Embedded Code Generation with CVXPY

Maximilian Schaller[⋆] Goran Banjac[⋆] Steven Diamond[§] Akshay Agrawal[*]
Bartolomeo Stellato[+] Stephen Boyd[*]

[⋆]ETH Zurich [§]Gridmatic [*]Stanford University [+]Princeton University

ACCESS Seminar, March 24 2022

Outline

Parametrized convex optimization

Code generation

CVXPYgen

Parametrized convex optimization

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

- $x \in \mathbf{R}^n$ is the optimization variable
- f_0 is the convex objective function, to be minimized
- $f_1, \ldots, f_m$ are convex inequality constraint functions
- $g_1 \ldots, g_p$ are affine equality constraint functions
- $\theta \in \mathbf{R}^d$ is the parameter

- used in control, signal processing, finance, and many other areas

Disciplined convex programming (DCP)

- f_i and g_i described by expression trees, built from a library of atomic functions
- must follow DCP composition rules from convex analysis (Grant, Ye, Boyd 2006)
- ensures that problem is convex

- example:

$$f_0(x) = \frac{x_1^2}{\min(x_2 - 1, 1)}, \quad x_2 > 1$$

  is convex
- express in DCP-compliant form as

```
f_0 = quad_over_lin(x[1], min(x[2]-1, 1))
```

Domain-specific languages (DSLs) for convex optimization

DSL for convex optimization

1. translates (canonicalizes) a DCP-compliant problem description to a canonical form, *e.g.*, a linear program (LP) or quadratic program (QP)
2. calls a standard solver to solve the canonicalized problem
3. retrieves solution of original problem from solution of the canonicalized problem

examples:

- CVX (Grant 2006) and YALMIP (Löfberg 2004) in Matlab
- CVXPY (Diamond 2013) in Python
- Convex.jl (Udell 2014) and JuMP (Dunning 2017) in Julia
- CVXR (Fu 2020) in R

Example: Original problem and canonicalized form

- original (nonnegative least squares) problem

$$\begin{array}{ll} \text{minimize} & \|Gx - h\|_2^2 \\ \text{subject to} & x \geq 0, \end{array}$$

with variable $x \in \mathbf{R}^n$, parameters $\theta = (G, h)$

- canonicalize to form accepted by QP solver OSQP (Stellato 2020),

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

with variable $\tilde{x} \in \mathbf{R}^{\tilde{n}}$, canonical parameters $\tilde{\theta} = (P, q, A, l, u)$

Example: Canonicalization and retrieval

- canonicalize original problem using $\tilde{x} = x$ and

$$P = 2G^T G, \quad q = -2G^T h, \quad A = I, \quad l = 0, \quad u = \infty$$

- retrieve solution of original problem as $x^\star = \tilde{x}^\star$

- this example was simple and could easily be done by hand
- more complex examples much less so

Example: CVXPY code

```
import cvxpy as cp

# declare variable
x = cp.Variable(n, name='x')

# declare parameters
G = cp.Parameter((m, n), name='G')
h = cp.Parameter(m, name='h')

# declare problem
problem = cp.Problem(cp.Minimize(cp.sum_squares(G@x-h)), [x>=0])

# specify parameter values
G.value = numpy.random.randn(m, n)
h.value = numpy.random.randn(m)

# solve
problem.solve(solver='OSQP')
```

Outline

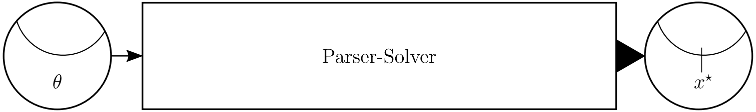
Parametrized convex optimization

Code generation

CVXPYgen

Parser-solvers

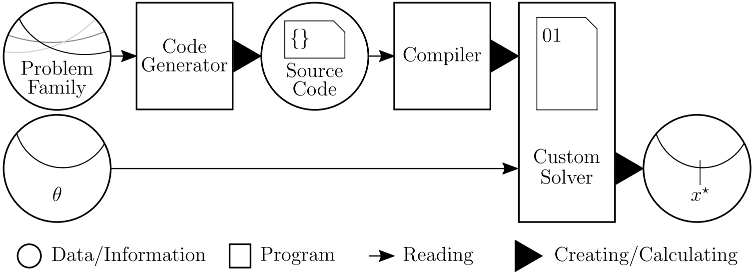
- parser-solvers canonicalize each time the problem is solved



- parser-solvers compile a *problem instance* into a *canonicalized problem instance*, then solve it
- most DSLs are parser-solvers

Code generators

- code generators compile a *problem family* into source code for a *custom solver*



- useful for
  - embedded applications, possibly with hard real-time deadlines
  - speeding up the solution of many different problem instances

CVXGEN code generator

- developed by Mattingley and Boyd in 2010
- handles problems transformable to QPs
- generates custom interior-point solver in flat, explicit C
- handles problem families with up to a few thousand parameters
- generated code suitable for real-time control systems
- used for autonomous driving, dynamic energy management, real-time trading, precision landing (*e.g.*, all SpaceX Falcon 9 and Falcon Heavy landings)

CVXGEN in action

https://blogs.nasa.gov/spacex/2019/06/25/side-boosters-have-landed/

Outline

Parametrized convex optimization

Code generation

CVXPYgen

CVXPYgen

- a new open-source code generator built on CVXPY
- developed by Schaller, Banjac, Diamond, Agrawal, Stellato, and Boyd in 2022
- generates custom canonicalizer and retrieval in flat C
- can be used with multiple solvers: OSQP, SCS (O'Donoghue 2016), ECOS (Domahidi 2013)
- first generic code generator that supports SOCPs
- supports warm-starting, which can give significant speedup
- handles high-dimensional parameters with user-defined sparsity patterns
- compiled CVXPYgen solver can be used as a custom solver for CVXPY (!)

Disciplined parametrized programming (DPP)

- restricts how parameters enter problem description, in addition to DCP rules
- for DPP-compliant problems, canonicalization and retrieval can be affine mappings (Agrawal 2019)

$$\tilde{\theta} = C \begin{bmatrix} \theta \\ 1 \end{bmatrix}, \qquad x^\star = R \begin{bmatrix} \tilde{x}^\star \\ 1 \end{bmatrix}$$

- C and R are (very) sparse matrices
- CVXPYgen generates flat C code to implement canonicalization and retrieval
- sparse-matrix-vector multiplies, using pointers or avoiding updates when possible

Example (again)

- canonicalize original nonnegative least squares problem

$$\begin{array}{ll} \text{minimize} & \|Gx - h\|_2^2 \\ \text{subject to} & x \geq 0 \end{array}$$

  to OSQP standard form

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

- canonicalization shown before is not an affine mapping from θ to $\tilde{\theta}$

Example: Affine canonicalization and retrieval

- first transform to problem

$$\begin{array}{ll} \text{minimize} & \|\tilde{x}_2\|_2^2 \\ \text{subject to} & \tilde{x}_2 = G\tilde{x}_1 - h, \quad \tilde{x}_1 \geq 0, \end{array}$$

  with variable $\tilde{x} = (\tilde{x}_1, \tilde{x}_2)$, $\tilde{x}_1 = x$

- canonicalization is affine:

$$P = \begin{bmatrix} 0 & 0 \\ 0 & 2I \end{bmatrix}, \quad q = 0, \quad A = \begin{bmatrix} G & -I \\ I & 0 \end{bmatrix}, \quad l = \begin{bmatrix} h \\ 0 \end{bmatrix}, \quad u = \begin{bmatrix} h \\ \infty \end{bmatrix}$$

- retrieval is affine: $x^\star = [I \;\, 0]\tilde{x}^\star$

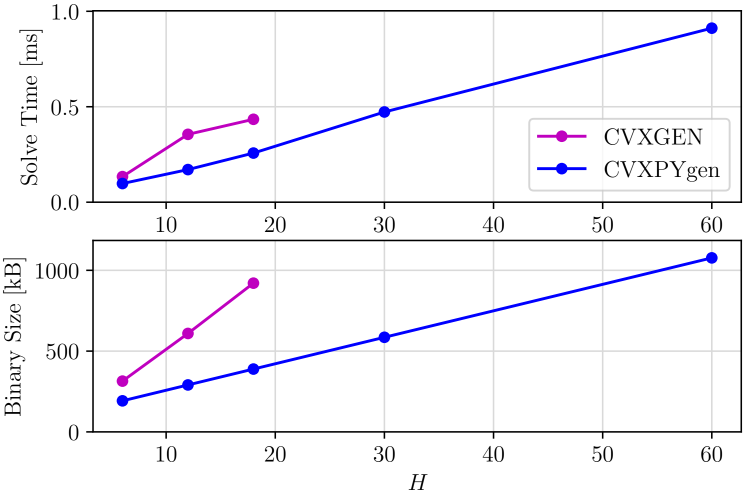
Example: CVXPY/CVXPYgen code

```
import cvxpy as cp
from cvxpygen import cpg

# model problem
x = cp.Variable(n, name='x')
G = cp.Parameter((m, n), name='G')
h = cp.Parameter(m, name='h')
problem = cp.Problem(cp.Minimize(cp.sum_squares(G@x-h)), [x>=0])

# generate code
cpg.generate_code(problem)
```

Example: Model predictive control (MPC)

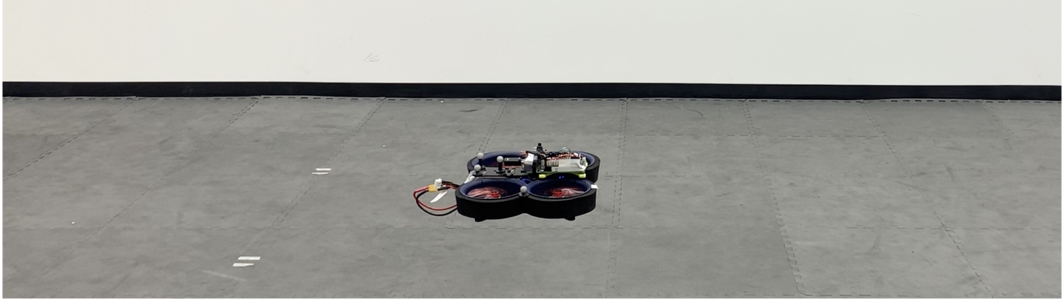
- family of MPC problems for control of a drone
- parametrized by horizon length $H \in \{6, 12, 18, 30, 60\}$
- number of variables around $10H$
- binary sizes and solve times on MacBook Pro 2.3GHz Intel i5, using gcc -O3

Comparison with CVXGEN



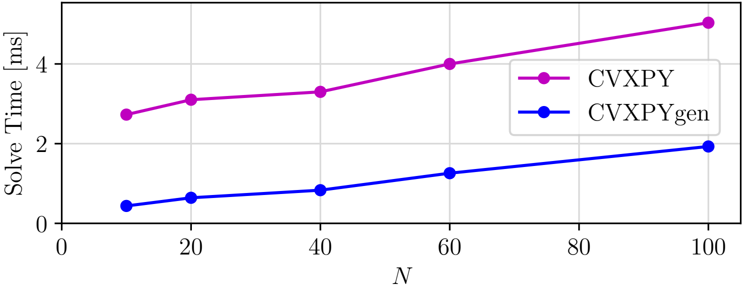
Deployment in embedded system

- use generated solver to control 14×14 cm quadcopter
- generated code compiled in a robot operating system (ROS) node
- run on drone's Intel Atom x5-Z8350 processor at 30Hz



Example: Portfolio trading

- family of portfolio optimization problems
- parametrized by number of assets $N \in \{10, 20, 40, 60, 100\}$
- number of variables around $2N$
- solve times with CVXPY and CVXPY interface to CVXPYgen



Break-even point

- *break-even point*: number of instances that need to be solved before CVXPYgen is faster than CVXPY, when we include the code generation and compilation time
- around 5000, and not too dependent on N
- typical portfolio optimization back-test involves daily trading over multiple years, with hundreds of different hyper-parameter values
- gives order 100k or more solves, well above the break-even point

Conclusions

CVXPYgen

- gives seemless path from prototyping in Python/CVXPY to implementation in C
- handles wider variety of problems than CVXGEN (*e.g.*, SOCPs)
- outperforms CVXGEN in terms of
  - allowable problem size
  - compiled code size
  - solve times
- gives significant speedup on general-purpose machines with many solves (compared with CVXPY)

Try it out!

- `https://github.com/cvxgrp/cvxpygen`
- `pip install cvxpygen`