methods and, more generally, any method that requires a direct solution to a linear system involving the A matrix of the cone program standard form (1). A possible solution is a first-order method, such as SCS, which only requires solving linear systems to moderate accuracy and thus can use indirect methods for the linear solver subroutine. As indirect methods require only matrix-vector computations with A and A^T, they can benefit tremendously from taking advantage of problem-specific structure in the form of specialized linear operator implementations, see *e.g.*, [DB16a]. However, this cannot be exploited by the traditional architecture which represents A with a sparse matrix, a fundamental necessity to take advantage of the low-level linear algebra interface.

Another line of work on solvers based on functions with efficient proximal operators also highlights the limitations of the traditional architecture. The standard forms used by Epsilon [WWK15] and ProxImaL [HDN+16] generalize the cone program, allowing nonlinear functions and enabling more problem structure to be retained by the solver. To support such standard forms, we require a more flexible intermediate representation

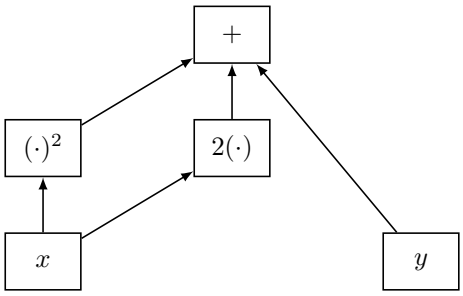

Fig. 1: A computation graph for $f(x, y) = x^2 + 2x + y$.

supporting a greater set of primitives beyond cones and linear functions. However, it is also desirable for such an intermediate representation to retain the platform independence that is provided by the the low-level linear algebra interface in the traditional architecture.

III. A NEW ARCHITECTURE

In this section we propose a new architecture for optimization modeling frameworks that overcomes the limitations of the traditional approach. The key abstraction in our architecture is a computation graph. We first explain computation graphs and then describe our architecture.

A. Computation graphs

A computation graph is a directed acyclic graph (DAG) where each vertex represents a mathematical operation and each edge represents data transfer. Input vertices have no incoming edges, while output vertices have no outgoing edges. A vertex is evaluated by applying its operation to the data on the vertex and broadcasting the result on its outgoing edges. The overall graph is evaluated by loading data onto the input vertices, evaluating the vertices in topological order, and reading the results off the output vertices.

For example, Fig. 1 shows a computation graph for the function $f(x, y) = x^2 + 2x + y$. The input vertices represent the variables x and y. The output vertex represent the top level sum. The internal vertices represent the operations $z \to z^2$ and $z \to 2z$.

Given a computation graph to evaluate a function, computation graphs for evaluating the function's gradient or adjoint (for linear functions) can be obtained through simple graph transformations [Gri89], [DB16a]. Function, gradient, and adjoint evaluations are the key operations in first-order and indirect solvers and are even sufficient to precondition a problem [DB16b].

Computation graphs are a useful intermediate representation for solvers because they abstract away the platform-specific details of both computation and memory management. These details are handled by a computation graph runtime system, which has platform-specific code to execute each mathematical operation and to pass data from one operation to the next. By contrast, a solver built on the traditional abstraction of a low-level linear algebra interface must implement its own platform-specific logic for mathematical operations not expressible as linear functions and for memory management.

B. Our approach

We propose a new graph-based architecture for optimization modeling frameworks. Our architecture divides the framework into three components: the modeling language, the solver generator, and the computation graph runtime system. The modeling language parses the high-level description of the optimization problem and converts it into a standard form. The solver generator takes a problem expressed in standard form and constructs a computation graph describing a numerical algorithm for solving that problem. The computation graph runtime system executes the computation graph on the target computational platform.

Our architecture addresses the limitations of the traditional architecture by making the interface between modeling language and solver more flexible. The modeling language and solver generator can both be implemented in high-level languages or even combined into a single library. The flexible interface allows us to modify the solver standard form to incorporate additional problem-specific structure. We also fully sever the link between solver implementation and computing platform. All that is needed to run a solver on a new platform is to change the target of the computation graph runtime system.

The drawback of our architecture is that we require a computation graph runtime system that supports the desired mathematical operations and platforms. For first-order and indirect solvers, the many frameworks developed for deep learning, such as TensorFlow [AAB+15], Theano [BBB+10], [BLP+12], Caffe [JSD+14], and Torch [CKF11], provide all the necessary functionality. The frameworks have only limited support, however, for the sparse matrix factorization routines used by direct solvers. Thus, given the available computation graph implementations, our architecture tends to favor first-order and indirect methods as opposed to interior point methods.

IV. Alternative approaches

Prior work has explored alternative approaches to bypassing the limitations of the traditional architecture for optimization modeling frameworks. Many of these approaches capture elements of our proposed architecture, but none offer the same degree of platform independence and flexibility in specifying the standard form.

Gondzio proposes a modeling framework for quadratic programs, a type of convex optimization problem, that exposes block matrix structure in the optimization problem. The block matrix structure is exploited to parallelize an interior point solver [GG07].

TFOCS is a MATLAB toolbox for solving convex optimization problems using first-order methods [BCG11]. Linear operators in the optimization problem can be expressed as computation graphs using the SPOT toolbox and passed to the TFOCS solvers [HHS+14].

Matrix-free CVXPY modifies the cone program standard form (1) to support computation graph representations of the matrix A. The modified standard form allows the modeling layer to pass information about structured linear operators in the optimization problem that solvers can exploit [DB16a].

The Epsilon solver [WWK15] introduces the standard form

$$\text{minimize} \quad \sum_{i=1}^{N} f_i(A_i x), \tag{2}$$

where $x \in \mathbf{R}^n$ is the optimization variable, $A_i \in \mathbf{R}^{m_i \times n}$ are linear operators, and f_i are functions with efficient proximal operators [PB14]. Epsilon exploits the flexibility of the standard form (2) to rewrite the problem so it can be solved efficiently by a variant of the alternating direction method of multipliers (ADMM) [BPC+11].

The ProxImaL modeling framework also targets the standard form (2), but supports a variety of solver algorithms and applies problem rewritings specialized to optimization problems in imaging [HDN+16]. ProxImaL moves towards platform independence by generating solver implementations using Halide [RKBA+13]. Halide is a language and compiler that allows for platform independent abstraction of individual mathematical operations, but not of full algorithms composed of many operations inside control logic.

V. Numerical examples

In this section, we present numerical examples of solving convex optimization problems in our proposed architecture. As solving linear systems forms the basis for convex methods, we first present results for an indirect linear solver with various linear operators. Using this indirect linear solver as a subroutine, we then implement a version of SCS [OCPB16] in the computation graph framework and compare with the native version of SCS implemented in C. We present results for both CPU and GPU environments; all experiments are run on a 32-core Intel Xeon 2.2GHz processor and an nVidia Titan X GPU with 12GB of RAM.

Our implementation builds on CVXPY [DB16c], a convex optimization modeling framework in Python. Using this framework, convex optimization problems can be expressed with minimal code and are automatically converted into the standard conic form (1). As an example, the nonnegative deconvolution problem we consider in Section V-C is written as the following Python code.

```
x = Variable(n)
f = norm(conv(c, x) - b, 2)
prob = Problem(Minimize(f), [x >= 0])
```

Here `c` and `b` are previously-defined problem inputs and `n` is the size of the optimization variable. Our implementation differs from the existing CVXPY functionality in that instead of solving problems by constructing sparse matrices and calling numerical routines written in C, we build a computation graph, as described in Section III, and evaluate with TensorFlow. Ultimately, this implementation achieves faster running times than existing methods—for example, on the large nonnegative deconvolution example, our implementation takes roughly 1/10th the time of SCS running on GPU, the existing state-of-the-art method for solving large convex problems to moderate accuracy.

Concurrent with the publication of this paper, we are releasing the `cvxflow` Python library; it is available at

http://github.com/mwytock/cvxflow

and includes the code for all of the examples in this section. The implementation is general and solves any problem modeled with CVXPY using TensorFlow.

A. Linear systems and regularized least squares

We begin with solving linear systems using the conjugate gradient method (CG) [HS52]. CG is matrix-free which makes it a natural fit for linear systems represented as a graph, allowing for specialized implementations of each linear operator including those that are inefficient to represent as sparse matrices such as convolution, kronecker products, and others.

We consider the regularized least squares problem

$$\text{minimize} \quad (1/2)\|Ax - b\|_2^2 + \lambda \|x\|_2^2 \tag{3}$$

where $x \in \mathbf{R}^n$ is the optimization variable, the linear map $A : \mathbf{R}^n \to \mathbf{R}^m$ and vector $b \in \mathbf{R}^m$ are problem

	dense matrix	sparse matrix	convolution
variables n	3000	3000	3000
nonzeros in A	18000000	180000	4095000
spsolve			
solve time	255 secs	28 secs	41 secs
memory usage	2.2 GB	1.06 GB	1.5 GB
objective	5.97×10^{-1}	5.97×10^{-1}	7.68×10^{-1}
CG TensorFlow			
solve time, CPU	3.0 secs	0.9 secs	2.9 secs
solve time, GPU	2.0 secs	0.7 secs	1.0 secs
graph build time	0.4 secs	0.1 secs	0.1 secs
memory usage	1.8 GB	755 MB	946 MB
objective	5.97×10^{1}	5.97×10^{-1}	7.68×10^{-1}
CG iterations	49	49	71

TABLE I: Results for regularized least squares.

data, and $\lambda > 0$ is the regularization parameter. This problem has the solution

$$x^\star = (\lambda I + A^T A)^{-1} A^T b, \tag{4}$$

which can be computed by solving a linear system.

It is often the case that A takes the form of a sparse or dense matrix; for example, in a statistical problem each row of A may represent an observation of multiple variables weighted by x in order to predict the response variable. However, A can also be an abstract linear operator; for example, a convolution with a vector c, written as $Ax = c * x$. We present results for each of these examples: a sparse matrix, a dense matrix, and convolution. In the matrix examples, entries are sampled from $\mathcal{N}(0, 1)$ with 1% nonzero in the sparse case. For convolution, we apply the Gaussian kernel with standard deviation $n/10$. In all cases, the response variable is formed by $b = A\hat{x} + v$ where v has entries sampled from $\mathcal{N}(0, 0.01^2)$ and $\hat{x}$ from $\mathcal{N}(0, 1)$. The conjugate gradient method is run until the residual satisfies $\|(\lambda I + A^T A)x^k - A^T b\|_2 \leq \|A^T b\|_2 \times 10^{-8}$.

Table I shows the results for these experiments, demonstrating that conjugate gradient on TensorFlow is significantly faster than the baseline method, `scipy.sparse.spsolve`. This is a somewhat weak baseline as `spsolve` does not run on GPU and is not well-suited for dense matrices. Nonetheless, this comparison highlights the difference in architecture exploited by TensorFlow which can take advantage of dedicated implementations for the linear operators leading to significantly faster solve times.

B. Lasso

Next we solve a convex problems with computation graph implementation of SCS [OCPB16], a first-order method based on the alternating direction method of multipliers [BPC+11]. In essence, the algorithm iterates between projections onto a linear subspace and a convex cone; the former is done through solving a linear system via conjugate gradient as in the previous section and the latter with simple closed-form expressions. This method is appealing as it works well with approximate solutions to linear systems, such as those produced by CG.

	dense matrix	sparse matrix	convolution
variables n	6001	6001	6001
constraints m	12002	12002	12001
nonzeros in A	18012002	1812002	4107002
SCS native			
solve time, CPU	29 secs	3.4 secs	6.4 secs
solve time, GPU	27 secs	3.8 secs	7.6 secs
matrix build time	13 secs	1.4 secs	2.8 secs
memory usage	3.1 GB	663 MB	927 MB
objective	3.36×10^{1}	3.19×10^{1}	2.02×10^{0}
SCS iterations	40	40	60
avg. CG iterations	2.66	2.71	2.72
SCS TensorFlow			
solve time, CPU	23 secs	25 secs	24 secs
solve time, GPU	9.9 secs	7.1 secs	5.3 secs
graph build time	1.8 secs	2.0 secs	0.8 secs
memory usage	8.7 GB	4.6 GB	1.2 GB
objective	3.36×10^{1}	3.19×10^{1}	2.02×10^{0}
SCS iterations	60	40	180
avg. CG iterations	3.35	3.55	1.93

TABLE II: Results for lasso.

We consider the lasso problem

$$\text{minimize} \quad (1/2)\|Ax - b\|_2^2 + \lambda\|x\|_1, \tag{5}$$

where the regularization term $\|x\|_1$ replaces the $\|x\|_2^2$ in the regularized least squares problem from the previous section. This problem is convex but no longer has a closed-form solution.

To generate problem instances, we construct same example linear operators A as in the previous section. We set the regularization parameter to $\lambda = 0.1 \times \|A^T b\|_\infty$ where $\|A^T b\|_\infty$ is the smallest value of λ such that the solution is zero. Table II compares the TensorFlow version of SCS to the native implementation and demonstrates that in the dense matrix and convolution cases, the solve time on GPU is faster with TensorFlow. This highlights the benefit of the computation graph: specialized implementations for dense matrix multiplication and convolution as opposed to reducing all linear functions to sparse matrix operations. In contrast, when the input linear operator A is a sparse matrix, native SCS is faster, suggesting that sparse matrix operations in TensorFlow are not as highly optimized as the cuBLAS routines.

	small	**medium**	**large**
variables n	101	1001	10001
constraints m	300	3000	30000
nonzeros in A	9401	816001	69220001
SCS native			
solve time, CPU	0.1 secs	2.2 secs	260 secs
solve time, GPU	2.0 secs	2.0 secs	105 secs
matrix build time	0.01 secs	0.6 secs	52 secs
memory usage	360 MB	470 MB	10.4 GB
objective	1.38×10^0	4.57×10^0	1.41×10^1
SCS iterations	380	100	160
avg. CG iterations	8.44	2.95	3.01
SCS TensorFlow			
solve time, CPU	3.4 secs	5.7 secs	88 secs
solve time, GPU	5.7 secs	3.2 secs	13 secs
graph build time	0.8 secs	0.8 secs	0.9 secs
memory usage	895 MB	984 MB	1.3 GB
objective	1.38×10^0	4.57×10^0	1.41×10^1
SCS iterations	480	100	160
avg. CG iterations	2.75	2.00	2.00

TABLE III: Results for nonnegative deconvolution.

C. Nonnegative deconvolution

As a final example further illustrating the benefit of abstract linear operators, we consider the nonnegative deconvolution problem

$$\begin{array}{ll} \text{minimize} & \|c * x - b\|_2 \\ \text{subject to} & x \geq 0 \end{array} \tag{6}$$

where $x \in \mathbf{R}^n$ is the optimization variable, and $c \in \mathbf{R}^n$, $b \in \mathbf{R}^{2n-1}$ are problem data. We generate problem instances by taking c to be the Gaussian kernel with standard deviation $n/10$ and convolving it with a sparse signal $\hat{x}$ with 5 nonzero entries sampled uniformly from $[0, n/10]$. We set the response $b = c * \hat{x} + v$ with $v \sim \mathcal{N}(0, 0.01^2)$.

Table III shows that on large problem sizes, the SCS TensorFlow implementation performs significantly better than the native implementation, requiring 13 seconds as compared to 105 seconds. This difference is largely due to differences in architecture, as the matrix-based SCS requires a considerable amount of time (52 seconds) to simply construct the sparse matrix representing the convolution operator. As many linear operators benefit from from specialized implementations (see *e.g.*, [HHS+14], [BGFB94], [VB95], [DB15]), one could easily demonstrate an even more significant gap between the proposed architecture and existing methods simply by choosing more egregious examples that highlight this difference.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-114747 and by the DARPA XDATA program.

REFERENCES

[AAB+15] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. http://tensorflow.org/. Cited 2 March 2016.

[ADV15] M. Andersen, J. Dahl, and L. Vandenberghe. CVXOPT: Python software for convex optimization, version 1.1. http://cvxopt.org/, May 2015.

[BBB+10] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference*, June 2010.

[BCG11] S. Becker, E. Candès, and M. Grant. Templates for convex cone problems with applications to sparse signal recovery. *Mathematical Programming Computation*, 3(3):165–218, 2011.

[BGFB94] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. *Linear matrix inequalities in system and control theory*, volume 15. SIAM, 1994.

[BKMR88] A. Brooke, D. Kendrick, A. Meeraus, and R. Rosenthal. *GAMS: A user's guide*. Course Technology, 1988.

[BLP+12] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio. Theano: new features and speed improvements. In *Deep Learning and Unsupervised Feature Learning, Neural Information Processing Systems Workshop*, 2012.

[BPC+11] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[CKF11] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A MATLAB-like environment for machine learning. In *BigLearn, Neural Information Processing Systems Workshop*, 2011.

[CPDB13] E. Chu, N. Parikh, A. Domahidi, and S. Boyd. Code generation for embedded second-order cone programming. In *Proceedings of the European Control Conference*, pages 1547–1552, 2013.

[Cpl07] ILOG Cplex. 11.0 users manual. *ILOG SA, Gentilly, France*, page 32, 2007.

[DB15] Steven Diamond and Stephen Boyd. Convex optimization with abstract linear operators. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 675–683, 2015.

[DB16a] S. Diamond and S. Boyd. Matrix-free convex optimization modeling. In *Optimization and Applications in Control and Data Sciences*. Springer, 2016. To appear.

[DB16b] S. Diamond and S. Boyd. Stochastic matrix-free equilibration. *To Appear in Journal of Optimization Theory and Applications*, 2016.

[DB16c] Steven Diamond and Stephen Boyd. Cvxpy: A python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[DCB13] A. Domahidi, E. Chu, and S. Boyd. ECOS: An SOCP solver for embedded systems. In *Proceedings of the European Control Conference*, pages 3071–3076, 2013.

[FB15] Christopher Fougner and Stephen Boyd. Parameter selection and pre-conditioning for a graph form solver. *arXiv preprint arXiv:1503.08366*, 2015.

[FFK+08] K. Fujisawa, M. Fukuda, K. Kobayashi, M. Kojima, K. Nakata, M. Nakata, and M. Yamashita. SDPA (semidefinite programming algorithm) user's manual – version 7.0.5. Technical report, 2008.

[FGK02] R. Fourer, D. Gay, and B. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Cengage Learning, 2002.

[GB14] M. Grant and S. Boyd. CVX: MATLAB software for disciplined convex programming, version 2.1. http://cvxr.com/cvx, March 2014.

[GG07] J. Gondzio and A. Grothey. Parallel interior-point solver for structured quadratic programs: Application to financial planning problems. *Ann. Oper. Res.*, 152(1):319–339, 2007.

[Gri89] A. Griewank. On automatic differentiation. In M. Iri and K. Tanabe, editors, *Mathematical Programming: Recent Developments and Applications*, pages 83–108. Kluwer Academic, Tokyo, 1989.

[HDN+16] Felix Heide, Steven Diamond, Matthias Nießner, Jonathan Ragan-Kelley, Wolfgang Heidrich, and Gordon Wetzstein. Proximal: Efficient image optimization using proximal algorithms. *ACM Trans. Graph.*, 35(4):84:1–84:15, July 2016.

[HHS+14] G. Hennenfent, F. Herrmann, R. Saab, O. Yilmaz, and C. Pajean. SPOT: A linear operator toolbox, version 1.2. http://www.cs.ubc.ca/labs/scl/spot/index.html, March 2014.

[HS52] M. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. N.B.S.*, 49(6):409–436, 1952.

[JSD+14] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *Preprint*, 2014. http://arxiv.org/abs/1408.5093. Cited 2 March 2016.

[LHKK79] Chuck L Lawson, Richard J. Hanson, David R Kincaid, and Fred T. Krogh. Basic linear algebra subprograms for fortran usage. *ACM Transactions on Mathematical Software (TOMS)*, 5(3):308–323, 1979.

[Lof04] J. Lofberg. YALMIP: A toolbox for modeling and optimization in MATLAB. In *Proceedings of the IEEE International Symposium on Computed Aided Control Systems Design*, pages 294–289, 2004.

[Mak00] Andrew Makhorin. Modeling language gnu mathprog. *Relatório Técnico, Moscow Aviation Institute*, 2000.

[MB12] J. Mattingley and S. Boyd. CVXGEN: A code generator for embedded convex optimization. *Optimization and Engineering*, 13(1):1–27, 2012.

[mos15] MOSEK optimization software, version 7. https://mosek.com/, January 2015.

[NN92] Y. Nesterov and A. Nemirovsky. Conic formulation of a convex programming problem and duality. *Optimization Methods and Software*, 1(2):95–115, 1992.

[Nvi08] CUDA Nvidia. cuBLAS library. *NVIDIA Corporation, Santa Clara, California*, 15:27, 2008.

[OCPB16] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169(3):1042–1068, 2016.

[PB14] N. Parikh and S. Boyd. Proximal algorithms. *Found. Trends Optim.*, 1(3):123–231, 2014.

[RKBA+13] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, pages 519–530, New York, NY, USA, 2013. ACM.

[Stu99] J. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11(1-4):625–653, 1999.

[TTT99] K.-C. Toh, M. Todd, and R. Tütüncü. SDPT3 — a MATLAB software package for semidefinite programming, version 4.0. *Optimization Methods and Software*, 11:545–581, 1999.

[UMZ+14] M. Udell, K. Mohan, D. Zeng, J. Hong, S. Diamond, and S. Boyd. Convex optimization in Julia. In *Proceedings of the Workshop for High Performance Technical Computing in Dynamic Languages*, pages 18–28, 2014.

[VB95] L. Vandenberghe and S. Boyd. A primaldual potential reduction method for problems involving matrix inequalities. *Mathematical Programming*, 69(1-3):205–236, 1995.

[WWK15] Matt Wytock, Po-Wei Wang, and J Zico Kolter. Convex programming with fast proximal and linear operators. *arXiv preprint arXiv:1511.04815*, 2015.