2017 IEEE 56th Annual Conference on Decision and Control (CDC)
December 12-15, 2017, Melbourne, Australia

Embedded Code Generation Using the OSQP Solver

Goran Banjac⋆, Bartolomeo Stellato⋆, Nicholas Moehle, Paul Goulart, Alberto Bemporad, and Stephen Boyd

***Abstract*— We introduce a code generation software package that accepts a parametric description of a quadratic program (QP) as input and generates tailored C code that compiles into a fast and reliable optimization solver for the QP that can run on embedded platforms. The generated code is based on OSQP, a novel open-source operator splitting solver for quadratic programming. Our software supports matrix factorization caching and warm starting, and allows updates of the problem parameters during runtime. The generated C code is library-free and has a very small compiled footprint. Examples arising in real-world applications show that the generated code outperforms state-of-the-art embedded and desktop QP solvers.**

I. Introduction

Convex optimization has become a standard tool across many engineering fields. In recent years, these methods have increasingly been applied on embedded systems where data are processed in real time and on low-cost computational platforms [1, Ch. 1], [2]. Current applications include, *e.g.*, model predictive control (MPC) [3], real-time signal processing [4], [5] and onboard trajectory planning in space missions [6], [7].

Real-time applications of embedded optimization impose special requirements on the solvers used [8]. First, embedded solvers must be reliable even in the presence of poor quality data, and should avoid exceptions caused by division by zero or memory faults caused by dynamic memory allocation. Second, the solver should be implementable on low-cost embedded platforms with very limited memory resources. In particular, solvers should have very small compiled footprint, should consist only of basic algebraic operations, and should not be linked to any external libraries, which also makes the solver easily verifiable. Finally, real-time applications typically require that the solver is fast and able to correctly identify infeasible problems.

On the other hand, optimization problems arising in embedded applications have certain features that can be exploited when designing an embedded solver [8]. First, embedded optimization is typically applied to the repeated solution of parametrized problems in which the problem data, but not its dimensions or sparsity pattern, change between problem instances. For such problems, the solver initialization and some part of its computations can be performed offline during the solver design phase. Second, requirements on the solution accuracy in embedded applications are often moderate because of noise in the data and arbitrariness of the objective function. Finally, in embedded applications one can typically assume that problems are reasonably scaled. As an example, the authors in [9] show that acceptable control performance of an MPC controller is achievable even when using a very low accuracy solver. This argument supports the use of first-order optimization methods, which are known to return solution of medium accuracy with low computational cost. Although the performance of first-order methods is known to depend strongly on problem scaling, this dependency can be reduced significantly by preconditioning the problem data [10].

A. Related work

In some cases the solution of a parametrized convex optimization problem can be precomputed offline using multi-parametric programming techniques [11], [12]. However, the memory required for storing such solutions grows exponentially with the problem dimensions, making this approach applicable only to small problems.

Over the last decade tools for generating custom online solvers for parametric problems have attracted increasing attention. CVXGEN [8] is a code generation software tool for small-scale parametric quadratic programs (QPs). The generated solver is fast and reliable, but its main disadvantage is that the code size grows rapidly with the problem dimensions. This issue is overcome in FORCES [13], [14] where the code size of the compiled code is broadly constant with respect to the problem dimensions. In HPMPC [15] tailored solvers for MPC are combined with high-performance optimized libraries for linear algebra. ECOS [16], [17] and Bsocp [18] are embedded solvers for a wider class of second-order cone programs (SOCPs). All of the aforementioned solvers are based on primal-dual interior point methods that are tailored for their specific problem classes. A known limitation of these methods is that they cannot use the *warm starting* technique, which is one of the dominant acceleration factors in applications such as MPC [19].

In contrast, qpOASES [20] is based on a parametric active-set method which can effectively use *a priori* information to speed-up computation of a QP solution. On the other hand, since qpOASES is based on dense linear algebra it cannot exploit sparsity in the problem data. Moreover,

⋆G. Banjac and B. Stellato contributed equally to this work.

This work was supported by the European Commission research project FP7-PEOPLE-2013-ITN under grant agreement no. 607957 (TEMPO).

G. Banjac, B. Stellato, and P. Goulart are with the Department of Engineering Science, University of Oxford, Oxford OX1 3PJ, UK. {goran.banjac, bartolomeo.stellato, paul.goulart}@eng.ox.ac.uk

N. Moehle and S. Boyd are with the Department of Electrical Engineering, Stanford University, Stanford CA 94305, USA. {moehle, boyd}@stanford.edu

A. Bemporad is with the IMT School for Advanced Studies Lucca, 55100 Lucca, Italy. alberto.bemporad@imtlucca.it

the computational complexity of active-set methods grows exponentially with the number of constraints.

FiOrdOs [21] uses first-order gradient methods as the basis for the embedded solvers it generates. In the case of a general QP, the methods require a Lipschitz constant of the gradient of the objective function in order to compute the stepsize. Alternatively, FiOrdOs implements an adaptive rule for the stepsize selection, but it requires a new matrix factorization each time the stepsize is updated. QPgen [22] uses optimal preconditioning of the problem data that can improve performance of first-order methods considerably. The main disadvantage of FiOrdOs and QPgen is their inability to detect infeasible problems.

In this paper we introduce a software package that generates tailored C code that compiles into a solver for a user-specified parametric QP. The generated code is based on the open-source solver OSQP [23] which exploits sparsity in the problem data, supports matrix factorization caching and warm starting, and is able to detect infeasible problems.

B. Structure of the paper

In Section II we introduce an operator splitting method which is the basis for the solvers generated. Two different classes of parametric QPs are described in Section III. The code generation software package is presented in Section IV. In Section V we present numerical results on problems arising in real-world applications. Section VI concludes the paper.

II. OSQP solver

We consider the following QP

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}x^T P x + q^T x \\ \text{subject to} & l \leq Ax \leq u, \end{array} \tag{$\mathcal{P}$}$$

where $x \in \mathbb{R}^n$ is the optimization variable. The objective function is defined by a positive semidefinite matrix $P \in \mathbb{S}^n_+$ and a vector $q \in \mathbb{R}^n$, and the constraints by a matrix $A \in \mathbb{R}^{m\times n}$ and vectors $l \in \{\mathbb{R} \cup -\infty\}^m$ and $u \in \{\mathbb{R} \cup +\infty\}^m$ such that $l \leq u$. Linear equality constraints can be enforced by setting $l_i = u_i \in \mathbb{R}$.

A. Algorithm

The algorithm used in OSQP [23] is based on the alternating direction method of multipliers (ADMM) [24] and is described in Algorithm 1. Scalars $\rho > 0$ and $\sigma > 0$ are *penalty parameters*, $\alpha \in (0, 2)$ is a *relaxation parameter*, and Π is the Euclidean projection onto the set $\{z \in \mathbb{R}^m \,|\, l \leq z \leq u\}$ which has a simple closed-form solution

$$\Pi(z) = \max\left(\min(z, u), l\right),$$

where the max and min operators are taken element-wise.

Note that steps 4 to 7 involve only scalar-vector multiplications, vector additions and the projection operator Π. We describe in the next subsection how to solve the linear system in step 3 efficiently.

Algorithm 1 OSQP solver

```
1: given initial values $x^0, z^0, y^0$ and parameters $\rho, \sigma, \alpha$
2: repeat
3:     solve $\begin{bmatrix} P+\sigma I & A^T \\ A & -\frac{1}{\rho}I \end{bmatrix}\begin{bmatrix}\tilde{x}^{k+1} \\ \nu^{k+1}\end{bmatrix} = \begin{bmatrix}\sigma x^k - q \\ z^k - \frac{1}{\rho}y^k\end{bmatrix}$
4:     $\tilde{z}^{k+1} \leftarrow z^k + \frac{1}{\rho}(\nu^{k+1} - y^k)$
5:     $x^{k+1} \leftarrow \alpha\tilde{x}^{k+1} + (1-\alpha)x^k$
6:     $z^{k+1} \leftarrow \Pi\left(\alpha\tilde{z}^{k+1} + (1-\alpha)z^k + \frac{1}{\rho}y^k\right)$
7:     $y^{k+1} \leftarrow y^k + \rho\left(\alpha\tilde{z}^{k+1} + (1-\alpha)z^k - z^{k+1}\right)$
8: until termination conditions are satisfied
```

B. Solving the linear system

Since the coefficient matrix in Algorithm 1 line 3 does not change between iterations of the algorithm, it must be factorized only once. This factorization is then cached and used in forward and backward solves in all subsequent iterations.

The coefficient matrix is quasi-definite, *i.e.*, it can be written as a 2-by-2 block-symmetric matrix where the $(1, 1)$-block is positive definite, and the $(2, 2)$-block is negative definite. It therefore always has a well defined LDL^T factorization, with L being a lower triangular matrix with unit diagonal elements and D a diagonal matrix with nonzero diagonal elements [25]. For any sparse quasidefinite matrix K, efficient algorithms can be used to compute a permutation matrix P for which the factorization $PKP^T = LDL^T$ results in a sparse matrix factor L [26]. We use the open-source approximate minimum degree (AMD) code [27] to compute such a permutation matrix.

In order to perform the LDL^T factorization of a sparse matrix K (or permuted matrix PKP^T) efficiently, the sparsity pattern for the factor L should be found before performing any numerical operations. Determining this sparsity pattern is known as *symbolic factorization* and requires only the nonzero structure of the matrix K, and not its numerical values. After the symbolic factorization finds the pattern of nonzero elements of L, the numerical values of these elements can be computed. This procedure is known as *numerical factorization*. Note that if the non-zero entries of the matrix K change, but the sparsity pattern and quasidefiniteness are preserved, then only the numerical factorization step needs to be performed again and the memory required to store the new factorization does not change.

We use the open-source SuiteSparse package [28] to perform symbolic and numerical factorizations. The computational complexity of factorization and forward and backward solves with SuiteSparse, as well as the memory required for storing the factor L, depend only on the number of nonzero elements in K and not on its dimensions. The code is thus efficient in terms of both the memory and computational effort required to solve the linear system in Algorithm 1 with sparse matrices P and A.

Observe that the matrix P is not required to be positive semidefinite for the linear system to be solvable. In particular, as long as $P + \sigma I$ is positive definite, the LDL^T factorization of the matrix in Algorithm 1 will return a factor D with nonzero diagonal elements, and exceptions caused by division by zero cannot occur when solving the linear system. This is critical in embedded applications in the presence of poor quality data.

C. Termination criterion

The algorithm used in OSQP generates at each iteration k a tuple (x^k, z^k, y^k) for which we define the primal and dual residuals as

$$r^k_{\text{prim}} = Ax^k - z^k,$$
$$r^k_{\text{dual}} = Px^k + q + A^T y^k.$$

If the problem $\mathcal{P}$ is solvable then the residuals converge to zero vectors as $k \to \infty$ [24]. If the residuals are small, we say that (x^k, y^k) is an approximate primal-dual solution of problem $\mathcal{P}$. A reasonable termination criterion is that the norms of the residuals r^k_{prim} and r^k_{dual} are smaller than some tolerance levels $\varepsilon_{\text{prim}} > 0$ and $\varepsilon_{\text{dual}} > 0$, respectively. Note that the tolerance levels are often chosen relative to the scaling of the algorithm iterates; see [24, Sec. 3.3] for details.

Detection of infeasible problems has been historically a weak point of first-order methods. An exception is the algorithm in [10] that can detect infeasibility in conic problems, and the works in [29], [30] that detect infeasible QPs under additional assumptions on the problem data. OSQP can detect primal and dual infeasibility when the problem is unsolvable. In particular, if the problem is primal infeasible, the algorithm generates a certificate of infeasibility, *i.e.*, a vector $v \in \mathbb{R}^m$ that satisfies

$$A^T v = 0, \quad u^T v_+ + l^T v_- < 0,$$

where $v_+ = \max(v, 0)$ and $v_- = \min(v, 0)$. Likewise, if the problem is dual infeasible the algorithm generates a vector $s \in \mathbb{R}^n$ that satisfies

$$Ps = 0, \quad q^T s < 0, \quad (As)_i \begin{cases} = 0 & l_i \in \mathbb{R}, u_i \in \mathbb{R} \\ \geq 0 & l_i \in \mathbb{R}, u_i = +\infty \\ \leq 0 & u_i \in \mathbb{R}, l_i = -\infty \end{cases}$$

which is a certificate of dual infeasibility. We refer the reader to [31] for details.

III. Parametric programs

In many applications problem $\mathcal{P}$ is solved for varying data, and is thus referred to as a *parametric program*. We make a distinction between two cases depending on which of the problem data are to be treated as parameters. We assume throughout that the problem dimensions m and n, and the sparsity patterns of P and A are fixed.

A. Vectors as parameters

If the vectors q, l and u in problem $\mathcal{P}$ are the only parameters, then the coefficient matrix in Algorithm 1 does not change across different instances of the parametric program. The matrix can then be pre-factored offline and only

```python
import osqp

# Create an OSQP object
m = osqp.OSQP()

# Solver initialization
m.setup(P, q, A, l, u, settings)

# Generate code
m.codegen('code', project_type='Makefile',
          parameters='vectors')
```

Listing 1. A simple Python script for generating the code for a given parametric QP.

backward and forward solves are performed during code execution. This enables a significant reduction of the code footprint. If the diagonal matrix D^{-1} is stored instead of D, then the resulting algorithm is division-free.

This important class of parametric programs arises, for instance, in linear MPC [32], linear regression with (weighted) ℓ_1 regularization [33], [34] and portfolio optimization [1].

B. Matrices and vectors as parameters

If values in matrices P and A are updated, then the coefficient matrix in Algorithm 1 must be refactored. We assume, however, that the sparsity patterns of P and A are fixed. In this case the symbolic factorization of the matrix does not change and only numerical factorization needs to be redone. This implies that no dynamic memory allocation is performed during the code execution.

This class of problems arises, *e.g.*, in nonlinear MPC, nonlinear moving horizon estimation (MHE) [32] and sequential quadratic programming (SQP).

C. Warm starting

When a series of similar optimization problems is solved the solutions across problem instances are often similar. Since the algorithm's running time depends largely on the distance between the algorithm's initial iterate and the problem's set of optimizers, one can set a solution of the previous problem instance as the initial iterate in the next instance. This strategy is known as *warm starting* and often improves running times of iterative optimization algorithms [19].

IV. Code generation with OSQP

OSQP is an open-source operator splitting QP solver written in the C language, with interfaces to high-level languages including Matlab, Python and Julia.

Listing 1 shows a simple Python script that generates code for a given problem family. To generate a solver, the end-user must provide the problem data and (optionally) configure the solver settings. The end-user has some flexibility to customize the solver prior to code generation. For instance, if the setting `early_terminate` is set to 1, then the solver will terminate when one of the termination criteria is satisfied, or when the maximum number of iterations is reached, whichever happens first. Checking the termination

```c
#include "osqp.h"
#include "workspace.h"

int main(int argc, char **argv) {

   // Solve problem
   osqp_solve(&workspace);

   return 0;
};
```

Listing 2. A simple C program that loads the problem data from header file `workspace.h` and solves the problem.

```
<dir_name>
 ├─ include
 │   └─ [*.h]
 ├─ src
 │   ├─ osqp
 │   │   └─ [*.c]
 │   └─ example.c
 └─ CMakeLists.txt
```

Fig. 1. The tree structure of the generated code. The main program is stored in `example.c`.

criteria in each iteration is computationally expensive since it involves several matrix-vector multiplications, and may slow down the code execution considerably. If the user instead sets `early_terminate` to 0, then the algorithm will run for the maximum number of iterations without checking the termination criteria. Alternatively, one can check the termination criteria every N iterations; this is specified with the setting `early_terminate_interval`. For the complete list of solver settings we refer the reader to [23].

To generate the code, the `codegen` method is called with the name of a target directory where the generated code is to be exported. Using the keyword argument `project_type` the user can define the build environment, *e.g.*, Makefiles or several supported IDEs such as Eclipse, Apple Xcode or Microsoft Visual Studio. The keyword argument `parameters` allows the user to specify which of the data are to be parameters. The option `vectors` indicates that only vectors q, l and u in problem $\mathcal{P}$ are parameters, while the option `matrices` allows matrices P and A to be parameters as well.

A. Generated files

Figure 1 shows the tree structure of the generated code. Directories `<dir_name>/src/osqp` and `<dir_name>/include` contain the solver source code and headers. The generated code is self-contained, has small footprint, does not perform dynamic memory allocation, and is thus suitable for embedded applications. `CMakeLists.txt` is a CMake configuration file that manages the compilation process in a compiler- and platform-independent manner [35].

The main program is stored in `example.c` whose content is shown in Listing 2. The program loads the problem data

```c
// Update linear cost
osqp_update_lin_cost(&workspace, &q_new);

// Update lower bound
osqp_update_lower_bound(&workspace, &l_new);

// Update upper bound
osqp_update_upper_bound(&workspace, &u_new);
```

Listing 3. Function calls for updating vectors of a parametric QP.

from the header file `workspace.h`, and solves the problem. The problem data must be updated in order to solve a different instance of a parametric problem. Listing 3 provides illustrative function calls for updating vectors in problem $\mathcal{P}$; for the complete documentation we refer the reader to [23].

V. Numerical results

We benchmarked the generated solvers against the open-source code generation tools CVXGEN [8], FiOrdOs [21], the open-source solver qpOASES [20], and the commercial solver GUROBI [36]. All the solvers were selected with their default options. We performed benchmarks on an Apple MacBook Pro 2.8GHz Intel Core i7 with 16GB RAM running Python 3.5. Code for all examples is available at [37].

A. Portfolio optimization

We consider a portfolio optimization problem [1, p. 185–186], [17] where we want to maximize risk-adjusted return. The problem is

$$\begin{array}{ll} \text{maximize} & \mu^T x - \gamma(x^T \Sigma x) \\ \text{subject to} & \mathbf{1}^T x = 1, \quad x \geq 0, \end{array} \tag{1}$$

where $x \in \mathbb{R}^n$ represents the portfolio, $\mu \in \mathbb{R}^n$ is the vector of expected returns, scalar $\gamma > 0$ is the risk-aversion parameter, and $\Sigma \in \mathbb{S}^n_+$ is the asset return covariance. A common assumption is the k-factor risk model [38], where the return covariance matrix is the sum of a diagonal matrix and a matrix of rank k, *i.e.*,

$$\Sigma = D + FF^T,$$

where $F \in \mathbb{R}^{n \times k}$ and $D \in \mathbb{R}^{n \times n}$ is diagonal with nonnegative diagonal elements. Problem (1) can be written in the standard form $\mathcal{P}$ with the linear cost q depending on the parameter γ; see Appendix I for details.

In order to obtain Pareto optimal portfolios, one needs to solve problem (1) for varying risk-aversion parameter γ. Since the parameter appears only in the linear cost, one does not need to perform any matrix factorization once the code is generated. Moreover, seeing that the optimal solution does not differ significantly with small changes in γ, we can make use of warm starting and get a range of Pareto optimal portfolios with minimal computational effort.

The data are generated as follows: $k = \lceil n/10 \rceil$, F has half of its elements set to zero, with the other half drawn from $\mathcal{N}(0, 1)$, diagonal elements of D are drawn from $\mathcal{U}(0, \sqrt{k})$, and the elements of μ are drawn from $\mathcal{N}(0, 1)$. We generate

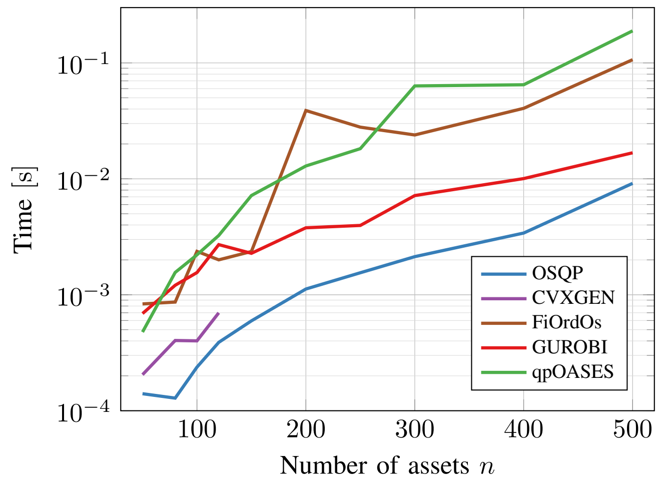

Fig. 2. Portfolio optimization example: the average computation times required to solve 11 instances of problem (1).

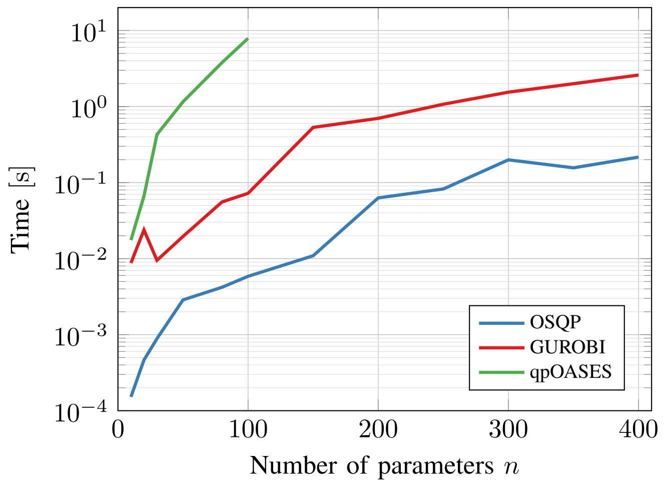

Fig. 3. Lasso example: the average computation times required to solve 21 instances of problem (2).

TABLE I

Portfolio optimization example: Sizes of executable files generated by the OSQP solver.

Assets n	Nonzeros in P and A	Size of file [kB]
50	235	46
80	496	58
100	720	70
120	984	82
150	1455	102
200	2440	142
250	3675	190
300	5160	246
400	8880	382
500	13600	554

11 values of γ equally spaced on a logarithmic scale between 10^{-2} and 10^{2}. For each solver and each dimension n we solve the generated problem for the 11 values of γ and average the execution time.

The results are shown in Figure 2. OSQP consistently outperforms all the other methods. CVXGEN is not able to generate the problem when $n > 120$ since the resulting coefficient matrix has more than 4000 nonzero elements. Note that CVXGEN, FiOrdOs and qpOASES exploit the simple bounds on variable x, while OSQP and GUROBI use the formulation $\mathcal{P}$. Table I shows the sizes of executable files generated by the OSQP solver, inclusive of problem data, as a function of the number of assets. The size of the compiled code for all the tested examples does not exceed 0.55 MB.

B. Sparse regressor selection

We seek a sparse solution of the regressor selection problem, which is in general a hard combinatorial problem [1, Ch. 6.3]. *Lasso* [33] is a popular heuristic for enforcing sparsity of the solution by adding an ℓ_1 regularization term in the objective. The problem is

$$\text{minimize} \quad \tfrac{1}{2}\|Cx - d\|_2^2 + \gamma\|x\|_1, \tag{2}$$

where $x \in \mathbb{R}^n$ is the vector of parameters, $C \in \mathbb{R}^{m\times n}$ is the data matrix whose columns are potential regressors, $d \in \mathbb{R}^m$ is the vector of measurements that is to be fit by a subset of regressors, and $\gamma > 0$ is the weighting parameter. Problem (2) can be written in the standard form $\mathcal{P}$ with the linear cost q depending on the parameter γ; see Appendix II for details.

In order to find a sparse solution with a given degree of sparsity, *i.e.*, not more than k nonzero elements in x, the above problem should be solved for varying weighting parameter γ. As in Section V-A, we can exploit the fact that γ enters only in the linear part of the cost function by caching the matrix factorization and warm starting the solver.

The data are generated as follows: n varies from 10 to 400, $m = 10n$, matrix C has 40% nonzero elements drawn from $\mathcal{N}(0,1)$, $\hat{x} \in \mathbb{R}^n$ has half of the elements set to zero, and the other half are drawn from $\mathcal{N}(0,1/n)$. We then set $d = C\hat{x} + \varepsilon$, where ε represents a vector of noise with elements drawn from $\mathcal{N}(0,1/4)$. We generate 21 values of γ equally spaced on the logarithmic scale from 10^{-2} to 10^{2}. For each solver and each dimension n we solve the generated problem for the 21 values of γ and average the execution time.

The results are shown in Figure 3. FiOrdOs does not converge within $50,000$ iterations for this problem type and CVXGEN is not able to generate code for $n > 10$. OSQP clearly outperforms both GUROBI and qpOASES for all dimensions considered. Note that qpOASES is not able to find a solution in less than 10 seconds for $n > 100$.

VI. Conclusion

This paper introduces a new code generation software tool based on the OSQP solver. The generated code is very efficient and robust. Moreover, it is general purpose and does not require additional assumptions on problem data beyond convexity. Numerical results show that the generated code not only outperforms state-of-the-art embedded code-generation tools but also desktop solvers.

Appendix I
Data for the portfolio optimization example

Problem (1) can be written in form $\mathcal{P}$ with the following problem data

$$P = \begin{bmatrix} D & \\ & I \end{bmatrix}, \quad q = \begin{bmatrix} -\frac{1}{2\gamma}\mu \\ 0 \end{bmatrix},$$
$$A = \begin{bmatrix} \mathbf{1}^T & \\ F^T & -I \\ I & \end{bmatrix}, \quad l = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad u = \begin{bmatrix} 1 \\ 0 \\ \mathbf{1} \end{bmatrix}.$$

Appendix II
Data for the lasso example

Problem (2) can be written in form $\mathcal{P}$ with the following problem data

$$P = \begin{bmatrix} 0 & & \\ & I & \\ & & 0 \end{bmatrix}, \quad q = \begin{bmatrix} 0 \\ 0 \\ \gamma\mathbf{1} \end{bmatrix},$$
$$A = \begin{bmatrix} C & -I & \\ I & & I \\ -I & & I \end{bmatrix}, \quad l = \begin{bmatrix} d \\ 0 \\ 0 \end{bmatrix}, \quad u = \begin{bmatrix} d \\ +\infty \\ +\infty \end{bmatrix}.$$

References

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

[2] S. Richter, C. N. Jones, and M. Morari, "Certification aspects of the fast gradient method for solving the dual of parametric convex programs," *Mathematical Methods of Operations Research*, vol. 77, no. 3, pp. 305–321, 2013.

[3] L. Biens and M. Kothare, "Real-time implementation of model predictive control," in *American Control Conference (ACC)*, vol. 6, 2005, pp. 4166–4171.

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

[5] B. Defraene, T. Van Waterschoot, H. J. Ferreau, M. Diehl, and M. Moonen, "Real-time perception-based clipping of audio signals using convex optimization," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 20, no. 10, pp. 2657–2671, 2012.

[6] L. Blackmore, B. Açikmeşe, and D. Scharf, "Minimum-landing-error powered-descent guidance for Mars landing using convex optimization," *Journal of Guidance, Control, and Dynamics*, vol. 33, no. 4, pp. 1161–1171, 2010.

[7] D. P. Scharf, B. Açikmeşe, D. Dueri, J. Benito, and J. Casoliva, "Implementation and experimental demonstration of onboard powered-descent guidance," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 2, pp. 213–229, 2017.

[8] J. Mattingley and S. Boyd, "CVXGEN: A code generator for embedded convex optimization," *Optimization and Engineering*, vol. 13, no. 1, pp. 1–27, 2012.

[9] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," *IEEE Transactions on Control Systems Technology*, vol. 18, no. 2, pp. 267–278, 2010.

[10] 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*, vol. 169, no. 3, pp. 1042–1068, 2016.

[11] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, pp. 3–20, 2002.

[12] P. Tøndel, T. A. Johansen, and A. Bemporad, "An algorithm for multi-parametric quadratic programming and explicit MPC solutions," *Automatica*, vol. 39, no. 3, pp. 489–497, 2003.

[13] A. Domahidi, A. U. Zgraggen, M. N. Zeilinger, M. Morari, and C. N. Jones, "Efficient interior point methods for multistage problems arising in receding horizon control," in *IEEE Conference on Decision and Control (CDC)*, 2012, pp. 668–674.

[14] A. Domahidi. (2012) FORCES: Fast optimization for real-time control on embedded systems. [Online]. Available: http://forces.ethz.ch

[15] G. Frison, H. H. B. Sorensen, B. Dammann, and J. B. Jorgensen, "High-performance small-scale solvers for linear model predictive control," in *European Control Conference (ECC)*, 2014, pp. 128–133.

[16] A. Domahidi, E. Chu, and S. Boyd, "ECOS: An SOCP solver for embedded systems," in *European Control Conference (ECC)*, 2013, pp. 3071–3076.

[17] E. Chu, N. Parikh, A. Domahidi, and S. Boyd, "Code generation for embedded second-order cone programming," in *European Control Conference (ECC)*, 2013, pp. 1547–1552.

[18] D. Dueri, J. Zhang, and B. Açikmeşe, "Automated custom code generation for embedded, real-time second order cone programming," in *IFAC World Congress*, vol. 47, no. 3, 2014, pp. 1605–1612.

[19] M. Herceg, C. N. Jones, and M. Morari, "Dominant speed factors of active set methods for fast MPC," *Optimal Control Applications and Methods*, vol. 36, no. 5, pp. 608–627, 2015.

[20] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl, "qpOASES: a parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, 2014.

[21] F. Ullmann and S. Richter. (2014) FiOrdOs: Code generation for first-order methods, version 2.0. [Online]. Available: http://fiordos.ethz.ch

[22] P. Giselsson and S. Boyd, "Linear convergence and metric selection for Douglas-Rachford splitting and ADMM," *IEEE Transactions on Automatic Control*, vol. 62, no. 2, pp. 532–544, 2017.

[23] B. Stellato and G. Banjac. (2017) OSQP: An operator splitting solver for quadratic programs. [Online]. Available: http://osqp.readthedocs.io

[24] 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*, vol. 3, no. 1, pp. 1–122, 2011.

[25] R. Vanderbei, "Symmetric quasi-definite matrices," *SIAM Journal on Optimization*, vol. 5, no. 1, pp. 100–113, 1995.

[26] T. A. Davis, *Direct Methods for Sparse Linear Systems (Fundamentals of Algorithms 2)*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2006.

[27] P. R. Amestoy, T. A. Davis, and I. S. Duff, "Algorithm 837: AMD, an approximate minimum degree ordering algorithm," *ACM Transactions on Mathematical Software*, vol. 30, no. 3, pp. 381–388, 2004.

[28] T. A. Davis, "Algorithm 849: A concise sparse Cholesky factorization package," *ACM Transactions on Mathematical Software*, vol. 31, no. 4, pp. 587–591, 2005.

[29] A. U. Raghunathan and S. Di Cairano, "Infeasibility detection in alternating direction method of multipliers for convex quadratic programs," in *IEEE Conference on Decision and Control (CDC)*, 2014, pp. 5819–5824.

[30] V. V. Naik and A. Bemporad, "Embedded mixed-integer quadratic optimization using accelerated dual gradient projection," in *IFAC World Congress*, 2017.

[31] G. Banjac, P. Goulart, B. Stellato, and S. Boyd, "Infeasibility detection in the alternating direction method of multipliers for convex optimization," available: http://www.optimization-online.org/DB_HTML/2017/06/6058.html, Jun 2017.

[32] J. B. Rawlings and D. Q. Mayne, *Model Predictive Control: Theory and Design*. Nob Hill Publishing, 2009.

[33] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society: Series B*, vol. 58, no. 1, pp. 267–288, 1996.

[34] E. J. Candés, M. B. Wakin, and S. Boyd, "Enhancing sparsity by reweighted ℓ_1 minimization," *Journal of Fourier Analysis and Applications*, vol. 14, no. 5, pp. 877–905, 2008.

[35] Kitware, Inc. (2012) CMake. [Online]. Available: https://cmake.org

[36] Gurobi Optimization Inc. (2016) Gurobi optimizer reference manual. [Online]. Available: http://www.gurobi.com

[37] B. Stellato and G. Banjac. (2017) OSQP code generation benchmarks. [Online]. Available: https://github.com/oxfordcontrol/osqp_codegen_benchmarks

[38] G. Connor and R. A. Korajczyk, "The arbitrage pricing theory and multifactor models of asset returns," in *Finance*, ser. Handbooks in Operations Research and Management Science, 1995, vol. 9, pp. 87–144.