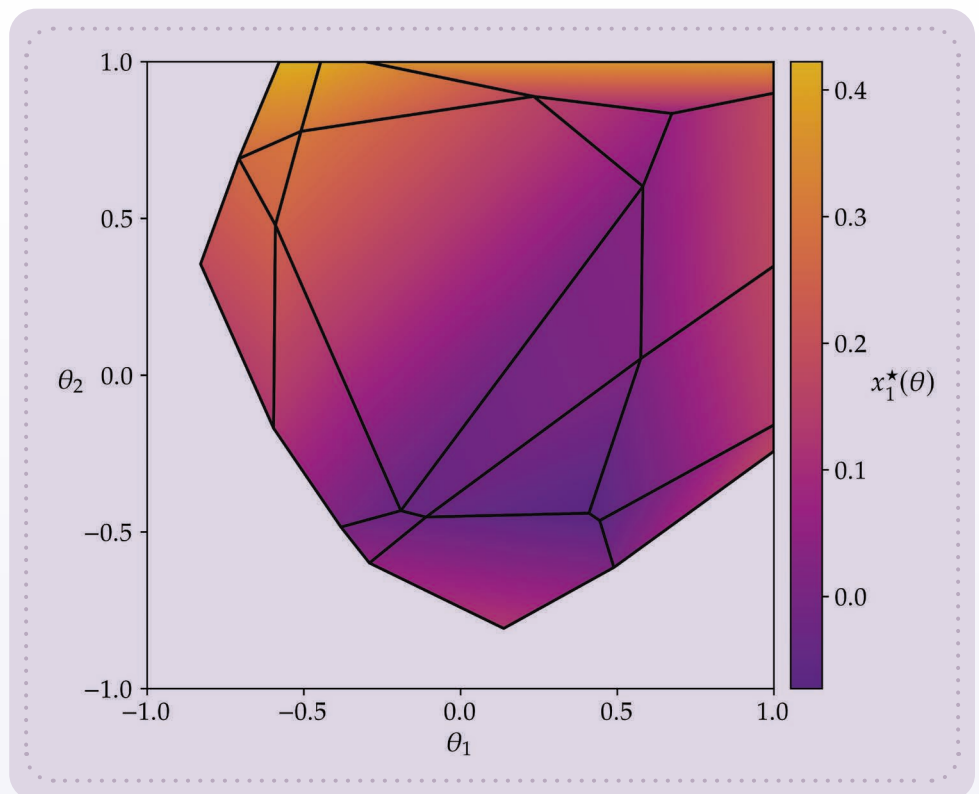


Automatic Generation of Explicit Quadratic Programming Solvers

A STREAMLINED PATH FROM THE MATH TO FLAT C CODE

MAXIMILIAN SCHALLER¹, DANIEL ARNSTRÖM²,
ALBERTO BEMPORAD³, and STEPHEN P. BOYD⁴

It is well known that the solution of a parametrized QP is a piecewise affine function of the parameter when the parameter enters only the linear objective term and the right-hand side of the constraints in an affine way. The number of (polyhedral) regions in the solution map can grow exponentially in problem size, but when the number of regions is moderate, an explicit solver is practical. Such a solver computes the coefficients of the affine functions and the linear inequalities defining the polyhedral regions offline; to solve a problem instance online, it simply evaluates this explicit solution map. Potential advantages of an explicit solver over a more general-purpose iterative solver can include interpretability, reliability, and computational efficiency. This article focuses on the use of code generation to automatically (and deterministically) generate an explicit solver from a high-level description of a parametrized QP (see “Summary”). Our method has been implemented in the open source software CVXPYgen, which is part of CVXPY, a domain-specific language (DSL) for general convex optimization.



INTRODUCTION

Convex optimization is used in various domains, including control systems [1], [2], [3], [4], [5], [6], [7], [8], signal and image processing [9], [10], [11], [12], and quantitative finance [13], [14], [15], [16], [17], [18], just to name a few that are particularly relevant to this work.

Parametric Convex Optimization

A parametric convex optimization problem can be written as

$$\begin{aligned}
& \text{minimize} && f_0(x, \theta) \\
& \text{subject to} && f_i(x, \theta) \leq 0, \quad i = 1, \dots, m \\
& && h_i(x, \theta) = 0, \quad i = 1, \dots, q
\end{aligned} \tag{1}$$

where $x \in \mathbf{R}^n$ is the variable, and $\theta \in \Theta \subseteq \mathbf{R}^p$ is the parameter, that is, data that are given and known whenever (1) is solved to find x . The objective function f_0 and the inequality constraint functions f_i , $i = 1, \dots, m$, are convex in x , and the equality constraint functions h_i , $i = 1, \dots, q$, are affine in x , for any given value of $\theta \in \Theta$ [19]. We refer to a solution of (1) as $x^*(\theta)$ to emphasize its dependence on the parameter θ . Here, we neglect that x^* may not be unique or may not exist and refer to the mapping from θ to x as the *solution map* of the parametrized problem.

Explicit Solvers for Multiparametric Programming

Traditionally, $x^*(\theta)$ is evaluated using an iterative numerical method that takes a given parameter value and computes an (almost) optimal point $x^*(\theta)$ [20], [21], [22], [23]. We focus here on a very special case in which $x^*(\theta)$ can be expressed in a closed form as an explicit function that maps a given value of θ directly to a solution $x^*(\theta)$.

We call a program that evaluates this explicit function an *explicit solver*, and we call a program that finds this explicit function an *explicit solver generator*, or the *offline phase* of an explicit solver. Such explicit solvers are practical only for some problems, and generally only for smaller instances, but when they are practical, they can offer a number of advantages over generic iterative solvers. Developing such solvers for parametric programs is now known as *multiparametric programming*.

Related Work

Multiparametric Programming

Investigations of the theoretical properties of parametric convex optimization problems date back to the 1960s [24] and were largely extended in the 1980s [25]. After early work on explicitly solving parametric linear programs (LPs) in the context of economics [26] in the 1970s, researchers started investigating the explicit solution of different convex optimization problems in the early 2000s. Some of the earliest articles developed explicit solutions to model predictive control problems based on QPs [27] and LPs [28], and general algorithms were developed for explicitly solving QPs [27], [29], [30], [31] and LPs [32]. These methods are often called *multiparametric linear programming* or *multiparametric quadratic programming*, respectively, where the latter is often abbreviated as MPQP. Further, research has been reported on verifying the complexity of such methods [33], on approximate or suboptimal multiparametric programming [34], [35], [36], on multiparametric programming for linear complementary problems [37], and on synthesizing specialized hardware for multiparametric programming [38], [39], [40]. Software implementations include the Hybrid

Summary

We consider a family of convex quadratic programs (QPs) in which the coefficients of the linear objective term and the right-hand side of the constraints are affine functions of a parameter. Such problem families often occur in fields such as control, machine learning, and finance. Instead of solving the problem from scratch for each parameter assignment, it is advantageous, for sufficiently small problems, to determine a so-called explicit solution as a function of these parameters as this approach allows transparency, reliability, and speed. This article proposes the use of (deterministic) code generation to automatically generate this explicit solver, in C, starting from a high-level description of a parametrized QP, as implemented in CVXPYgen.

Toolbox [41], the Multi-Parametric Toolbox 3 (MPT3) [42], the Model Predictive Control Toolbox [43], and the POP Toolbox [44] in MATLAB, the MPQP solver in the proprietary FORCES PRO software [45], and the PDAQP solver [46] in Julia and Python, which we use in this work.

The standard method for evaluating the piecewise affine functions computed by these software packages is a balanced binary search tree [47]. The major drawback with this approach is that the offline computational complexity for balancing such trees can be limiting. Alternatives for evaluating piecewise affine functions include convex lifting [48], [49], axis-aligned and truncated binary search trees [35], [51], and approximation through basis functions [52]. The lower offline computational complexity for these approaches comes at the cost of slower online evaluation times, higher memory footprints, or approximate solutions.

Potential advantages of an explicit solver over a more general-purpose iterative solver can include transparency, interpretability, reliability, and speed. Since the solver is essentially a lookup table with an explicit affine function associated with each region, there is no question of convergence. Indeed, we can explicitly determine the maximum number of floating-point (FP) operations required to compute $x^*(\theta)$ given θ [33], [53]. An explicit solver involves no division, so FP overflow or divide-by-zero exceptions cannot occur. For the same reason, it is possible to store the coefficients in a lower-precision format, such as 16-bit FP (FP16), to reduce storage and possibly to carry out the computations in lower precision as well to increase speed or to use low-cost electronic boards (at the cost of a modest decrease in accuracy).

The disadvantages of using an explicit solver all relate to its worst-case exponential scaling with problem size. This limits its practical use to relatively small problems, which, however, do arise in many application areas. Even when it is practical to use an explicit solver, the solver data size can be large since we must store the coefficients of the explicit solution map.

Potential advantages of an explicit solver over a more general-purpose iterative solver can include transparency, interpretability, reliability, and speed.

Code Generation for Convex Optimization

While typical convex optimization solvers are designed for general-purpose computers [20], [54], we are mostly interested in embedded applications with hard real-time constraints, and also, nonembedded applications where extreme speeds are required. A *code generator* exploits the structure of the functions in (1) and generates custom C code to solve the problem quickly and reliably for changing values of θ while adhering to safety-critical code guidelines [55]. Examples of code generators for iterative solvers are CVXGEN [56] (for general QPs), CVXPYgen [57], which interfaces with the OSQP code generator [58], and QOCOGEN [59] (for QPs and second-order cone programs, respectively) and the proprietary FORCES PRO [45], specifically designed for QP-based and nonlinear control problems. Similarly, *acados* [60] is a framework for embedded optimal control, directly written in C, and provides high-performance solvers via its modular architecture instead of custom code generation. Code generators are used for many applications. CVXGEN, for example, is used to guide and control all of the SpaceX first-stage landings [61].

DSLs for Optimization

Code generators typically accept problem specifications given in a DSL for convex optimization [62], [63]. A DSL allows the user to describe the problem in a natural, high-level, and humanly readable way, eliminating the effort (and risk of error) in transforming the problem to the standard form required by a solver. For example, there can be multiple variables or parameters, with names that make sense in the application; the code generator takes care of mapping them to our generic $x \in \mathbf{R}^n$ and $\theta \in \Theta$. Well-known DSLs include YALMIP [62] and CVX [63] in MATLAB, CVXPY [54] in Python, Convex.jl [64] and JuMP [65] in Julia, and CVXR [66] in R. We focus on CVXPY.

Contribution

In this article, we extend the code generator CVXPYgen so that it can interface with the multiparametric explicit QP solver PDAQP to generate explicit solvers for QP-representable convex optimization problems described in CVXPY. Along with C and C++ code for the generated solver, we generate a Python interface for rapid prototyping and non-embedded applications.

Integrating an explicit QP solver into the code generation system CVXPYgen makes it easy for a user to go

directly from a prototype simulation, developed in a Python environment, to low-level code suitable for embedded and real-time applications, with just a few small changes in the CVXPYgen configuration. While it was possible before to prototype in CVXPY and implement using an explicit code generator such as PDAQP, our integrated system makes this much more streamlined.

We give four representative application examples, involving linear regression, power management in residential buildings, model predictive control, and financial portfolio optimization. Our code generator accelerates the solve time for these examples between one and more than two orders of magnitude compared to directly using CVXPY and its default QP solver, with solve times down to hundreds of nanoseconds (on a standard laptop).

Outline

We begin with a quick derivation of the explicit solution map for a parametric QP. Then, we explain how we generate source code for an explicit solver described in the modeling language CVXPY and how to interface with the code. After illustrating the explicit solver code generation process with a simple example, we assess the numerical performance of the explicit solvers for several practical examples. We report the time it takes to generate and compile the code, the size of the resulting binary files, and the solve times.

EXPLICIT SOLUTION OF PARAMETRIC QPS

Parametric QP

We consider the QP

$$\begin{aligned} & \text{minimize} \quad (1/2)x^T Px + q^T x \\ & \text{subject to} \quad Ax \leq b \end{aligned} \quad (2)$$

where $x \in \mathbf{R}^n$ is the variable, and the data are $P \in \mathbf{S}_{++}^n$ (the set of symmetric positive definite $n \times n$ matrices), $q \in \mathbf{R}^n$, $A \in \mathbf{R}^{m \times n}$, and $b \in \mathbf{R}^m$. The inequality constraints are interpreted elementwise.

Our focus is on the case when the data P and A are given, and q and b are affine functions of a parameter $\theta \in \mathbf{R}^p$

$$q = u + U\theta, \quad b = v + V\theta \quad (3)$$

where $u \in \mathbf{R}^n$, $U \in \mathbf{R}^{n \times p}$, $v \in \mathbf{R}^m$, and $V \in \mathbf{R}^{m \times p}$ are given. We refer to the QP (2) parametrized by θ in (3) as a

The disadvantages of using an explicit solver all relate to its worst-case exponential scaling with problem size.

parametrized QP. Since the objective is strictly convex, there is at most one solution of the QP (2) for each value of θ . We refer to the mapping from θ to the optimal x (when it exists) as the solution map of the parametrized QP (2).

Other QP Forms

There are several other standard forms for a parametrized QP, both for analysis and as an interface to solvers [27], [29], [46], but it is easy to translate between them by introducing additional variables and constraints [19].

Constraints on the Parameters

In many applications, we are also given a set $\Theta \subseteq \mathbb{R}^p$ of possible parameter values. For simplicity, we ignore this, but we occasionally mention how this set of possible values of θ can be handled. When we do address the parameter set, we assume it is a polyhedron.

Optimality Conditions

Active Constraints

We denote the i th row of A as a_i^T , so the inequality constraints in (2) can be expressed as $a_i^T x \leq b_i$, $i = 1, \dots, m$. We say that the i th inequality constraint is tight or active if $a_i^T x = b_i$. We let $\mathcal{A} = \{i \mid a_i^T x = b_i\} \subseteq \{1, \dots, m\}$ denote the set of active constraints [33], [67] (which depends on x , A , and b).

Let $\lambda \in \mathbb{R}^m$ denote a dual variable associated with the linear inequality constraints in (2). The optimality conditions for problem (2) are

$$\begin{aligned} Ax &\leq b, \\ \lambda &\geq 0, \\ Px + q + A^T \lambda &= 0, \\ \lambda_i(a_i^T x - b_i) &= 0, \quad i = 1, \dots, m. \end{aligned}$$

The first is primal feasibility; the second is the nonnegativity of dual variables; the third is dual feasibility (the stationarity of the Lagrangian); and the last one is complementary slackness [19, §5.5.3].

Let $\tilde{A}$, $\tilde{b}$, and $\tilde{\lambda}$ denote the row slices of A , b , and λ , respectively, corresponding to the active constraints, that is, $i \in \mathcal{A}$. Let $\hat{A}$, $\hat{b}$, and $\hat{\lambda}$ denote the row slices of A , b , and λ , respectively, corresponding to the inactive constraints, that is, $i \notin \mathcal{A}$. By complementary slackness, we must have $\lambda_i = 0$ for $i \notin \mathcal{A}$, so $\hat{\lambda} = 0$. With $\hat{\lambda} = 0$, which we now assume, complementary slackness holds. Since $\hat{\lambda} = 0$, $A^T \lambda$

can be expressed as $\tilde{A}^T \tilde{\lambda}$, and dual feasibility can be expressed as

$$Px + q + \tilde{A}^T \tilde{\lambda} = 0.$$

Since $\mathcal{A}$ is the active set corresponding to x , we have

$$\tilde{A}x = \tilde{b}.$$

These two sets of linear equations can be summarized as the Karush-Kuhn-Tucker (KKT) system

$$\begin{bmatrix} P & \tilde{A}^T \\ \tilde{A} & 0 \end{bmatrix} \begin{bmatrix} x \\ \tilde{\lambda} \end{bmatrix} = \begin{bmatrix} -q \\ \tilde{b} \end{bmatrix}. \quad (4)$$

We assume that the linear independence constraint qualification (LICQ) [19], [68], [69] holds, that is, that the rows of $\tilde{A}$ are linearly independent. Then, it is well known that (4) can be uniquely solved for $(x, \tilde{\lambda})$ [19, §10.1.1], [70, §12.3], and we rewrite (4) as

$$\begin{bmatrix} x \\ \tilde{\lambda} \end{bmatrix} = \begin{bmatrix} P & \tilde{A}^T \\ \tilde{A} & 0 \end{bmatrix}^{-1} \begin{bmatrix} -q \\ \tilde{b} \end{bmatrix}. \quad (5)$$

This shows that knowledge of the active set $\mathcal{A}$ determines the primal and dual solutions of the QP (2). We note that the pair $(x, \tilde{\lambda})$ is the solution of the problem of minimizing the objective of the QP subject to the linear equality constraints $\tilde{A}x = \tilde{b}$.

From (5), we see that the solution is a linear function of the data q and b , provided that the active set does not change. This implies that the solution is an affine function of the parameter θ , provided that the active set remains unchanged.

When $(x, \tilde{\lambda})$ have the values (5) (with $\hat{\lambda} = 0$), they satisfy complementary slackness and dual feasibility. The remaining two optimality conditions, primal feasibility and dual nonnegativity, can be expressed as

$$\begin{bmatrix} \hat{A} & 0 \\ 0 & -I \end{bmatrix} \begin{bmatrix} P & \tilde{A}^T \\ \tilde{A} & 0 \end{bmatrix}^{-1} \begin{bmatrix} -q \\ \tilde{b} \end{bmatrix} \leq \begin{bmatrix} \hat{b} \\ 0 \end{bmatrix} \quad (6)$$

since $Ax \leq b$ holds (as equality) for rows with $i \in \mathcal{A}$ and $\hat{\lambda} = 0$. Inequality (6) is a set of linear inequalities in data b and q and therefore defines a polyhedron. When (b, q) is in this polyhedron, called the *critical region* associated with the active set $\mathcal{A}$, $(x, \tilde{\lambda})$ given by the linear function (5) are primal and dual optimal for QP (2).

Since the compositions of affine functions are affine, it follows that the primal and dual solutions of the QP (2) are (locally) affine functions of θ . Since the inverse image of a polyhedron under an affine mapping is a polyhedron, the values of θ over which this affine function gives the solution are also a polyhedron. Thus, the solution map is a piecewise affine function of θ , with the polyhedral regions determined by the active set. Because the solution is unique, it is not difficult to prove that this map is also continuous across region boundaries [27]. This continuity property guarantees a certain degree of robustness to numerical errors when evaluating the map.

Explicit Parametric QP Solver

The optimality conditions discussed previously suggest a naive explicit solver, which can be practical when m is small. We search over all 2^m potential active sets. For each, we compute x and λ via (5) and then check whether (6) is valid. If this happens, we have found the solution; if not, the problem is infeasible.

Now we consider the parameter dependence. For each of the 2^m potential active sets, we can express (5) as

$$(x, \tilde{\lambda}) = F\theta + g, \quad \hat{\lambda} = 0$$

where

$$F = \begin{bmatrix} P & \tilde{A}^T \\ \tilde{A} & 0 \end{bmatrix}^{-1} \begin{bmatrix} -U \\ \tilde{V} \end{bmatrix}, \quad g = \begin{bmatrix} P & \tilde{A}^T \\ \tilde{A} & 0 \end{bmatrix}^{-1} \begin{bmatrix} -u \\ \tilde{v} \end{bmatrix}$$

where $\tilde{V}$ and $\tilde{v}$ are the (row) slices of V and v , respectively, corresponding to $\mathcal{A}$. We then express (6) in terms of θ as

$$H\theta \leq j$$

where

$$H = \begin{bmatrix} \hat{A} & 0 \\ 0 & -I \end{bmatrix} F - \begin{bmatrix} \hat{V} \\ 0 \end{bmatrix}, \quad j = -\begin{bmatrix} \hat{A} & 0 \\ 0 & -I \end{bmatrix} g + \begin{bmatrix} \hat{v} \\ 0 \end{bmatrix}.$$

For some choices of potential active sets, the inequalities $H\theta \leq j$ (together with $\theta \in \Theta$) are infeasible. We drop these and consider only the remaining K potential active sets, labeling them $k = 1, \dots, K$. We can compute the coefficients of the piecewise affine solution map, denoted F_k and g_k , and their associated region, defined by H_k and j_k , before knowing the specific value of θ . Thus, we have the explicit solution map

$$(x, \lambda) = F_k \theta + g_k \quad \text{when} \quad H_k \theta \leq j_k, \quad k = 1, \dots, K. \quad (7)$$

Different existing MPQP solvers differ in how they avoid enumerating all 2^m active sets [46, §II-c]. When $\theta \in \Theta$ satisfies none of the previous inequalities, the QP is infeasible. The collection of coefficient matrices and vectors $F_k, g_k, H_k, j_k, k = 1, \dots, K$ gives an explicit representation of the

solution map of the parametrized QP (2). Since we can compute these matrices and vectors (and determine the value of K) before we have specified θ , we refer to this computation as an *offline solution*. Evaluating (7) for a given value of θ is called the *online solve*. Note that it involves no division.

The number of coefficients in the explicit solver is roughly $K(n+m)p$, up to a factor of K larger than the number of coefficients in the original problem, $n^2 + nm + (n+m)p$ (neglecting sparsity). Even though K can grow exponentially with m , it is often practical for small problems [27], [29], [33].

Implementation

When we explicitly solve a parametrized QP in practice, some of our initial assumptions can be relaxed. We can handle positive semidefinite P (instead of just positive definite); we directly handle equality constraints; and we do not require LICQ. (When P is not positive definite, the solver provides a solution, rather than *the* solution, since the solution need not be unique in this case.)

In the offline phase, implementations do not search all 2^m possible active sets but instead find nonempty regions one by one. This allows us to handle problems where 2^m is very large, but K , the number of (nonempty) regions, remains moderate.

In the online solve, explicit solvers store the regions in a way that facilitates faster search than a simple linear search over $k = 1, \dots, K$, typically involving a precomputed tree. Other methods are used to reduce the storage or increase the speed of the online evaluations.

We use the specific solver PDAQP [46], which has already implemented several such accelerations. The offline solve is made more efficient by systematic searching over neighboring regions. The online solve uses a balanced binary search tree [47]. PDAQP balances the tree efficiently by solving feasibility problems using the solver DAQP [71]. The tight integration with DAQP for the balancing, alongside other implementation choices, results in PDAQP usually forming a balanced binary search tree several orders of magnitude faster than, for example, MPT3.

The resulting binary search tree also supports, for example, field-programmable gate array synthesis [72]. Complete details can be found in <https://github.com/darnstrom/pdaqp>.

CODE GENERATION

DSLs for Optimization

When solving a problem instance, DSLs perform a sequence of three steps. First, the DSL transforms the user-defined problem into a form accepted by a standard or canonical solver. For example, a constraint like

$$0 \leq x \leq 1$$

for $x \in \mathbf{R}$ is translated to

$$Ax \leq b, \quad A = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

as it appears in a canonical form like (2). In a second step, a canonical solver is called to solve the canonical problem. Ultimately, a solution for the user-defined problem is retrieved from the solution returned by the canonical solver.

Typically, these three steps are performed every time a problem instance is solved. We call such systems *parser solvers*. When dealing with a parametric QP, whose structure does not change between solves, repeated parsing, that is, discovering how to reduce the problem to canonical form, is unnecessary and usually inefficient.

Code Generation for Explicitly Solving QPs

Consider an application in which we solve many instances of a specific parametric QP, possibly in an embedded system with hard real-time constraints. For such applications, a *code generator* makes more sense.

As illustrated in Figure 1, a code generator to explicitly solve QPs takes as input the parametric QP and generates source code that is tailored to explicitly solve instances of that parametric QP. The source code is then compiled into an efficient custom solver, which offers several advantages over parser solvers. First, by exploiting the parametric structure and caching canonicalization, the compiled solver becomes faster. Second, the compiled solver can be deployed on embedded systems, satisfying rules for safety-critical code [55].

We extend the CVXPYgen open source code generator [57] to generate code to explicitly solve QPs. The QP is modeled with CVXPY before CVXPYgen generates library-, allocation-, and division-free code to translate the user-defined problem into a canonical form (in this case, that of PDAQP) for explicitly solving QPs. For details, see “Integration of CVXPYgen and PDAQP” later. We made several modifications to accommodate for canonicalization to an explicit QP solver, including the extraction of parameter limits, eliminating fixed and redundant parameters, and letting the user select a subset of variables to compute at

every solve. The latter is useful, for example, for model predictive control, where out of the optimal trajectory of inputs and states, only the first input is used [8], [27]. The open source code and full documentation for CVXPYgen and its explicit solve feature are available at <https://github.com/cvxgrp/cvxpygen>.

Disciplined Parametrized Programming

We require the QP to be modeled in CVXPY using disciplined parametrized programming (DPP) [73]. The DPP rules mildly restrict how parameters may enter the objective and constraints. Generally speaking, if the parameters enter all expressions in an affine way, then the problem is DPP compliant. Details on the DPP rules can be found at <https://www.cvxpy.org>.

Canonicalization

When a problem is modeled according to DPP, parameter canonicalization and solution retrieval are affine mappings

$$\theta = C\theta^{\text{user}} + c, \quad x^{\text{user}} = Rx + r$$

where θ^{user} and x^{user} are the user-defined parameter and variable, respectively. The matrices C and R are typically very sparse. The retrieval matrix R is usually wide, that is, there are more canonical variables than user-defined variables since the canonicalization step introduces auxiliary variables [74]. CVXPYgen generates code for the respective sparse matrix-vector multiplications.

Parameter Constraints

The user may specify constraints on parameters. Suppose that the values of the user-defined parameter θ^{user} lie in the set Θ^{user} . Similar to how the parameter θ^{user} itself is canonicalized, the new version of CVXPYgen also translates the parameter constraints Θ^{user} to the canonical set of possible parameters Θ .

Limitations

In PDAQP, the number of inequality constraints m is limited to 1,024, so the regions can be efficiently represented as a bit string. If K or the size of the data in the explicit solver exceeds a given limit, the offline phase is terminated with a warning.

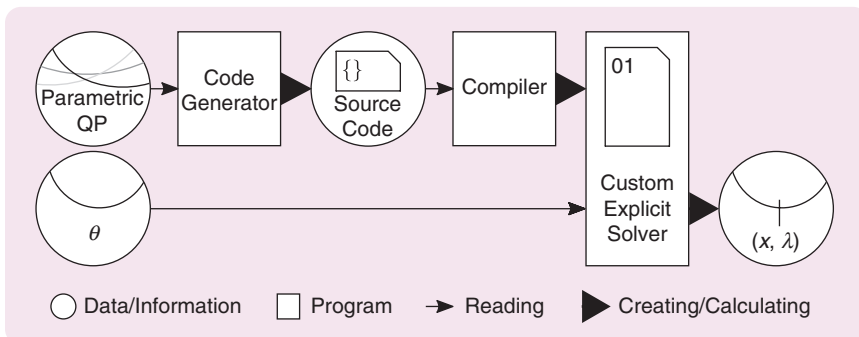


FIGURE 1 Code generation for explicitly solving a parametric QP.

HELLO WORLD

Here, we present a simple example to illustrate how explicit solver code is generated. Consider the parametric QP

$$\begin{aligned} &\text{minimize} \quad \|X\beta + v\mathbf{1} - y\|_2^2 \\ &\text{subject to} \quad \beta \geq 0 \end{aligned} \quad (8)$$

where $\beta \in \mathbf{R}^d$ and $v \in \mathbf{R}$ are the variables, $\theta^{\text{user}} = y \in \mathbf{R}^p$ is the parameter with $\Theta^{\text{user}} = \{y \mid l \leq y \leq u\}$, and

Integration of CVXPYgen and PDAQP

The code generator CVXPYgen is designed with the same modular architecture as the modeling language CVXPY itself. Problem reduction to a standard, that is, canonical, form and solution retrieval from the canonical form are performed by CVXPY, and a canonical solver is invoked in between. Similarly, CVXPYgen generates custom C code that computes the canonicalization and retrieval efficiently and calls the canonical solver's code generation method that generates a custom C implementation of the canonical solver.

Parameter Treatment

When interfacing with the explicit PDAQP solver, CVXPYgen adds several preprocessing steps beyond the problem reductions already performed by CVXPY.

- *Parameter limits:* CVXPY constraints that only limit the parameter θ (but have no effect on variables) are removed from the set of constraints and translated to the set of allowable parameter values Θ .
- *Eliminating parameters:* Fixed parameters are identified by inspecting Θ , and redundant parameters are determined by identifying zero columns in U and V . In both cases, the parameters are eliminated by slicing U and V .

This way, numerical redundancies and degeneracies are prevented before the problem is handled by PDAQP.

Variable Treatment

The canonical version of a problem typically includes auxiliary variables whose values are uninteresting to the user. (For example, the expression $\|Ax - b\|_2^2$ with variable x might be canonicalized to $\|t\|_2^2$ with constraint $t = Ax - b$, where t serves as an auxiliary variable.) CVXPYgen indicates to PDAQP which canonical variables correspond to the user-defined ones, and PDAQP generates an explicit solver for only those by slicing the respective rows of (F_k, g_k) .

In addition, the user may specify that only a slice of a variable should be computed. For example, when only the first column of a matrix of control inputs U should be computed, the user may specify `solver_opts={'stored_vars': U[:, 0]}` when calling `generate_code` as in the last line of Figure 2.

Similarly, the computation of the dual variables can be toggled on or off by setting `{'dual': True}` or `{'dual': False}`, respectively, in `solver_opts`. A comprehensive list of options is available at <https://github.com/cvxgrp/cvxpygen>.

$\mathbf{1}$ denotes the vector with all entries equal to one. The bounds $l \in \mathbb{R}^p$ and $u \in \mathbb{R}^p$ and the matrix $X \in \mathbb{R}^{p \times d}$ are given data.

Modeling and Code Generation

The problem can be formulated in CVXPY, as shown in Figure 2, up to line 11. Note that in line 10, we specify Θ^{user} with standard CVXPY constraints. We generate the explicit solver in line 13.

C Interface

Figure 3 shows how the generated explicit solver can be used in C. In line 7, the first entry of the parameter y is

updated to the value 1.2. The function `cpg_update_y` ensures that the parameter values are within their prespecified limits (otherwise, it maps the given value back onto Θ). In line 8, the problem is explicitly solved with the function `cpg_solve`. In lines 9 and 10, respectively, the first entry of the optimal coefficients β and the optimal v is read and printed. In line 11, the resulting objective value is calculated and printed. Note that the calculation of the objective value is kept in a separate function from the solve function for maximal efficiency.

Figure 4 shows the C structs that store the result. In the previous example, we access the primal solution `beta`

```
1 import cvxpy as cp
2 from cvxpygen import cpg
3
4 d, p, X, l, u = ...
5 beta = cp.Variable(d, name='beta')
6 v = cp.Variable(name='v')
7 y = cp.Parameter(p, name='y')
8
9 obj = cp.Minimize(cp.sum_squares(X@beta+v-y))
10 constr = [beta>=0, l<=y, y<=u]
11 prob = cp.Problem(obj, constr)
12
13 cpg.generate_code(prob, solver='explicit')
```

FIGURE 2 Generating an explicit solver with CVXPYgen.

```
1 #include <stdio.h>
2 #include "cpg_workspace.h"
3 #include "cpg_solve.h"
4
5 int main(int argc, char *argv[]){
6
7     cpg_update_y(0, 1.2);
8     cpg_solve();
9     printf("%f\n", CPG_Result.prim->beta[0]);
10    printf("%f\n", CPG_Result.prim->v);
11    printf("%f\n", cpg_obj());
12
13    return 0;
14
15 }
```

FIGURE 3 Using the explicit solver in C.

and v via the `prim` field of the result struct `CPG_Result`. The dual variables in `CPG_Dual_t` are named according to the index in the CVXPY constraint list.

CVXPY Interface

We consider a small instance of the parametric QP (8) with $d = 2$, $p = 3$, the entries of X generated independent identically distributed (IID) from $\mathcal{N}(0, 1)$, $l = 0$, and $u = 1$. We randomly assign the entries of y between 0 and 1 and solve the problem three times: with CVXPY using the iterative OSQP solver [21], with CVXPYgen using OSQP, and with CVXPYgen using the explicit PDAQP solver.

Figure 5 shows the comparison, demonstrating how the explicit solver can be used via its auto-generated CVXPY interface. Starting at line 15, we show that the primal and dual solutions and the objective values agree closely.

APPLICATIONS

In this section, we report timing and code size details for several typical application examples. In each case, we compare CVXPYgen using the explicit PDAQP solver to CVXPYgen using the iterative OSQP solver and standard CVXPY using OSQP. We also provide a comparison to the state-of-the-art multiparametric programming solver MPT3 [42], which we use via the DSL YALMIP [62]. When using CVXPYgen with the OSQP solver, we use OSQP's code-generation feature, which caches the factorization of the KKT system to accelerate solving [58]. When using OSQP in CVXPY or CVXPYgen, we set both the relative and absolute tolerances to 10^{-4} (the default in CVXPY). We run the experiments on an Apple M1 Pro, compiling with GCC at optimization level 3. For iterative and explicit code generation, we report solve times in C, and we report the overall time when solving from Python via the CVXPY interface generated by CVXPYgen. We also give the time it takes to generate and compile the code.

Monotone Regression

We consider the monotone regression problem

$$\begin{aligned} & \text{minimize} \quad \|Ax - b\|_2^2 \\ & \text{subject to} \quad x_1 \leq x_2 \leq \dots \leq x_d \end{aligned}$$

```
1 typedef struct {
2   cpg_float *beta; // primal variable beta
3   cpg_float v; // primal variable v
4 } CPG_Prim_t;
5
6 typedef struct {
7   cpg_float *d0; // dual variable d0
8 } CPG_Dual_t;
9
10 typedef struct {
11   CPG_Prim_t *prim; // primal solution
12   CPG_Dual_t *dual; // dual solution
13 } CPG_Result_t;
```

FIGURE 4 The data structure of the explicit solver result.

where $x \in \mathbf{R}^d$ is the variable, and $\theta^{\text{user}} = b \in \mathbf{R}^q$ is the parameter, with $\Theta^{\text{user}} = [-1, 1]^q$. The matrix $A \in \mathbf{R}^{q \times d}$ is given. (This is called *monotone regression* since the components x_i are constrained to be monotonically nondecreasing. For example, when estimating the degradation of a machine over time, x_i might represent the degradation at time i due to wear and tear, which can only increase over time [75].)

Problem Instances

We consider $d = 5$ and $q = 10$. (In the previous example, this might correspond to estimating the degradation over five months, given biweekly measurements.) The entries of A are generated IID from $\mathcal{N}(0, 1)$, and we generate 100 problem instances where b is sampled uniformly from Θ^{user} .

Results

We obtain $n = 15$ variables, $m = d - 1 = 4$ inequality constraints, and $p = q = 10$ parameters. [In this case, the canonicalization step introduced $n - d = 10$ auxiliary variables. CVXPYgen tracks the indices of the auxiliary variables and flags to PDAQP that they should not be

```
1 from code_osqp.cpg_solver import cpg_solve
2 prob.register_solve('gen_OSQP', cpg_solve)
3
4 from code_explicit.cpg_solver import cpg_solve
5 prob.register_solve('gen_explicit', cpg_solve)
6
7 def print_result():
8     print(f'v: {v.value}')
9     print(f'beta: {beta.value}')
10    print(f'dual: {constr[0].dual_value}')
11    print(f'obj: {obj.value}')
12
13 y.value = [0.6, 0.8, 0.2]
14
15 prob.solve(solver='OSQP')
16 print_result()
17
18 # v: 0.916741
19 # beta: [0.000000 0.288547]
20 # dual: [0.750370 0.000000]
21 # obj: 0.059628
22
23 prob.solve(method='gen_OSQP')
24 print_result()
25
26 # v: 0.916741
27 # beta: [0.000000 0.288547]
28 # dual: [0.750370 0.000000]
29 # obj: 0.059628
30
31 prob.solve(method='gen_explicit')
32 print_result()
33
34 # v: 0.916740
35 # beta: [0.000000 0.288546]
36 # dual: [0.750370 0.000000]
37 # obj: 0.059628
```

FIGURE 5 Using the explicit solver in CVXPY.

Generally speaking, if the parameters enter all expressions in an affine way, then the problem is DPP compliant.

computed when solving the parametric problem. We find $K = 16$ regions (which equals 2^m , the maximum possible number of active sets)]. Table 1 shows the average solve times and binary sizes. The column titled “Setup” gives the time it takes to set up the Python-Julia interface with PDAQP and the time to set up MATLAB with MPT3, respectively. Compared to code generation with the iterative OSQP solver, we observe a speed up of more than two orders of magnitude when solving in C. YALMIP and MPT3 give similar solve times in C (after a much longer generation time). When using the Python wrapper (and incurring the associated overhead), the speed up compared to CVXPY is about a factor of 50, which may be useful for prototyping or nonembedded applications.

Power Management

With the increasing availability of alternative energy sources, real-time optimal power management becomes crucial for residential or office buildings, factories, or data centers [76]. We consider the setting where a nonnegative electrical power load L is served by a photovoltaic (PV) solar panel system, a storage battery, and a grid connection [77]. We denote solar power by s , battery power by b , and grid power by g . These three power sources supply the load, so we have

$$L = s + b + g.$$

The PV power satisfies $0 \leq s \leq S$, where $S \geq 0$ is the PV power available. The battery power satisfies $-C \leq b \leq D$, where $D > 0$ is the maximum possible discharge power, and $C > 0$ is the maximum possible charge power. The power of the grid satisfies $g \geq 0$, that is, we cannot sell power back to the grid. The (positive) price of the grid power is P , so the cost of the grid is Pgh , where h is the duration of a period of time during which we keep the power values constant.

The battery state of charge at the beginning of the time period is denoted q and satisfies $0 \leq q \leq Q$, where Q is the battery capacity. At the beginning of the next time period, the battery charge is $q^+ = q - hb$. We must have $0 \leq q^+ \leq Q$.

We take the cost function

$$Pgh + \alpha(q^+ - q^{\text{tar}})^2 + \beta b^2$$

where α and β are given and positive, and q^{tar} is a given target battery charge value.

To choose the powers, we solve the QP

$$\begin{aligned} &\text{minimize} && Pgh + \alpha(q^+ - q^{\text{tar}})^2 + \beta b^2 \\ &\text{subject to} && L = s + b + g, \\ &&& 0 \leq s \leq S, \quad -C \leq b \leq D, \quad g \geq 0, \\ &&& q^+ = q - hb, \quad 0 \leq q^+ \leq Q \end{aligned}$$

where s, b, g , and q^+ are the variables, and $\theta^{\text{user}} = (L, S, P, q)$ are parameters. The remaining constants, $C, D, h, Q, q^{\text{tar}}, \alpha$, and β , are known. We take

$$\Theta^{\text{user}} = [0, 1] \times [0, 0.5] \times [1, 2] \times [0, Q].$$

Problem Instances

We set $C = D = 1$, $h = 0.05$, $Q = 1$, $q^{\text{tar}} = 0.5$, and $\alpha = \beta = 0.1$. We generate 100 problem instances where (L, S, P, q) is sampled uniformly from Θ^{user} .

Results

After canonicalization, there are $n = 5$ variables (including one auxiliary variable), $m = 7$ inequality constraints, and $p = 4$ parameters. We find $K = 5$ regions. Table 2 shows the average solve times and binary sizes. Compared to iterative solver generation, we observe a similar speedup of the solve times as in the previous example. Here, the canonical solve time with PDAQP is so small (about $0.1 \mu\text{s}$, as with MPT3), so canonicalization accounts for about half of the total solve time, and the solver generated by YALMIP and MPT3 is overall faster.

Model Predictive Control

We consider the linear dynamical system [7]

$$z_{t+1} = Az_t + Bu_t, \quad t = 0, 1, \dots, H-1$$

TABLE 1 Timing and binary sizes for the monotone regression problem.

	Solve (Python)	Solve (C)	Setup	Gen. + compile	Gen.	Binary size
CVXPY	0.6089 ms	—	—	—	—	—
CVXPYgen OSQP	0.1764 ms	0.0253 ms	—	6.6 s	0.1 s	81 KB
CVXPYgen explicit	0.0127 ms	0.0002 ms	9.4 s	1.7 s	0.1 s	23 KB
YALMIP + MPT3	—	0.0002 ms	38.1 s	4.1 s	3.5 s	21 KB

Gen.: generate.

Compared to the iterative solver, our explicit solver is faster by a factor of more than 200 and about as fast as the solver generated by YALMIP and MPT3.

where $z_t \in \mathbf{R}^{n_z}$ is the state, and $u_t \in \mathbf{R}^{n_u}$ is the input, which must satisfy $\|u_t\|_\infty \leq 1$. The matrices $A \in \mathbf{R}^{n_z \times n_z}$ and $B \in \mathbf{R}^{n_z \times n_u}$ are given. We solve the model predictive control problem [3], [8]

$$\begin{aligned} & \text{minimize} \quad z_H^T P z_H + \sum_{t=0}^{H-1} (z_t^T Q z_t + u_t^T R u_t) \\ & \text{subject to} \quad z_{t+1} = A z_t + B u_t, \quad t = 0, \dots, H-1, \\ & \quad \quad \quad \|u_t\|_\infty \leq 1, \quad t = 0, \dots, H-1, \\ & \quad \quad \quad z_0 = z^{\text{init}} \end{aligned}$$

where $z_0, \dots, z_H$ and $u_0, \dots, u_{H-1}$ are the variables and $\theta^{\text{user}} = z^{\text{init}}$ is the parameter. We take $\Theta^{\text{user}} = [-1, 1]^{n_z}$. The objective matrices $P \in \mathbf{S}_{++}^{n_z}$, $Q \in \mathbf{S}_{++}^{n_z}$, and $R \in \mathbf{S}_{++}^{n_u}$ (along with A and B) are given.

This is a typical case where one may use our feature for selecting which variables should be computed after the explicit solver is generated. The standard choice would be to select the initial control input u_0 (as we did here), which would be applied to the system, before the problem is resolved with the next measurement.

Problem Instances

We consider $n_z = 6$ states, $n_u = 1$ input, and a horizon length of $H = 5$. We construct A by sampling its diagonal entries from $\mathcal{N}(0, 1)$ and its off-diagonal entries IID from

$\mathcal{N}(0, 0.01)$ before scaling the whole matrix so that A has spectral radius 1. The entries in the input matrix B are sampled IID from $\mathcal{N}(0, 0.001)$. We set the controller weights to $Q = I$ and $R = 0.1I$, and we compute P as the solution to the algebraic Riccati equation associated with the infinite-horizon problem [78]. We generate 100 problem instances where the entries of z^{init} are generated uniformly from Θ^{user} .

Results

After canonicalization, we have $n = 77$ variables (36 of which are auxiliary variables), $m = 10$ inequality constraints, and $p = n_z = 6$ parameters. We find $K = 63$ regions. (For the reader familiar with control, we can view this as 63 linear controllers, which we select based on the value of the current state.) Table 3 shows the average solve times and binary sizes. Compared to the iterative solver, our explicit solver is faster by a factor of more than 200 and about as fast as the solver generated by YALMIP and MPT3. Compared to the latter, our solver generation is orders of magnitude faster, and the executable has about half the size.

Portfolio Optimization

We construct a financial portfolio consisting of holdings in N assets [13], [16], [17]. We represent the holdings relative to the total (positive) portfolio value in terms of nonnegative weights $w \in \mathbf{R}_+^N$, where $\mathbf{1}^T w = 1$, with w_i being the fraction of the (positive) total portfolio value invested in asset i . With estimated mean annualized asset returns $\mu \in \mathbf{R}^N$ [17], the estimated mean annualized portfolio return is $\mu^T w$. The variance or risk of the portfolio return is $w^T \Sigma w$, where $\Sigma \in \mathbf{S}_{++}^N$ is an estimate of the covariance matrix of annualized asset returns. Our objective is to maximize the risk-adjusted expected annualized return

$$\mu^T w - \gamma w^T \Sigma w$$

where $\gamma > 0$ is the risk-aversion factor. To find the portfolio, we solve the Markowitz problem [14], [18]

$$\begin{aligned} & \text{maximize} \quad \mu^T w - \gamma w^T \Sigma w \\ & \text{subject to} \quad \mathbf{1}^T w = 1, \quad w \geq 0 \end{aligned}$$

where the portfolio weights $w \in \mathbf{R}^N$ are the variable, and the parameter is

TABLE 2 Timing and binary sizes for the power management problem.

	Solve (Python)	Solve (C)	Setup	Gen. + compile	Gen.	Binary size
CVXPY	0.6564 ms	—	—	—	—	—
CVXPYgen OSQP	0.1261 ms	0.0122 ms	—	6.8 s	0.1 s	73 KB
CVXPYgen explicit	0.0207 ms	0.0003 ms	8.6 s	2.6 s	0.1 s	20 KB
YALMIP + MPT3	—	0.0001 ms	38.1 s	1.6 s	0.6 s	9 KB

TABLE 3 Timing and binary sizes for the model predictive control problem.

	Solve (Python)	Solve (C)	Setup	Gen. + compile	Gen.	Binary size
CVXPY	1.102 ms	—	—	—	—	—
CVXPYgen OSQP	0.875 ms	0.063 ms	—	6.8 s	0.1 s	109 KB
CVXPYgen explicit	0.025 ms	0.0003 ms	8.2 s	2.5 s	0.9 s	65 KB
YALMIP + MPT3	—	0.0002 ms	38.1 s	23.1 s	21.5 s	115 KB

$\theta^{\text{user}} = \mu$ with $\Theta^{\text{user}} = [-1, 1]^N$. This means that we expect no annualized returns beyond $\pm 100\%$. The covariance matrix $\Sigma \in \mathbf{S}_+^N$ and the risk-aversion factor $\gamma > 0$ are given. Especially in high-frequency trading, being able to repeatedly solve this problem at high speed is crucial [50].

Problem Instances

We take $N = 7$ assets. To obtain data, we choose the seven stocks with the highest market capitalization as of 1 January 2017, listed in Table 4.

We compute Σ by first taking the sample covariance of the daily returns of the seven assets for 2017 and 2018 and then annualizing the result. We choose $\gamma = 2$. We generate 250 problem instances where μ is the one-year trailing average of returns for 250 trading days in 2019.

Results

We have $n = N = 7$ variables, $m = N = 7$ inequality constraints, and $p = N = 7$ parameters. We find $K = 127$ regions, only one fewer than the maximum 2^m potential combinations of investing in an asset or not, since the constraint $\mathbf{1}^T w = 1$ prevents $w = 0$ (not investing in any asset).

Table 5 shows the average solve times and binary sizes. In this case, due to the relatively large number of regions K , the binary size is about tripled compared to that of the iterative solver. YALMIP and MPT3 failed to generate a binary search tree within 1 h, so the generated solver searches the regions in linear time and is therefore about five times slower than the solver generated with CVXPYgen and PDAQP. (Note that YALMIP and MPT3 show a faster generation time because no search tree is computed.) Table 6 shows the binary sizes when using CVXPYgen with different FP precision settings for storing the explicit solution.

CONCLUSION

This article has focused on the use of code generation to generate an explicit solver for a class of parametrized convex optimization problems. The chosen code generator is CVXPYgen, to which a new functionality is added that generates this explicit solver. The user can prototype a problem in CVXPY, with code that closely resembles the mathematical formulation and with convenient names for multiple variables and parameters, using a generic iterative solver; changing a single code-generation option will generate an explicit solver for the parametrized problem. For typical small problems in various application domains, our numerical experiments show that the generated explicit solvers exhibit solve times below 1 ms, up to 200 times faster than with an iterative solver. Our explicit solvers scale better than those of YALMIP and MPT3. While it is hard to beat the latter on our smallest

TABLE 4 Stocks used in the portfolio optimization problem.

Ticker Symbol	Company
AAPL	Apple
AMZN	Amazon
BRK.A	Berkshire Hathaway
FB (now META)	Facebook (now Meta)
GOOGL	Alphabet
MSFT	Microsoft
XOM	ExxonMobil

TABLE 5 Timing and binary sizes for the portfolio optimization problem. *without binary search tree.

	Solve (Python)	Solve (C)	Setup	Gen. + compile	Gen.	Binary size
CVXPY	0.5441 ms	—	—	—	—	—
CVXPYgen OSQP	0.0502 ms	0.0012 ms	—	6.9 s	0.1 s	76 KB
CVXPYgen explicit	0.0113 ms	0.0004 ms	9.6 s	8.8 s	6.9 s	244 KB
YALMIP + MPT3	—	0.0023 ms	38.1 s	4.1 s	3 s*	103 KB

TABLE 6 Binary sizes with different precision settings.

	Binary Size
Double	285 KB
Float (default)	244 KB
fp16	224 KB

example problem, our tool succeeds in generating binary search trees for the largest problem (whereas YALMIP and MPT3 failed to do so) and is therefore much faster in that case.

AUTHOR INFORMATION

Maximilian Schaller (mschall@stanford.edu) received his M.Sc. degree in robotics, systems, and control from ETH Zurich and his B.Sc. degree in mechanical and process Engineering from TU Darmstadt. He is a Ph.D. candidate at Stanford University, Stanford, CA 94305 USA, where he works with Prof. Stephen P. Boyd on modeling and solving convex optimization problems. Before attending Stanford, he was a consultant and data scientist at McKinsey, where he worked on quantitative projects in the automotive and energy sectors.

Daniel Arnström is a senior researcher at Ericsson Research in the Decision and Control Group, 16440 Kista, Sweden. His work focuses on the intersection of reliability and performance, specifically regarding embedded

optimization and real-time model predictive control. He investigates the reliability of optimization-based controllers across computational, safety, security, and communication dimensions.

Alberto Bemporad received his M.S. degree cum laude in electrical engineering (1993) and his Ph.D. in control engineering (1997) from the University of Florence, Italy. He held research positions at Washington University in St. Louis (1996–1997) and ETH Zurich (1997–2002). He served at the University of Siena (1999–2009) and at the University of Trento (2010–2011). Since 2011, he has been a full professor at the IMT School for Advanced Studies Lucca, 55100 Lucca, Italy, where he was the director from 2012 to 2015. He cofounded ODYS S.r.l. in 2011, a company specializing in industrial model predictive control systems, and has held visiting appointments at Stanford University, the University of Michigan, and Zhejiang University. He has authored more than 400 publications and 21 patents in model predictive control, hybrid systems, optimization, and automotive control and co-developed the Model Predictive Control Toolbox (MathWorks) for MATLAB. He was an associate editor of *IEEE Transactions on Automatic Control* during 2001–2004 and chair of the IEEE CSS Technical Committee on Hybrid Systems during 2002–2010. His awards include the IFAC High-Impact Paper Award (2011–2014), IEEE CSS Transition to Practice Award (2019), SAE Environmental Excellence in Transportation Award (2021), the Beale–Orchard-Hays Prize (2024), and an ERC Advanced Grant (2024). He is a Fellow of IEEE (since 2010) and IFAC (since 2025).

Stephen P. Boyd received an AB degree in mathematics, summa cum laude, from Harvard University in 1980, and a Ph.D. degree in electrical engineering and computer science from U. C. Berkeley in 1985. He is the Samsung Professor of Engineering, Professor of Electrical Engineering, and a member of the Institute for Mathematical and Computational and Engineering, Stanford University, Stanford, CA 94305 USA. He holds honorary Ph.D.s from Royal Institute of Technology (KTH), Stockholm, and from the Catholic University of Louvain (UCL). His research interests include convex programming applications in control, signal processing, and finance. He has been invited to deliver more than 100 plenary and keynote lectures at major conferences in control, optimization, signal processing, and machine learning. He has coauthored books on control, optimization, and applied linear algebra, including *Convex Optimization*. His group has developed several widely used open source software packages for convex optimization, including CVXPY. He is a Life Fellow of the IEEE; a Fellow of SIAM, INFORMS, IFAC, and ; a Distinguished Lecturer of the IEEE Control Systems Society; a member of the U.S. National Academy of Engineering (NAE); a foreign member of the Chinese Academy of Engineering (CAE); and a foreign member of the National Academy of Engineering of Korea (NAEK).

REFERENCES

- [1] F. Borrelli, A. Bemporad, and M. Morari, *Predictive Control for Linear and Hybrid Systems*. Cambridge, U.K.: Cambridge Univ. Press, 2017.
- [2] J. Rawlings, D. Mayne, M. Diehl et al., *Model Predictive Control: Theory, Computation, and Design*. Madison, WI, USA: Nob Hill Publ., 2017.
- [3] B. Kouvaritakis and M. Cannon, *Model Predictive Control*. London, U.K.: Springer-Verlag, 2016.
- [4] A. Keshavarz and S. Boyd, "Quadratic approximate dynamic programming for input-affine systems," *Int. J. Robust Nonlinear Control*, vol. 24, no. 3, pp. 432–449, 2014, doi: [10.1002/rnc.2894](https://doi.org/10.1002/rnc.2894).
- [5] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," *IEEE Trans. Control Syst. Technol.*, vol. 18, no. 2, pp. 267–278, 2010, doi: [10.1109/TCST.2009.2017934](https://doi.org/10.1109/TCST.2009.2017934).
- [6] A. Bemporad and M. Morari, "Robust model predictive control: A survey," in *Robustness in Identification and Control*. London, U.K.: Springer-Verlag, 2007, pp. 207–226.
- [7] S. Boyd and C. Barratt, *Linear Controller Design: Limits of Performance*. vol. 78, University Park, PA, USA: Citeseer, 1991.
- [8] C. Garcia, D. Prett, and M. Morari, "Model predictive control: Theory and practice – a survey," *Automatica*, vol. 25, no. 3, pp. 335–348, 1989, doi: [10.1016/0005-1098\(89\)90002-2](https://doi.org/10.1016/0005-1098(89)90002-2).
- [9] A. Chambolle and T. Pock, "An introduction to continuous optimization for imaging," *Acta Numerica*, vol. 25, pp. 161–319, May 2016, doi: [10.1017/S096249291600009X](https://doi.org/10.1017/S096249291600009X).
- [10] P. Combettes and J. Pesquet, "Proximal splitting methods in signal processing," *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*, New York, NY, USA: Springer-Verlag, 2011, pp. 185–212, doi: [10.1007/978-1-4419-9569-8_10](https://doi.org/10.1007/978-1-4419-9569-8_10).
- [11] J. Mattingley and S. Boyd, "Real-time convex optimization in signal processing," *IEEE Signal Process. Mag.*, vol. 27, no. 3, pp. 50–61, 2010, doi: [10.1109/MSP.2010.936020](https://doi.org/10.1109/MSP.2010.936020).
- [12] M. Zibulevsky and M. Elad, "L1-L2 optimization in signal and image processing," *IEEE Signal Process. Mag.*, vol. 27, no. 3, pp. 76–88, 2010, doi: [10.1109/MSP.2010.936023](https://doi.org/10.1109/MSP.2010.936023).
- [13] D. Palomar, *Portfolio Optimization: Theory and Application*. Cambridge, U.K.: Cambridge Univ. Press, 2025.
- [14] S. Boyd, K. Johansson, R. Kahn, P. Schiele, and T. Schmelzer, "Markowitz portfolio construction at seventy," *J. Portfolio Manage.*, vol. 50, no. 8, pp. 117–160, 2024, doi: [10.3905/jpm.2024.50.8.117](https://doi.org/10.3905/jpm.2024.50.8.117).
- [15] S. Boyd et al., "Multi-period trading via convex optimization," *Found. Trends Optim.*, vol. 3, no. 1, pp. 1–76, 2017, doi: [10.1561/24000000023](https://doi.org/10.1561/24000000023).
- [16] R. Narang, *Inside the Black Box: A Simple Guide to Quantitative and High-Frequency Trading*. Hoboken, NJ, USA: Wiley, 2013.
- [17] R. Grinold and R. Kahn, *Active Portfolio Management*. New York, NY, USA: McGraw Hill, 2000.
- [18] H. Markowitz, "Portfolio selection," *J. Finance*, vol. 7, no. 1, pp. 77–91, 1952, doi: [10.2307/2975974](https://doi.org/10.2307/2975974).
- [19] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.
- [20] P. Goulart and Y. Chen, "Clarabel: An interior-point solver for conic programs with quadratic objectives," 2024, *arXiv:2405.12762*.
- [21] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: An operator splitting solver for quadratic programs," *Math. Program. Comput.*, vol. 12, no. 4, pp. 637–672, 2020, doi: [10.1007/s12532-020-00179-2](https://doi.org/10.1007/s12532-020-00179-2).
- [22] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd, "Conic optimization via operator splitting and homogeneous self-dual embedding," *J. Optim. Theory Appl.*, vol. 169, no. 3, pp. 1042–1068, 2016, doi: [10.1007/s10957-016-0892-3](https://doi.org/10.1007/s10957-016-0892-3).
- [23] A. Domahidi, E. Chu, and S. Boyd, "ECOS: An SOCP solver for embedded systems," in *Proc. Eur. Control Conf. (ECC)*, 2013, pp. 3071–3076.
- [24] O. Mangasarian and J. Rosen, "Inequalities for stochastic nonlinear programming problems," *Oper. Res.*, vol. 12, no. 1, pp. 143–154, 1964, doi: [10.1287/opre.12.1.143](https://doi.org/10.1287/opre.12.1.143).
- [25] A. Fiacco, *Introduction to Sensitivity and Stability Analysis in Nonlinear Programming*. London, U.K.: Academic Press, 1983.
- [26] T. Gal and J. Nedoma, "Multiparametric linear programming," *Manage. Sci.*, vol. 18, no. 7, pp. 406–422, 1972, doi: [10.1287/mnsc.18.7.406](https://doi.org/10.1287/mnsc.18.7.406).
- [27] A. Bemporad, M. Morari, V. Dua, and E. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, pp. 3–20, 2002, doi: [10.1016/S0005-1098\(01\)00174-1](https://doi.org/10.1016/S0005-1098(01)00174-1).
- [28] A. Bemporad, F. Borrelli, and M. Morari, "Model predictive control based on linear programming – the explicit solution," *IEEE Trans. Autom. Control*, vol. 47, no. 12, pp. 1974–1985, 2002, doi: [10.1109/TAC.2002.805688](https://doi.org/10.1109/TAC.2002.805688).

- [29] P. Tøndel, T. Johansen, and A. Bemporad, "An algorithm for multi-parametric quadratic programming and explicit MPC solutions," *Automatica*, vol. 39, no. 3, pp. 489–497, 2003, doi: [10.1016/S0005-1098\(02\)00250-9](https://doi.org/10.1016/S0005-1098(02)00250-9).
- [30] P. Patrinos and H. Sarimveis, "A new algorithm for solving convex parametric quadratic programs based on graphical derivatives of solution mappings," *Automatica*, vol. 46, no. 9, pp. 1405–1418, 2010, doi: [10.1016/j.automatica.2010.06.008](https://doi.org/10.1016/j.automatica.2010.06.008).
- [31] A. Gupta, S. Bhartiya, and P. Nataraj, "A novel approach to multiparametric quadratic programming," *Automatica*, vol. 47, no. 9, pp. 2112–2117, 2011, doi: [10.1016/j.automatica.2011.06.019](https://doi.org/10.1016/j.automatica.2011.06.019).
- [32] F. Borrelli, A. Bemporad, and M. Morari, "Geometric algorithm for multiparametric linear programming," *J. Optim. Theory Appl.*, vol. 118, no. 3, pp. 515–540, 2003, doi: [10.1023/B:JOTA.0000004869.66331.5c](https://doi.org/10.1023/B:JOTA.0000004869.66331.5c).
- [33] G. Cimini and A. Bemporad, "Exact complexity certification of active-set methods for quadratic programming," *IEEE Trans. Autom. Control*, vol. 62, no. 12, pp. 6094–6109, Dec. 2017, doi: [10.1109/TAC.2017.2696742](https://doi.org/10.1109/TAC.2017.2696742).
- [34] A. Bemporad and C. Filippi, "Suboptimal explicit receding horizon control via approximate multiparametric quadratic programming," *J. Optim. Theory Appl.*, vol. 117, no. 1, pp. 9–38, 2003, doi: [10.1023/A:1023696221899](https://doi.org/10.1023/A:1023696221899).
- [35] T. Johansen and A. Grancharova, "Approximate explicit constrained linear model predictive control via orthogonal search tree," *IEEE Trans. Autom. Control*, vol. 48, no. 5, pp. 810–815, May 2003, doi: [10.1109/TAC.2003.811259](https://doi.org/10.1109/TAC.2003.811259).
- [36] A. Bemporad and C. Filippi, "An algorithm for approximate multiparametric convex programming," *Comput. Optim. Appl.*, vol. 35, no. 1, pp. 87–108, 2006, doi: [10.1007/s10589-006-6447-z](https://doi.org/10.1007/s10589-006-6447-z).
- [37] C. Jones and M. Morari, "Multiparametric linear complementarity problems," in *Proc. 45th IEEE Conf. Decis. Control*, 2006, pp. 5687–5692, doi: [10.1109/CDC.2006.377797](https://doi.org/10.1109/CDC.2006.377797).
- [38] T. Johansen, W. Jackson, R. Schreiber, and P. Tøndel, "Hardware synthesis of explicit model predictive controllers," *IEEE Trans. Control Syst. Technol.*, vol. 15, no. 1, pp. 191–197, Jan. 2007, doi: [10.1109/TCST.2006.883206](https://doi.org/10.1109/TCST.2006.883206).
- [39] A. Bemporad, A. Oliveri, T. Poggi, and M. Storace, "Ultra-fast stabilizing model predictive control via canonical piecewise affine approximations," *IEEE Trans. Autom. Control*, vol. 56, no. 12, pp. 2883–2897, Dec. 2011, doi: [10.1109/TAC.2011.2141410](https://doi.org/10.1109/TAC.2011.2141410).
- [40] A. Ravera et al., "Co-design of a controller and its digital implementation: The MOBY-DIC2 toolbox for embedded model predictive control," *IEEE Trans. Control Syst. Technol.*, vol. 31, no. 6, pp. 2871–2878, Nov. 2023, doi: [10.1109/TCST.2023.3254133](https://doi.org/10.1109/TCST.2023.3254133).
- [41] A. Bemporad, *Hybrid Toolbox - User's Guide*. 2004. Accessed: Jul. 28, 2026. [Online]. Available: <http://cse.lab.imtlucca.it/~bemporad/hybrid/toolbox/>
- [42] M. Herceg, M. Kvasnica, C. N. Jones, and M. Morari, "Multi-parametric toolbox 3.0," in *Proc. Eur. Control Conf. (ECC)*, 2013, pp. 502–510.
- [43] A. Bemporad, "A multiparametric quadratic programming algorithm with polyhedral computations based on nonnegative least squares," *IEEE Trans. Autom. Control*, vol. 60, no. 11, pp. 2892–2903, Nov. 2015, doi: [10.1109/TAC.2015.2417851](https://doi.org/10.1109/TAC.2015.2417851).
- [44] R. Oberdieck, N. Dangelakis, M. Papathanasiou, I. Nascu, and E. Pistikopoulos, "POP – parametric optimization toolbox," *Ind. Eng. Chemistry Res.*, vol. 55, no. 33, pp. 8979–8991, 2016, doi: [10.1021/acs.iecr.6b01913](https://doi.org/10.1021/acs.iecr.6b01913).
- [45] A. Domahidi and J. Jerez, "FORCES professional. Embotech GmbH," 2014 [Online]. Available: <https://embotech.com/forcespro>.
- [46] D. Arnström and D. Axehill, "A high-performance multi-parametric quadratic programming solver," in *Proc. IEEE 63rd Conf. Decis. Control (CDC)*, 2024, pp. 303–308, doi: [10.1109/CDC56724.2024.10886132](https://doi.org/10.1109/CDC56724.2024.10886132).
- [47] P. Tøndel, T. Johansen, and A. Bemporad, "Evaluation of piecewise affine control via binary search tree," *Automatica*, vol. 39, no. 5, pp. 945–950, 2003, doi: [10.1016/S0005-1098\(02\)00308-4](https://doi.org/10.1016/S0005-1098(02)00308-4).
- [48] M. Baotic, F. Borrelli, A. Bemporad, and M. Morari, "Efficient on-line computation of constrained optimal control," *SIAM J. Control Optim.*, vol. 47, no. 5, pp. 2470–2489, 2008, doi: [10.1137/060659314](https://doi.org/10.1137/060659314).
- [49] N. Nguyen, M. Gulan, S. Olaru, and P. Rodriguez-Ayerbe, "Convex lifting: Theory and control applications," *IEEE Trans. Autom. Control*, vol. 63, no. 5, pp. 1243–1258, May 2018, doi: [10.1109/TAC.2017.2737234](https://doi.org/10.1109/TAC.2017.2737234).
- [50] A. Cartea, S. Jaimungal, and J. Penalva, *Algorithmic and High-Frequency Trading*. Cambridge, U.K.: Cambridge Univ. Press, 2015.
- [51] F. Bayat, R. Johansen, and A. Jalali, "Flexible piecewise function evaluation methods based on truncated binary search trees and lattice representation in explicit MPC," *IEEE Trans. Control Syst. Technol.*, vol. 20, no. 3, pp. 632–640, May 2012, doi: [10.1109/TCST.2011.2141134](https://doi.org/10.1109/TCST.2011.2141134).
- [52] P. Julian, A. Desages, and O. Agamennoni, "High-level canonical piecewise linear representation using a simplicial partition," *IEEE Trans. Circuits Syst. I: Fundam. Theory Appl.*, vol. 46, no. 4, pp. 463–480, Apr. 1999, doi: [10.1109/81.754847](https://doi.org/10.1109/81.754847).
- [53] D. Arnström, D. Broman, and D. Axehill, "Exact worst-case execution-time analysis for implicit model predictive control," *IEEE Trans. Autom. Control*, vol. 69, no. 10, pp. 7190–7196, Oct. 2024, doi: [10.1109/TAC.2024.3395521](https://doi.org/10.1109/TAC.2024.3395521).
- [54] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *J. Mach. Learn. Res.*, vol. 17, no. 83, pp. 1–5, 2016.
- [55] G. Holzmann, "The power of 10: Rules for developing safety-critical code," *Computer*, vol. 39, no. 6, pp. 95–99, 2006, doi: [10.1109/MC.2006.212](https://doi.org/10.1109/MC.2006.212).
- [56] J. Mattingley and S. Boyd, "CVXGEN: A code generator for embedded convex optimization," *Optim. Eng.*, vol. 13, no. 1, pp. 1–27, 2012, doi: [10.1007/s11081-011-9176-9](https://doi.org/10.1007/s11081-011-9176-9).
- [57] M. Schaller, G. Banjac, S. Diamond, A. Agrawal, B. Stellato, and S. Boyd, "Embedded code generation with CVXPY," *IEEE Contr. Syst. Lett.*, vol. 6, pp. 2653–2658, 2022, doi: [10.1109/LCSYS.2022.3173209](https://doi.org/10.1109/LCSYS.2022.3173209).
- [58] G. Banjac, B. Stellato, N. Moehle, P. Goulart, A. Bemporad, and S. Boyd, "Embedded code generation using the OSQP solver," in *Proc. IEEE 56th Annu. Conf. Decis. Control*, 2017, pp. 1906–1911.
- [59] G. Chari and B. Açikmeşe, "QOCO: A quadratic objective conic optimizer with custom solver generation," 2025, *arXiv:2503.12658*.
- [60] R. Verschuere et al., "acados – a modular open-source framework for fast embedded optimal control," *Math. Program. Comput.*, vol. 14, no. 1, pp. 147–183, Mar. 2022, doi: [10.1007/s12532-021-00208-8](https://doi.org/10.1007/s12532-021-00208-8).
- [61] L. Blackmore, "Autonomous precision landing of space rockets," in *Frontiers of Engineering: Reports on Leading-Edge Engineering From the 2016 Symposium*, vol. 46. Washington, DC, USA: The Bridge, 2016, pp. 15–20.
- [62] J. Löfberg, "YALMIP: A toolbox for modeling and optimization in MATLAB," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2004, pp. 284–289.
- [63] M. Grant and S. Boyd, "CVX: Matlab software for disciplined convex programming, Version 2.1," 2014. [Online]. Available: <https://cvxr.com/cvx/>
- [64] M. Udell, K. Mohan, D. Zeng, J. Hong, S. Diamond, and S. Boyd, "Convex optimization in Julia," in *Proc. 1st Workshop High Perform. Tech. Comput. Dyn. Lang.*, 2014, pp. 18–28.
- [65] I. Dunning, J. Huchette, and M. Lubin, "JuMP: A modeling language for mathematical optimization," *SIAM Rev.*, vol. 59, no. 2, pp. 295–320, 2017, doi: [10.1137/15M1020575](https://doi.org/10.1137/15M1020575).
- [66] A. Fu, B. Narasimhan, and S. Boyd, "CVXR: An R package for disciplined convex optimization," *J. Statist. Softw.*, vol. 94, no. 14, pp. 1–34, 2020, doi: [10.18637/jss.v094.i14](https://doi.org/10.18637/jss.v094.i14).
- [67] P. Gill, W. Murray, and M. Wright, *Practical Optimization*. New Delhi, India: SIAM, 2019.
- [68] D. Bertsekas, *Nonlinear Programming*. 2nd ed., Nashua, NH, USA: Athena Scientific, 1999.
- [69] J. Nocedal and S. Wright, *Numerical Optimization*. 2nd ed., New York, NY, USA: Springer-Verlag, 2006.
- [70] S. Boyd and L. Vandenberghe, *Introduction to Applied Linear Algebra: Vectors, Matrices, and Least Squares*. Cambridge, U.K.: Cambridge Univ. Press, 2018.
- [71] D. Arnström, A. Bemporad, and D. Axehill, "A dual active-set solver for embedded quadratic programming using recursive LDL^T updates," *IEEE Trans. Autom. Control*, vol. 67, no. 8, pp. 4362–4369, 2022, doi: [10.1109/TAC.2022.3176430](https://doi.org/10.1109/TAC.2022.3176430).
- [72] I. Kuon, R. Tessier, and J. Rose, "FPGA architecture: Survey and challenges," *Found. Trends® Electron. Des. Autom.*, vol. 2, no. 2, pp. 135–253, Apr. 2008, doi: [10.1561/10000000005](https://doi.org/10.1561/10000000005).
- [73] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and Z. Kolter, "Differentiable convex optimization layers," in *Proc. 33rd Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2019, pp. 9562–9574.
- [74] A. Agrawal, R. Verschuere, S. Diamond, and S. Boyd, "A rewriting system for convex optimization problems," *J. Control Decision*, vol. 5, no. 1, pp. 42–60, 2018, doi: [10.1080/23307706.2017.1397554](https://doi.org/10.1080/23307706.2017.1397554).
- [75] X. Si, W. Wang, C. Hu, and D. Zhou, "Remaining useful life estimation—a review on the statistical data driven approaches," *Eur. J. Oper. Res.*, vol. 213, no. 1, pp. 1–14, 2011, doi: [10.1016/j.ejor.2010.11.018](https://doi.org/10.1016/j.ejor.2010.11.018).
- [76] R. Byrne, T. Nguyen, D. Copp, B. Chalamala, and I. Gyuk, "Energy management and optimization methods for grid energy storage systems," *IEEE Access*, vol. 6, pp. 13,231–13,260, 2018, doi: [10.1109/ACCESS.2017.2741578](https://doi.org/10.1109/ACCESS.2017.2741578).
- [77] O. Nnorom, G. Ogut, S. Boyd, and P. Levis, "Aging-aware battery control via convex optimization," *Optim. Eng.*, vol. 27, pp. 1303–1326, 2026.
- [78] H. Kwakernaak and R. Sivan, *Linear Optimal Control Systems*. New York, NY, USA: Wiley, 1972.