Open Source Optimization Software: CVXGEN, OSQP

Chee Wei TAN

The Power of Open Source Optimization Software[1]

[1]World's first vertical-landing rocket, SpaceX's Falcon9, runs on CVXGEN. Autonomous Precision Landing of Space Rockets, L, Blackmore, NAE 2016

CVXGEN Software

- CVXGEN turns a mathematical problem description into a high-speed solver
- CVXGEN: A Code Generator for Embedded Convex Optimization J. Mattingley and S. Boyd, Optimization and Engineering, 13(1):1–27, 2012
- Generates fast custom code for small quadratic programming problems using a web interface (Software-as-a-Service)
- What are practical QP's you can think of?
- What are some features of a fastest solver with the smallest code size?

Operator Splitting Quadratic Program Solver

- The OSQP (Operator Splitting Quadratic Program) solver is a numerical optimization package for solving convex quadratic programs.[2] https://osqp.org
- Uses custom ADMM-based first-order method requiring only a single matrix factorization in the setup phase. All the other operations are extremely cheap. Also implements custom sparse linear algebra routines exploiting structures in the problem data.
- Open source software with many interface and hardware wrappers

[2] Stellato, B., G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. "OSQP: An Operator Splitting Solver for Quadratic Programs". Mathematical Programming Computation, 2020, doi: https://doi.org/10.1007/s12532-020-00179-2.

OSQP Solver

OSQP solves convex quadratic programs (QPs) of the form

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

where $x \in \mathbb{R}^n$ is the optimization variable. The objective function is defined by a positive semidefinite matrix $P \in \mathbf{S}_+^n$ and vector $q \in \mathbb{R}^n$. The linear constraints are defined by matrix $A \in \mathbb{R}^{m \times n}$ and vectors l and u so that $l_i \in \mathbb{R} \cup \{-\infty\}$ and $u_i \in \mathbb{R} \cup \{+\infty\}$ for all $i \in \{1, \cdots, m\}$.

OSQP Solver

Algorithm: The solver runs the following ADMM algorithm (details of ADMM are covered in later lectures):

$$
\begin{aligned}
(x^{k+1}, v^{k+1}) &\leftarrow \text{solve linear system} \\
\tilde{z}^{k+1} &\leftarrow z^k + \rho^{-1}(v^{k+1} - y^k) \\
z^{k+1} &\leftarrow \prod(\tilde{z}^k + \rho^{-1} y^k) \\
y^{k+1} &\leftarrow y^k + \rho(\tilde{z}^{k+1} - z^{k+1})
\end{aligned}
$$

where $\prod$ is the projection onto the hyperbox $[l, u]$ and ρ is the ADMM step-size.

OSQP Solver

Linear system solution: The linear system solution is the core part of the algorithm. It can be done using a direct or indirect method. With a direct linear system solver we solve the following linear system with a quasi-definite matrix

$$\begin{bmatrix} P+\sigma I & A^T \\ A & -\rho^{-1} I \end{bmatrix} \begin{bmatrix} x^{k+1} \\ v^{k+1} \end{bmatrix} = \begin{bmatrix} \sigma x^k - q \\ z^k - \rho^{-1} y^k \end{bmatrix}$$

With an indirect linear system solver we solve the following linear system with a positive definite matrix

$$(P+\sigma I+\rho A^T A)x^{k+1} = \sigma x^k - q + A^T(\rho z^k - y^k).$$

OSQP Solver

Convergence: At each iteration k OSQP produces a tuple (x^k, z^k, y^k) with $x^k \in \mathbb{R}^n$ and $z^k, y^k \in \mathbb{R}^m$.
The primal and and dual residuals associated to (x^k, z^k, y^k) are

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

Complementary slackness is satisfied by construction at machine precision. If the problem is feasible, the residuals converge to zero as $k \to \infty$. The algorithm stops when the norms of r^k_{prim} and r^k_{dual} are within the specified tolerance levels ϵ_{prim} >0 and $\epsilon_{dual} > 0$

$$\|r^k_{prim}\|_\infty \leq \epsilon_{prim}, \|r^k_{dual}\|_\infty \leq \epsilon_{dual}.$$

We set the tolerance levels as

$$\epsilon_{prim} = \epsilon_{abs} + \epsilon_{rel} \max\{\|Ax^k\|_\infty, \|z^k\|_\infty\}$$
$$\epsilon_{prim} = \epsilon_{abs} + \epsilon_{rel} \max\{\|Px^k\|_\infty, \|A^T y^k\|_\infty, \|q\|_\infty\}$$

OSQP Solver

ρ step-size: To ensure quick convergence of the algorithm we adapt ρ by balancing the residuals. In default mode, the inteval (i.e., number of iterations) at which we update ρ is defined by a time measurement. When the iterations time becomes greater than a certain fraction of the setup time, i.e. `adaptive_rho_fraction`, we set the current number of iterations as the interval to update ρ. The update happens as follows

$$\rho^{k+1} \leftarrow \rho^k \sqrt{\frac{\|r_{prim}\|_\infty}{\|r_{dual}\|_\infty}}$$

Note that ρ is updated only if it is sufficiently different than the current one, e.g, if it is `adaptive_rho_tolerance` times larger or smaller than the current one.

OSQP Solver

OSQP is able to detect if the problem is primal or dual infeasible.
Primal infeasibility:
When the problem is primal infeasible, the algorithm generates a certificate of infeasibility, i.e., a vector $v \in \mathbb{R}^m$ such that

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

where $v_+ = max(v, 0)$ and $v_- = min(v, 0)$.
The primal infeasibility check is then

$$\|A^T v\|_\infty \leq \epsilon_{\text{prim\_inf}}, \quad u^T (v)_+ + l^T (v)_- \leq -\epsilon_{\text{prim\_inf}}.$$

OSQP Solver

Dual infeasibility:
When the problem is dual infeasible, OSQP generates a vector $s \in \mathbb{R}^n$ being a certificate of dual infeasibility, i.e.,

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

The dual infeasibility check is then

$$\|Ps\|_\infty \leq \epsilon_{\text{dual\_inf}}, \quad q^T s \leq -\epsilon_{\text{dual\_inf}},$$
$$(As)_i \begin{cases} \in [-\epsilon_{\text{dual\_inf}}, \epsilon_{\text{dual\_inf}}], & u_i, l_i \in R \\ \geq \epsilon_{\text{dual\_inf}}, & u_i = +\infty \\ \leq -\epsilon_{\text{dual\_inf}}, & l_i = -\infty. \end{cases}$$

OSQP Example

Toy example:

$$
\begin{array}{ll}
\text{minimize} & (1/2)x^T P x + q^T x \\
\text{subject to} & Gx \leq h \\
& Ax = b.
\end{array}
$$

Here $P \in S_+^n$, $q \in \mathbb{R}^n$, $G \in \mathbb{R}^{n \times m}$, $h \in \mathbb{R}^m$, $A \in \mathbb{R}^{p \times n}$, and $b \in \mathbb{R}^n$ are problem data and $x \in \mathbb{R}^n$ is the optimization variable. The inequality constraint $Gx \leq h$ is elementwise.

OSQP Example

```
import cvxpy as cp
import numpy as np

# Generate a random non-trivial quadratic program.
m, n, p = 15, 10, 5
np.random.seed(1)
P = np.random.randn(n, n)
P = P.T@P
q = np.random.randn(n)
G = np.random.randn(m, n)
h = G@np.random.randn(n)
A = np.random.randn(p, n)
b = np.random.randn(p)

# Define and solve the CVXPY problem.
x = cp.Variable(n)
prob = cp.Problem(cp.Minimize((1/2)*cp.quad_form(x, P) + q.T@x),
                 [G@x <= h,
                  A@x == b])
prob.solve(solver='OSQP', max_iter=2000)

# Print result.
print("\nThe optimal value is", prob.value)
print("A solution x is")
print(x.value)
print("A dual solution corresponding to the inequality constraints is")
print(prob.constraints[0].dual_value)
```