Automatic Generation of Explicit Quadratic Programming Solvers

Maximilian Schaller Daniel Arnström Alberto Bemporad **Stephen Boyd**

Stanford University, September 12 2025

Parametric convex optimization

parametric convex optimization problem, a.k.a. multiparametric program:

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

- $x \in \mathbf{R}^n$ is the optimization variable
- f_0 is convex objective function
- $f_1, \dots, f_m$ are convex inequality constraint functions
- $h_1 \dots, h_q$ are affine equality constraint functions
- $\theta \in \Theta \subseteq \mathbf{R}^p$ is the **parameter**

Applications

- control, supply chain, finance
  - repeatedly solve problem with different data (‘context’)
  - these applications require total reliability
  - and sometimes real-time guarantees
- machine learning
  - fit multiple models with different hyper-parameters
  - refit model periodically as distribution shifts

Solution methods

we assume parametrized problem is given in a domain specific language such as `CVXPY`

solution approaches:

- parse, form, and solve problem for each θ
- use `Parameter` in `CVXPY` to avoid re-compilation
- use `CVXPYgen` to generate custom solver (in C or Rust) for parametrized problem

this talk: a special case in which the generated solver is **explicit**, not iterative

- applies only to **small quadratic programs** of special form
- still widely useful

Multiparametric quadratic programming

parametric QP:

$$
\begin{array}{ll}
\text{minimize} & (1/2)x^T P x + (u + U\theta)^T x \\
\text{subject to} & Ax \leq v + V\theta,
\end{array}
$$

- $x \in \mathbf{R}^n$ is the variable
- $\theta \in \Theta \subseteq \mathbf{R}^p$ is the parameter
- $P \in \mathbf{S}^n_{++}$, $u \in \mathbf{R}^n$, $U \in \mathbf{R}^{n \times p}$, $A \in \mathbf{R}^{m \times n}$, $v \in \mathbf{R}^m$, and $V \in \mathbf{R}^{m \times p}$ are given

- solution map is $\theta \mapsto x^\star(\theta)$
- dual solution map is $\theta \mapsto \lambda^\star(\theta)$

Piecewise affine solution map

- solution map is **piecewise affine** on K polyhedral regions:

$$x^\star = F_k\theta + g_k \quad \text{when} \quad H_k\theta \leq j_k, \quad k = 1, \ldots, K$$

  (dual solution map is also piecewise affine, with same regions)
- set of active constraints is $\mathcal{A} = \{i \mid (Ax^\star)_i = (v + V\theta)_i\}$
- solution $x^\star$ is affine function of θ for each set of active constraints
- can search over all 2^m possible active sets and
  - determine if it can occur
  - if so, compute F_k, g_k, H_k, j_k
- can construct tree for efficient search for region, given θ

Explicit parametric QP solver

- **ahead of time:** pre-compute coefficient matrices for regions and search tree

- **on-line:** given θ
  - determine region k for which $H_k\theta \leq j_k$ holds
  - evaluate $x^\star = F_k\theta + g_k$

- table lookup, followed by evaluating affine function
- super fast, with bounded solve time
- division free
- can implement in reduced precision

- only practical when memory to store coefficient matrices is small enough

CVXPYgen

- `CVXPYgen` (Schaller et al. 2022) is a general purpose code generator for `CVXPY`
- `CVXPYgen` now supports explicit QP solver option
- relies on explicit QP solver package `PDAQP` (Arnström et al. 2022)
- generates flat C code for solver, Python interface, documentation, . . .

Example: Portfolio optimization

- choose portfolio weights $w \in \mathbf{R}^N$ for N financial assets
- $w_i \geq 0$ is fraction of portfolio value invested in asset i, $w_1 + \cdots + w_N = 1$
- asset return mean $\mu \in \mathbf{R}^N$, covariance $\Sigma \in \mathbf{S}^N_{++}$
- **Markowitz portfolio construction:** maximize risk-adjusted return

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

  variable $w \in \mathbf{R}^N$, parameter $\mu \in \mathbf{R}^N$, risk aversion hyper-parameter $\gamma > 0$
- we consider case where Σ and γ are fixed, μ is a parameter

Portfolio optimization

- $N = 7$ stocks with largest market capitalization as of January 1, 2017
- Σ is sample covariance over years 2017 and 2018; $\gamma = 2$
- generated explicit solver has $K = 127$ regions
- test on 250 problem instances with μ the one-year trailing asset return average for each trading day in 2019

Generating the explicit solver

```
import cvxpy as cp
from cvxpygen import cpg

N, Sigma, gamma = ...
w = cp.Variable(N, name='w')
mu = cp.Parameter(N, name='mu')

obj = cp.Maximize(mu @ w - gamma * cp.quad_form(w, Sigma))
constr = [cp.sum(w) == 1, w >= 0]
prob = cp.Problem(obj, constr)

cpg.generate_code(prob, solver='explicit')
```

Data structure

```
typedef struct {
  cpg_float    *w;              // primal variable w
} CPG_Prim_t;

typedef struct {
  cpg_float    d0;              // dual variable d0
  cpg_float    *d1;             // dual variable d1
} CPG_Dual_t;

typedef struct {
  CPG_Prim_t *prim;             // primal solution
  CPG_Dual_t *dual;             // dual solution
} CPG_Result_t;
```

Using the explicit solver in C

```c
#include ...

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

  cpg_update_mu(0, 0.01);
  cpg_solve();
  printf("%f\n", CPG_Result.prim->w[0]);
  printf("%f\n", cpg_obj());

  return 0;

}
```

Using the explicit solver in CVXPY

```
from code_dir.cpg_solver import cpg_solve
prob.register_solve('explicit', cpg_solve)

mu.value = ...
prob.solve(method='explicit')

print(f'w:     {w.value}')
print(f'dual:  {constr[1].dual_value}')
print(f'obj:   {obj.value}')
```

Timing and code size

	Solve (Python)	Solve (C)	Gen. + compile	Binary size
`CVXPY`	0.5441 ms	–	–	–
`CVXPYgen` iterative	0.0502 ms	0.0070 ms	5.1 s	76 KB
`CVXPYgen` explicit	0.0113 ms	0.0005 ms	20.8 s	234 KB

Conclusions

- solution of a QP is piecewise affine function of linear objective coefficients and righthand sides of constraints
- can explicitly compute coefficient matrices that define regions and solution map (when the number of regions is small enough)
- `CVXPYgen` now supports explicit QP solving
- generated solvers are
  - typically $100\times$ to $10000\times$ faster than iterative methods
  - suitable for real-time safety-critical applications

References

M. Schaller, D. Arnström, A. Bemporad, and S. Boyd, "Automatic Generation of Explicit Quadratic Programming Solvers", arXiv preprint *arXiv:2506.11513*, 2025

- `https://stanford.edu/~boyd/papers/cvxpygen_mpqp.html`

`CVXPY` documentation

- `https://www.cvxpy.org`

Code

- `https://github.com/cvxgrp/cvxpygen`
- `https://github.com/darnstrom/pdaqp`