Structure-aware Linear Solver for Realtime Convex Optimization for Embedded Systems

Ichitaro Yamazaki, Saeid Nooshabadi, *Senior Member, IEEE*, Stanimire Tomov, and Jack Dongarra.

***Abstract*— With the increasing sophistication in the use of optimization algorithms such as deep learning on embedded systems, the convex optimization solvers on embedded systems have found widespread use. This letter presents a novel linear solver technique to reduce the run-time of convex optimization solver by using the property that some parameters are fixed during the solution iterations of a solve instance. Our experimental results show that the run-time can be reduced by two orders of magnitude.**

***Index Terms*—Realtime embedded convex optimization solver, KKT**

I. Introduction

Recent advances in convex optimization [1] [2] have enabled their use as realtime solvers for embedded systems [3] [4]. Embedded solvers come with certain features that can be exploited to reduce the complexity of the design. For instance, in many cases, the embedded solvers only require a low accuracy. One such example is the model predictive control (MPC), where very low accuracy is needed to obtain acceptable control performance [5]. Another feature is that, for an embedded solver, the structure of the problem does not change from one solve to the next. For example, for a given problem instance of Kalman filtering, an embedded solver perform many solves, where the dimensions and structure of the system state, input and output vectors, and steady-state error covariance matrix are all fixed, and the system parameters remain unchanged with each realtime solution iteration. Furthermore, the change in the numerical values of the solver parameters between two subsequent instances of the problem is usually small. The fixed size and structure of the system provides an opportunity to design and optimize the solver at the development time and to significantly reduce the solve time.

To obtain high performance on the embedded system, recent tools such as CVXGEN code generator [6] and ECOS [7] provide frameworks for realtime convex optimization solvers. There are also some attempts in code generation for small-sized basic linear algebra opeations like vector-matrix multiplication [8]. CVXGEN takes a high-level description of the optimization problem, employs the CVX technique of disciplined convex programming (DCP) [9] to ensure convexity, and then generates a flat, and library-free C code. The generated code can be compiled into a high performance solver for the specific problem family (*e.g.*, all the matrices have the same sparsity structure). Though CVXGEN solver obtains a high performance that meets the strict realtime constraint enforced on the solution time, the size of the problem is limited to 100 or so variables for the code generator to successfully generate a flat loop-free C code.

In DCP, a quadratic programming (QP) convex problem is transformed into a standard form that seeks for the vector variable v that minimizes the following optimization problem [10],

$$\begin{array}{ll} \text{minimize} & (1/2)v^{\mathrm{T}}Qv + q^{\mathrm{T}}v \\ \text{subject to} & Gv \preceq h \ \text{ and } \ Av = b, \end{array} \tag{1}$$

where $v, q \in \mathbf{R}^n$, $Q \in \mathbf{S}^n_+ \succeq 0$ (a square symmetric positive semidefinite matrix), $A \in \mathbf{R}^{m\times n}$, $G \in \mathbf{R}^{p\times n}$, $b \in \mathbf{R}^m$, and $h \in \mathbf{R}^p$. At each iteration of the solve instance, most of the time is spent solving a family of Karush Kuhn Tucker (KKT) [2] linear system of equations $\widehat{K}\widehat{x} = \widehat{c}$, whose coefficient matrices $\widehat{K}$ all have the following block structure:

$$\widehat{K} = \left(\begin{array}{cc|cc} Q & 0 & G^{\mathrm{T}} & A^{\mathrm{T}} \\ 0 & S^{-1}Z & I_p & 0 \\ \hline G & I_p & 0 & 0 \\ A & 0 & 0 & 0 \end{array}\right), \tag{2}$$

where $(.)^{\mathrm{T}}$ denote the matrix transpose, $I_p \in \mathbf{R}^{p\times p}$ is the identity matrix, $S = \mathrm{diag}(s) \in \mathbf{R}^{p\times p}$ and $Z = \mathrm{diag}(z) \in \mathbf{R}^{p\times p}$ are diagonal matrices, and $s \in \mathbf{R}^p$ and $z \in \mathbf{R}^p$ are vectors representing the slack variables and inequality multipliers, respectively, in the KKT condition [11]. The optimization solver performs several iterations until a pre-determined accuracy or the maximum number of iterations is reached.

To solve the linear system, CVXGEN first computes the $\mathrm{LDL}^{\mathrm{T}}$ decomposition of the KKT matrix $\widehat{K}$, i.e., $P\widehat{K}P^{\mathrm{T}} = \widehat{L}\widehat{D}\widehat{L}^{\mathrm{T}}$ [12], where P is a permutation matrix, $\widehat{L}$ a lower triangular matrix with unit diagonals, and $\widehat{D}$ is a diagonal matrix. Then, the solution $\widehat{x}$ is computed through the sequence of forward substitution, scaling, and backward substitution.

The current embedded optimization solvers, while good at taking advantage of the structure of the problem family (*e.g.* sparsity), fails to take advantage of the fact that several blocks of the KKT matrix are fixed during the iterations of a given solve instance. In most practical signal processing applications for embedded systems, such as linearizing pre-equalizer, Kalman filtering, sliding window smoothing or estimation [13], only the vectors q, h or b in (1) change from one solve

Saeid Nooshabadi is with the Department of Electrical and Computer Engineering, Michigan Tech, Houghton, MI, e-mail:{saeid}@mtu.edu; Ichitaro Yamazaki, Stanimire Tomov and Jack Dongara are with Innovative Computing Laboratory (ICL), the University of Tennessee, Knoxville, e-mail{iyamazak, tomov, dongarra}@icl.utk.edu

Manuscript received January 2017; revised: April 2017

instance to the next. These vectors only affect the right-hand-side vector $\widehat{c}$ of the KKT linear system. In the online array weight design for adaptive array signal processing [3], only the matrix G changes between the solve instances. In many other problems of interests, only the diagonal block $S^{-1}Z$ changes at each solve instance. This letter proposes to reduce the time to solve the family of the KKT linear systems by exploiting the fact that several blocks in the KKT matrix $\widehat{K}$ are fixed during the solution iterations of a given problem instance.

II. Algorithms

We consider two types of changes in the KKT matrix for the interior-point methods [1] [2]: A) the change that occurs at each iteration of a solve instance and B) the change that persists across all the solution iterations in one solve instance. Hence, our linear solver consists of three phases: 1) off-line setup which is done once for all the solve instances of a problem family, 2) off-line update which is done once for each solve instance, and 3) on-line factorization and solve which are done once at each iteration. To avoid factorizing singular blocks, like CVXGEN, we regularize $\widehat{K}$ using the standard techique [11].

A. Fixed Q, A, G and variable $S^{-1}Z$

Between the solution iterations, it is often that only the diagonal block $S^{-1}Z$ changes. We take advantage of this property and amortize the cost of the operations that involve Q, A and G over all the solve instances. For this, we solve an equivalent, *implicitly-reordered*, linear system, $Kx = b$, *i.e.*,

$$\left(\begin{array}{c|c|c|c} Q & A^{\mathrm{T}} & 0 & G^{\mathrm{T}} \\ \hline A & 0 & 0 & 0 \\ \hline 0 & 0 & S^{-1}Z & I_p \\ \hline G & 0 & I_p & 0 \end{array}\right)\left(\begin{array}{c} \widehat{x}_1 \\ \hline \widehat{x}_4 \\ \hline \widehat{x}_2 \\ \hline \widehat{x}_3 \end{array}\right)=\left(\begin{array}{c} \widehat{c}_1 \\ \hline \widehat{c}_4 \\ \hline \widehat{c}_2 \\ \hline \widehat{c}_3 \end{array}\right). \tag{3}$$

We emphasize that the matrix is not explicitly reordered, but we factorize and solve the linear system in the order given in (3) as follows:

a) Initial off-line setup: As part of our off-line setup, we partially factorize the matrix such that $K = LDL^{\mathrm{T}}$, where

$$L = \left(\begin{array}{c|c|c|c} L_{1,1} & 0 & 0 & 0 \\ \hline L_{2,1} & L_{2,2} & 0 & 0 \\ \hline 0 & 0 & I_p & 0 \\ \hline L_{4,1} & L_{4,2} & 0 & I_p \end{array}\right),\quad D = \left(\begin{array}{c|c|c|c} D_{11} & 0 & 0 & 0 \\ \hline 0 & D_{22} & 0 & 0 \\ \hline 0 & 0 & S^{-1}Z & I_p \\ \hline 0 & 0 & I_p & C \end{array}\right) \tag{4}$$

For this partial factorization, the last 2-by-2 trailing blocks of L and D are not yet fully factorized, but will be factorized in the on-line factorization phase. Therefore, neither L nor D are yet in its respective full lower triangular nor diagonal form. We compute this partial LDL$^{\mathrm{T}}$ factorization as follows:

1) We compute the LDL$^{\mathrm{T}}$ factorization of Q such that $Q = L_{1,1}D_{1,1}L_{1,1}^{\mathrm{T}}$.
2) We compute the off-diagonal blocks $L_{2,1}$ and $L_{4,1}$ in the first block column of L such that $L_{2,1} := A(D_{1,1}L_{1,1}^{\mathrm{T}})^{-1}$ and $L_{4,1} := G(D_{1,1}L_{1,1}^{\mathrm{T}})^{-1}$.
3) We compute the LDL$^{\mathrm{T}}$ factorization of the second diagonal block $\widetilde{K}_{2,2}$ such that $\widetilde{K}_{2,2} = L_{2,2}D_{2,2}L_{2,2}^{\mathrm{T}}$, where $\widetilde{K}_{2,2} := -(L_{2,1}D_{1,1}L_{2,1}^{\mathrm{T}})$.

   We put $\widetilde{\ }$ on top of the block to distinguish it from the corresponding block of the original matrix K.
4) We compute the off-diagonal block $L_{4,2}$ in the second block column of L such that $L_{4,2} := \widetilde{K}_{4,2}(D_{2,2}L_{2,2}^{\mathrm{T}})^{-1}$, where $\widetilde{K}_{4,2} := -(L_{4,1}D_{1,1}L_{2,1}^{\mathrm{T}})$
5) We compute the last diagonal block C of D such that $C := -(L_{4,1}D_{1,1}L_{4,1}^{\mathrm{T}}) - (L_{4,2}D_{2,2}L_{4,2}^{\mathrm{T}})$.

It is not shown to simplify the notations, but the LDL$^{\mathrm{T}}$ factorization of each diagonal block is computed with permutation.

b) On-line Factorization: At each iteration of the solve instance with a new $S^{-1}Z$, we can factorize the last two diagonal blocks of D in (4), *i.e.*,

$$\left(\begin{array}{c|c} S^{-1}Z & I_p \\ \hline I_p & C \end{array}\right)=\left(\begin{array}{c|c} I_p & 0 \\ \hline L_{4,3} & L_{4,4} \end{array}\right)\left(\begin{array}{c|c} D_{3,3} & \\ \hline & D_{4,4} \end{array}\right)\left(\begin{array}{c|c} I_p & L_{4,3}^{\mathrm{T}} \\ \hline & L_{4,4}^{\mathrm{T}} \end{array}\right)$$

where

$$\left\{\begin{array}{lcl} D_{3,3} & := & S^{-1}Z \\ L_{4,3} & := & D_{3,3}^{-1} = Z^{-1}S, \end{array}\right.$$

and $\widetilde{C} = L_{4,4}D_{4,4}L_{4,4}^{\mathrm{T}}$ with $\widetilde{C} := C - (Z^{-1}S)$. Since both Z and S are diagonal, the most computationally expensive part of the on-line factorization is the LDL$^{\mathrm{T}}$ factorization of $\widetilde{C}$. In other words, our on-line factorization only factorizes a matrix of the dimension p, and this algorithm allows us to solve the realtime convex optimization where p, instead of $n + m + 2p$, is the largest dimension of the matrix that must be factorized within the realtime constraint.

In this letter, we only consider the QP problem, *i.e.*, diagonal $S^{-1}Z$. However, the algorithm can be trivially extended to a more general case such as the second order cone programming (SOCP) [2], where $S^{-1}Z$ is replaced with a symmetric positive definite scaling matrix computed from the primal and dual variables (such as $S^{-1/2}ZS^{-1/2}$).

c) On-line Solve: With the complete LDL^{T} factorization of K, the solution to the linear system can be computed through the forward and backward substitutions.

B. Fixed Q, A and variable G, $S^{-1}Z$

In some cases, the submatrix G changes between the solve instances, but stays fixed during the solution iterations of one solve instance. In such cases, we introduce an intermediate off-line update step, where we only recompute the steps that use G. The cost of this intermediate step is amortized over all the solution iterations of the solve instance.

a) Initial off-line setup:

1) We compute the LDL$^{\mathrm{T}}$ factorization of Q such that $Q = L_{1,1}D_{1,1}L_{1,1}^{\mathrm{T}}$.
2) We compute the off-diagonal block $L_{2,1}$ in the first block column of L such that $L_{2,1} := A(D_{1,1}L_{1,1}^{\mathrm{T}})^{-1}$.

TABLE I
NUMBER OF FLOATING POINT OPERATIONS NEEDED FOR FACTORIZATION.

Fixed Q, A, G		Fixed Q and A	
Off-line setup			
LDL^T factor Q	$n^3/3$		
Compute $L_{2,1}$	n^2m		
LDL^T factor $\widetilde{K}_{2,2}$	$m^2n + m^3/3$		
Compute $L_{4,1}$	n^2p		
Compute $L_{4,2}$	$2pmn + m^2p$	Compute H	m^2n
Compute C	$p^2(n+m)$		
Off-line update			
		Compute $L_{4,1}$	n^2p
		Compute $L_{4,2}$	$2pmn$
		Compute C	$p^2(n+m)$
On-line factorization			
LDL^T factor $\widetilde{C}$	$p^3/3$		

3) We factorize the second diagonal block $\widetilde{K}_{2,2}$ such that $\widetilde{K}_{2,2} = L_{2,2}D_{2,2}L_{2,2}^{\mathrm{T}}$, where $\widetilde{K}_{2,2} := -(L_{2,1}D_{1,1}L_{2,1})$
4) We partially compute the off-diagonal block in the second block column of L, *i.e.,* $H := -D_{1,1}L_{2,1}^{\mathrm{T}}(D_{2,2}L_{2,2})^{-1}$.

b) Off-line update: To solve each convex optimization problem with a new submatrix G, we complete the off-line factorization as follows:

1) We compute the off-diagonal block $L_{4,1}$ in the first block column of L such that $L_{4,1} := G(D_{1,1}L_{1,1}^{\mathrm{T}})^{-1}$.
2) We compute the off-diagonal block $L_{4,2}$ in the second block column of L such that $L_{4,2} := L_{4,1}H$.
3) We compute the last block C of D such that $C := -(L_{4,1}D_{1,1}L_{4,1}^{\mathrm{T}}) - (L_{4,2}D_{2,2}L_{4,2}^{\mathrm{T}})$.

c) On-line Factorization: Finally, at each iteration with the new diagonal submatrix $S^{-1}Z$, we can cheaply perform the on-line factorization and solve as in Section II-A.

To summarize our two structure-aware linear solvers, Table I lists the required number of floating point operations at each step of factorization (in non-complex precision).

III. IMPLEMENTATION

For our experiments, we focused on the dense blocks Q, A, and G, and our implementation is based on LAPACK [14]. For example, to factorize the diagonal blocks, we used LAPACK's `dsytrf` subroutine that dynamically permutes the matrix to ensure the numerical stability. In contrast, the CVXGEN's generated solver uses static permutation and regularization to avoid the small diagonal entries through small diagonal shifts [6]. Compared to the dynamic pivoting, the static pivoting may be less stable or accurate, but it can be more efficient and leads to the fixed factorization time.

Unfortunately, LAPACK does not provide a flexible enough interface to take the full advantage of the symmetry in the KKT matrix. In particular, LAPACK has the subroutine `dsytrs` to apply $(LDL^{\mathrm{T}})^{-1}$ using the LDL^T factorization computed by `dsytrf`, but it does not have a subroutine to apply L^{-1} or D^{-1} alone, which is needed, for example, at Step 2 of the off-line setup in Section II-A. Hence, we compute the non-symmetric LDU factorization of the KKT matrix, $K = LDU$, where L and D have the same block structures as those in (4), and U has the same structure as that of L^{T}. With this implementation, at Step 1 of the initial off-line setup in Section II-A, we let $L_{1,1} = I_n$, $D_{1,1} = I_n$, and $U_{1,1} = Q$, and then, we use `dsytrf` to compute the LDL$^{\mathrm{T}}$ factorization of Q. Then, at Step 2, we compute, $L_{2,1} := AU_{1,1}^{-1}$ and $L_{4,1} := GU_{1,1}^{-1}$, while $U_{1,2} = A^{\mathrm{T}}$ and $U_{1,4} = G^{\mathrm{T}}$. We use LAPACK's `dsytrs` to apply $U_{1,1}^{-1}$ based on the LDL$^{\mathrm{T}}$ factorization of Q.

Similary, at Steps 3, we set $L_{2,2} = I_m$, $D_{2,2} = I_m$, and $U_{2,2} = \widetilde{K}_{2,2}$, where $\widetilde{K}_{2,2} := -(L_{2,1}U_{2,1})$, and we factorize $\widetilde{K}_{2,2}$ using `dsytrf`. Then, at Step 4, we compute, $L_{4,2} := -(L_{4,1}U_{1,2})U_{2,2}^{-1}$ and $U_{2,4} := L_{2,1}U_{1,4}$. Finally, at Step 5, we compute $C := -(L_{4,1}U_{1,4}) - (L_{4,2}U_{2,4})$.

IV. EXPERIMENTS

We conducted all of our experiments on MacBook Pro®, using just one core of 2.7 GHz Intel® Core™ i7. Our codes were compiled using `gcc` of Apple® LLVM version 5.1 with the optimization flag `-Os`. For our experiments, we linked our codes to BLAS and LAPACK provided in the Apple's Accelerate Framework [15], but on any other embedded system, it could be statically linked to open-source reference implementations of BLAS [16] and LAPACK [14].

A. Results with fixed Q, A, G and variable $S^{-1}Z$

The performance results in Table II correspond to the largest convex optimization problem of the form (1), that CVXGEN could handle. The KKT matrix is of dimension 131, and the respective dimensions of the submatrices Q, Z, G, and A are 95×95, 12×12, 95×12, and 95×12.

For this particular dimension of the matrix, the CVXGEN's generated solver was slower than calling LAPACK's non-symmetric or symmetric solver, `dgesv` or `dsysv`, respectively, on the matrix $\widehat{K}$, even though LAPACK ignores the structure of the KKT matrix and performs dynamic pivoting.

Since LAPACK interface does not allow our off-line factorization to take full advantage of the symmetry of the KKT matrix, it was slower than the LAPACK's symmetric `dsysv` solver. However, it was faster than the LAPACK's non-symmetric `dgesv` solver or the CVXGEN's generated solver. More importantly, the cost of this initial off-line setup is amortized over all the solve instances, and by taking advantage of the fixed structure, our on-line factorization was significantly faster than the CVXGEN's factorization with speedups of about 131.3.

In Table II, we also show the results when we consider the submatrix Q to be diagonal. Since the submatrix Q does not have to be factorized, compared to the case of a dense Q, our off-line factorization obtained a greater speedup over the CVXGEN's factorization.

TABLE II
RUN TIME (IN SECONDS) FOR SOLVING KKT LINEAR SYSTEM.

Technique	Factor	Solve
CVXGEN	$5.25 \cdot 10^{-4}$	$1.70 \cdot 10^{-5}$
LAPACK (`dgesv`)	$3.03 \cdot 10^{-4}$	$2.20 \cdot 10^{-5}$
LAPACK (`dsysv`)	$1.78 \cdot 10^{-4}$	$1.90 \cdot 10^{-5}$
fixed Q,A,G (dense Q):		
initial off-line setup	$1.85 \cdot 10^{-4}$	
on-line factor and solve	$4.00 \cdot 10^{-6}$	$2.30 \cdot 10^{-5}$
fixed Q,A,G (diagonal Q):		
initial off-line setup	$3.90 \cdot 10^{-5}$	
on-line factor and solve	$4.00 \cdot 10^{-6}$	$6.00 \cdot 10^{-6}$
fixed Q,A (dense Q):		
initial off-line setup	$1.21 \cdot 10^{-4}$	
off-line update	$7.00 \cdot 10^{-5}$	
on-line factor and solve	$4.00 \cdot 10^{-6}$	$2.30 \cdot 10^{-6}$

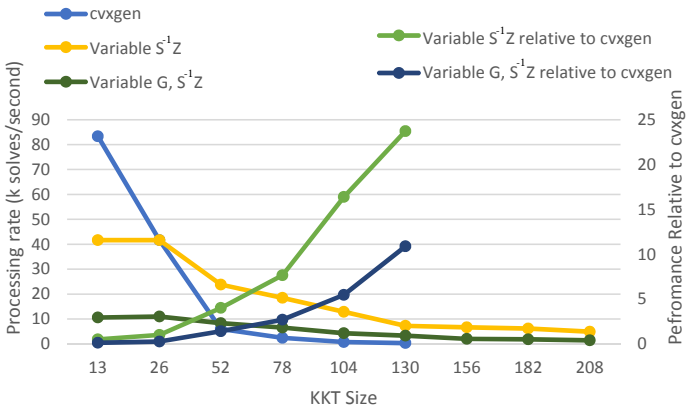

Fig. 1. Processing rate and relative performance with respect to CVXGEN

B. Results with fixed Q, A and variable G, $S^{-1}Z$

The last few rows of Table II show the performance of our solver when only the submatrix G changes between the instances of the solve. Even with the extra operations required for the non-symmetric factorization, by taking advantage of the structure, our on-line update was faster than the CVXGEN's factorization by a factor of about 7.5.

C. Scaling with the problem size

We compare the performance of our solver with that of CVXGEN, using a set of test problems of varying sizes; the KKT matrix of dimension from 13 ($n = 10$, $m = 1$, and $p = 1$) to 208 ($n = 160$, $m = 16$, and $p = 16$). Figure 1 plots the processing rates (the number of solves per seconds) and the relative performance of various schemes. For the KKT matrix with its dimension greater than 52 ($n = 40$, $m = 4$, and $p = 4$), both of our solvers performed better than CVXGEN's. For problems with the matrix size larger than 130 ($n = 100$, $m = 10$, and $p = 10$), CVXGEN could not generate the necessary code.

Figure 1 also shows that the relative performance of our two solvers, compared with CVXGEN, grows with the problem size, reaching to about 24 and 12 for the matrix size of 130 ($n = 100$, $m = 10$, and $p = 10$).

V. CONCLUSION

In this letter, we proposed techniques to reduce the time to factorize the KKT matrices for solving a realtime convex optimization on an embedded system. This technique takes advantage of the fact that many blocks of the KKT matrix do not change during the iterations of one solve instance. Our experimental results have shown that compared to the CVXGEN's generated solver, the factorization time can be reduced by two orders of magnitude.

REFERENCES

[1] D. P. Bertsekas, *Convex Optimization Algorithms*. Athena Scientific, Belmont, MA., 2015.

[2] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge University Press, Cambridge, 2012.

[3] J. Mattingley and S. Boyd, "Realtime convex optimization in signal processing," *IEEE Signal Processing Magazine*, vol. 27, no. 3, pp. 50–61, May 2010.

[4] D. Burns, W. Weiss, and M. Guay, "Realtime setpoint optimization with time-varying extremum seeking for vapor compression systems," in *American Control Conference (ACC), 2015*, July 2015, pp. 974–979.

[5] Y. Wang and S. P. Boyd, "Fast model predictive control using online optimization," in *Proceedings World Congress of the International Federation of Automatic Control (IFAC)*, Jejudo Island, Korea, June 2001, pp. 6974–6997.

[6] J. Mattingley and S. Boyd, "CVXGEN – code generation for convex optimization," *Optimization and Engineering*, vol. 13, no. 1, pp. 1–27, March 2012. [Online]. Available: http://http://cvxgen.com/

[7] A. Domahidi, E. Chu, and S. Boyd, "ECOS: An SOCP solver for embedded systems," in *European Control Conference (ECC)*, Zurich, Switzerland, July 2013, pp. 3071–3076. [Online]. Available: https://www.embotech.com/ECOS

[8] A. Stojanov, G. Ofenbeck, T. Rompf, and M. Püschel, "Abstracting vector architectures in library generators: Case study convolution filters," in *ACM International Workshop on Libraries, Languages and Compilers for Array Programming (ARRAY)*, 2014, p. 14.

[9] M. Grant, S. Boyd, and Y. Ye, "Disciplined convex programming," in *Global Optimization: From Theory to Implementation (Nonconvex Optimization and Its Applications)*, L. Liberti and N. Maculan, Eds. Springer Science and Business Media, 2006, pp. 155–210.

[10] M. Grant and S. Boyd, "CVX: matlab software for disciplined convex programming version 2.1," 2015. [Online]. Available: http://cvxr.com/cvx

[11] J. Nocedal and S. Wright, *Numerical Optimization, 2nd ed.* Springer Verlag, 2006.

[12] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C: The Art of Scientific Computing, 3rd Ed.* Cambridge University Press, Cambridge, 20007.

[13] J. Mattingley and S. Boyd, "Automatic code generation for real-time convex optimization," in *Convex Optimization in Signal Processing and Communications*, D. P. Palomar and Y. C. Eldar, Eds. Cambridge: Cambridge University Press, 2009, ch. 1, pp. 1–41.

[14] "LAPACK: Linear Algebra PACKage," 2016. [Online]. Available: http://www.netlib.org/lapack/

[15] Apple Inc, "Accelerate Framework," 2016. [Online]. Available: https://developer.apple.com/reference/accelerate

[16] "BLAS: Basic Linear Algebra Subprograms," 2012. [Online]. Available: http://www.netlib.org/blas/