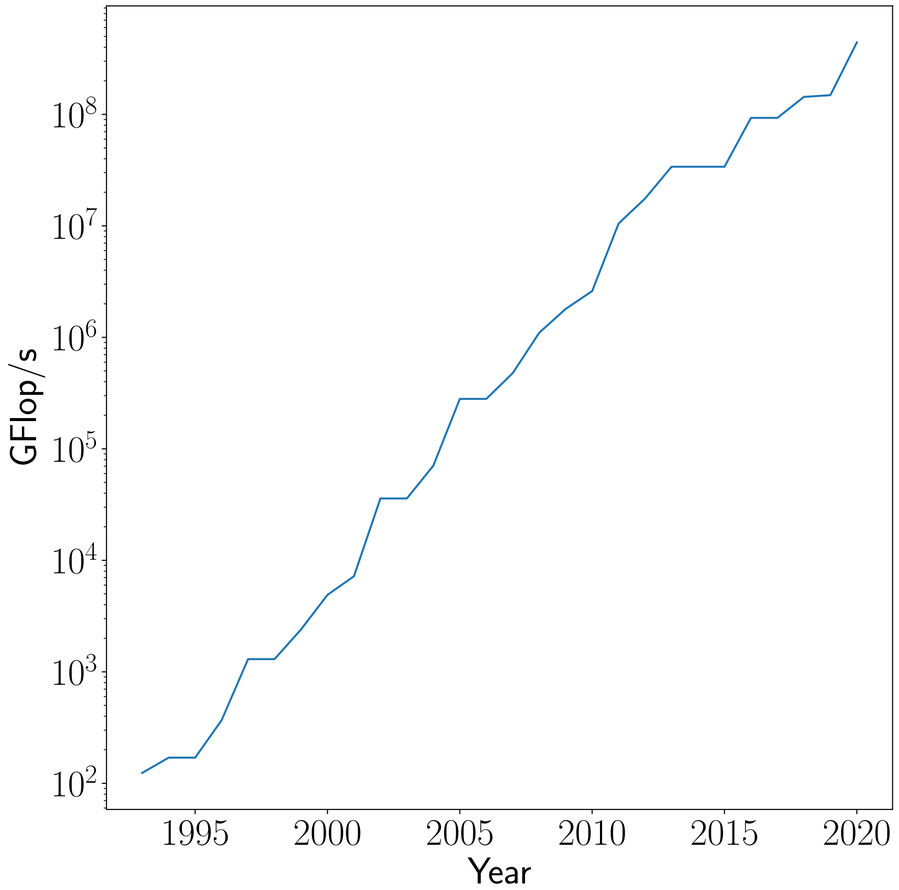
Data-Driven Embedded Optimization for Control

PRINCETON UNIVERSITY ORFE

Bartolomeo Stellato — Joint Princeton Robotics and Optimization Seminar, May 6 2021

Tremendous progress in optimization

Top500 peak CPU power



Hardware + Software

400 billion times speedups!

400,000 years → 30 seconds

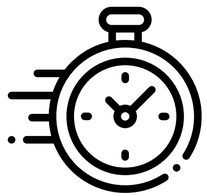
[Our World in Data, https://github.com/owid/owid-datasets]

Is it enough?

400,000 years → 30 seconds

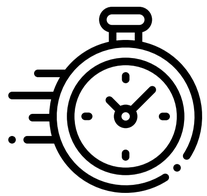
Robotics

< 10 milliseconds

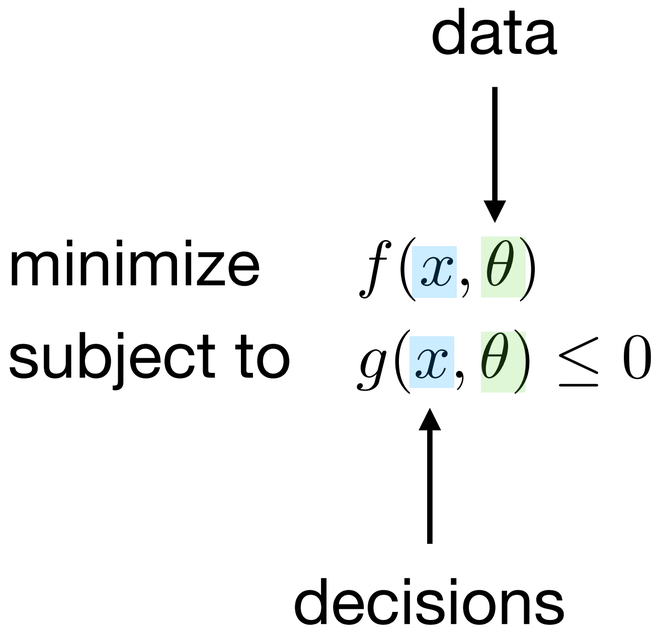


High-Frequency Trading

< 1 millisecond



Same problem with varying data

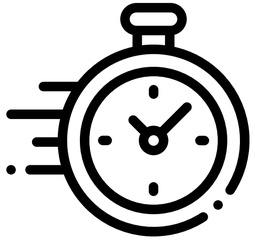


Can we solve it in **milliseconds** *or* **microseconds***?*

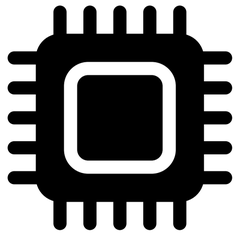
Challenges in real-time optimization

Hardware

Real-Time



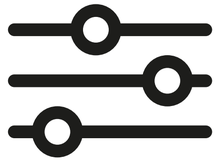
Limited resources



Software

Reliability

Easy tuning

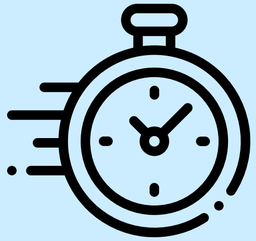


Today's talk

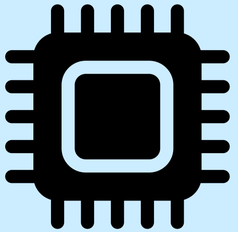
Data-Driven Embedded Optimization for Control

OSQP Solver

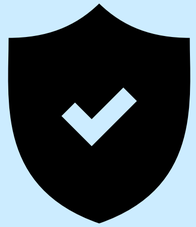
Real-Time



Limited resources

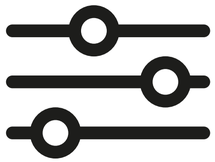


Reliability

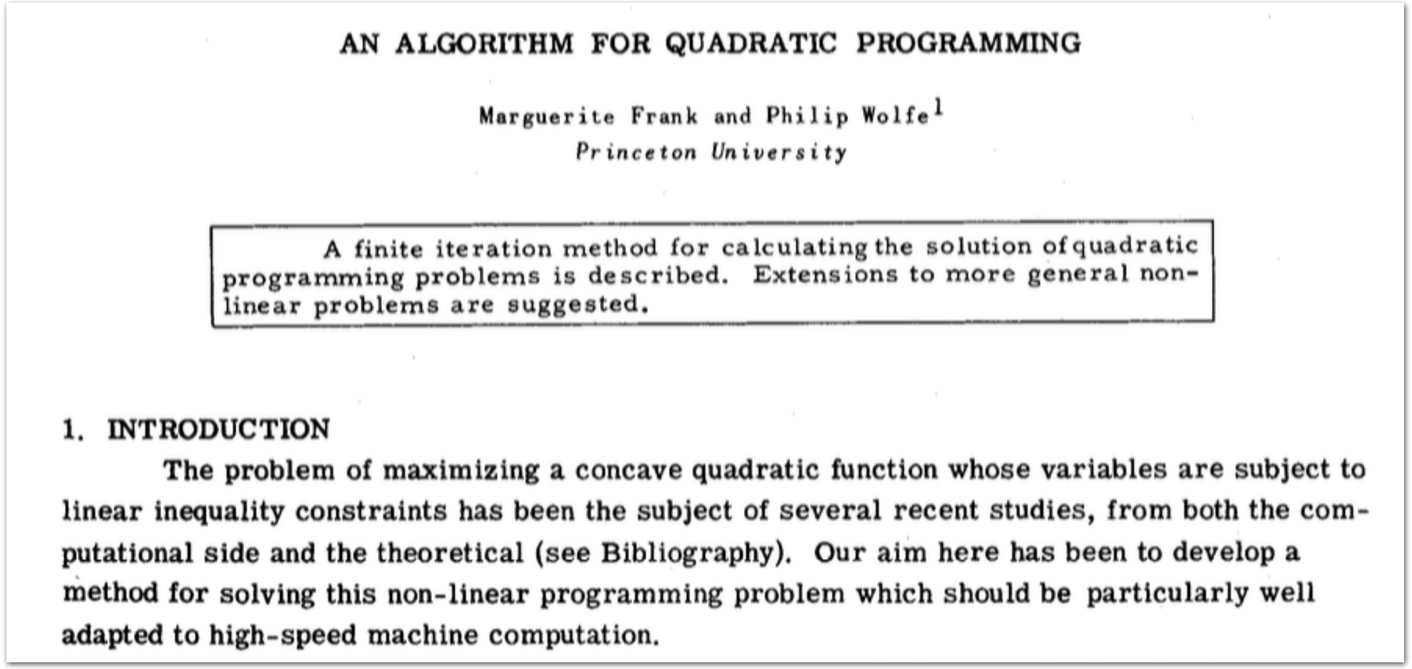


Learning
Convex Optimization
Control Policies

Easy tuning



Still quadratic programming?



AN ALGORITHM FOR QUADRATIC PROGRAMMING

Marguerite Frank and Philip Wolfe[1]
Princeton University

A finite iteration method for calculating the solution of quadratic programming problems is described. Extensions to more general non-linear problems are suggested.

1. INTRODUCTION

The problem of maximizing a concave quadratic function whose variables are subject to linear inequality constraints has been the subject of several recent studies, from both the computational side and the theoretical (see Bibliography). Our aim here has been to develop a method for solving this non-linear programming problem which should be particularly well adapted to high-speed machine computation.

March 1956!


145
140
135
130
125
Chg.
1.64

First-order methods

Wide popularity

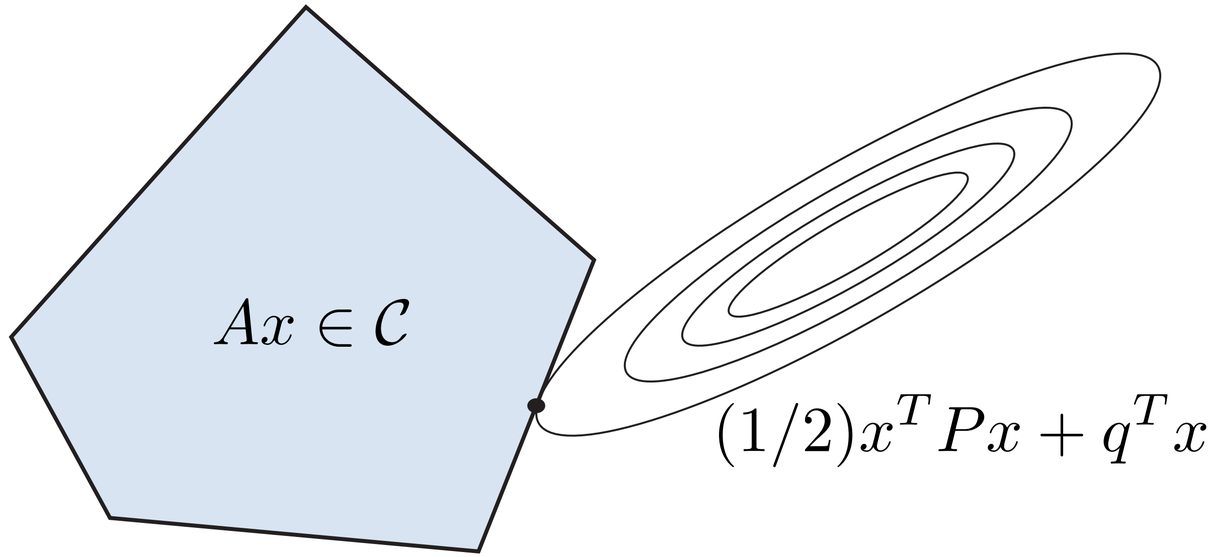
Pros	Cons	OSQP
Warm-starting	Low quality solutions	High-quality solutions
Large-scale problems	Can’t detect infeasibility	Detects infeasibility
Embeddable	Problem data dependent	Robust

The problem

minimize $(1/2)x^T Px + q^T x$

subject to $Ax \in \mathcal{C}$

Quadratic program: $\mathcal{C} = [l, u]$



ADMM

Alternating Direction Method of Multipliers

minimize $f(x) + g(x)$ ⟶

Splitting

$$\begin{aligned} \text{minimize} \quad & f(\tilde{x}) + g(x) \\ \text{subject to} \quad & \tilde{x} = x \end{aligned}$$

Iterations

$$\tilde{x}^{k+1} \leftarrow \underset{\tilde{x}}{\text{argmin}} \left(f(\tilde{x}) + \rho/2 \left\| \tilde{x} - (x^k - y^k/\rho) \right\|^2 \right)$$

$$x^{k+1} \leftarrow \underset{x}{\text{argmin}} \left(g(x) + \rho/2 \left\| x - (\tilde{x}^{k+1} + y^k/\rho) \right\|^2 \right)$$

$$y^{k+1} \leftarrow y^k + \rho \left(\tilde{x}^{k+1} - x^{k+1} \right)$$

How do we split the QP?

$$
\begin{array}{lll}
\text{minimize} & (1/2)x^T P x + q^T x & f \\
\text{subject to} & Ax = z & \\
& z \in \mathcal{C} & g
\end{array}
$$

Splitting formulation

$$
\begin{array}{lll}
 & f & g \\
\text{minimize} & (1/2)\tilde{x}^T P \tilde{x} + q^T \tilde{x} + \mathcal{I}_{Ax=z}(\tilde{x}, \tilde{z}) & + \mathcal{I}_{\mathcal{C}}(z) \\
\text{subject to} & \tilde{x} = x & \\
 & \tilde{z} = z &
\end{array}
$$

Step sizes

$\sigma > 0$

$\rho > 0$

ADMM iterations

Inner QP

$$(x^{k+1}, \tilde{z}^{k+1}) \leftarrow \underset{(x,z):Ax=z}{\operatorname{argmin}} \; (1/2)x^T P x + q^T x + \sigma/2 \left\|x - x^k\right\|^2 + \rho/2 \left\|z - z^k + y^k/\rho\right\|^2$$

$$z^{k+1} \leftarrow \Pi\left(\tilde{z}^{k+1} + y^k/\rho\right)$$

Projection onto $\mathcal{C}$

$$y^{k+1} \leftarrow y^k + \rho\left(\tilde{z}^{k+1} - z^{k+1}\right)$$

Solving the inner QP

Equality-constrained

$$\begin{array}{ll} \text{minimize} & (1/2)x^T Px + q^T x + \sigma/2 \left\|x - x^k\right\|^2 + \rho/2 \left\|z - z^k + y^k/\rho\right\|^2 \\ \text{subject to} & Ax = z \end{array}$$

Always solvable!

Reduced KKT system

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

Solving the linear system

Direct method (small to medium scale)

Quasi-definite matrix

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

Well-defined LDL^T factorization

Factorization caching

→

QDLDL
Free quasi-definite linear system solver

[https://github.com/oxfordcontrol/qdldl]

Solving the linear system

Indirect method (large scale)

Positive-definite matrix

$$\left(P + \sigma I + \rho A^T A\right) x = \sigma x^k - q + A^T(\rho z^k - y^k)$$

Conjugate gradient

Solve very large systems

→

GPU implementation

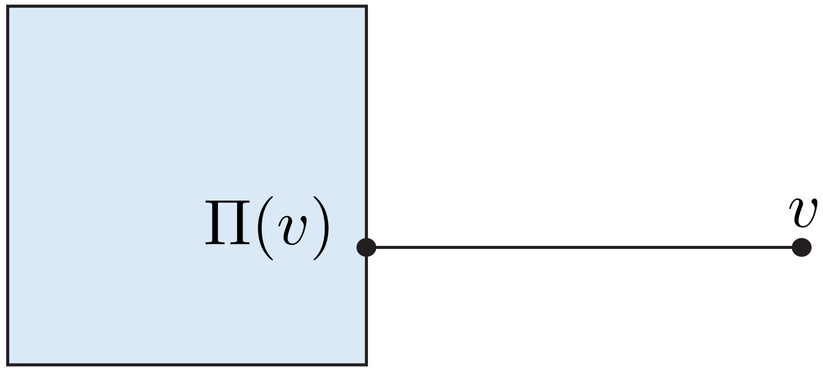
[https://github.com/oxfordcontrol/cuosqp]

Computing the projection

Quadratic program: $\mathcal{C} = [l, u]$

Box projection

$$\Pi(v) = \max(\min(v, u), l)$$



Complete algorithm

Problem

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

Algorithm

Linear system solve

$$(x^{k+1}, \nu^{k+1}) \leftarrow \text{solve} \begin{bmatrix} P + \sigma I & A^T \\ A & -\frac{1}{\rho} I \end{bmatrix} \begin{bmatrix} x^{k+1} \\ \nu^{k+1} \end{bmatrix} = \begin{bmatrix} \sigma x^k - q \\ z^k - \frac{1}{\rho} y^k \end{bmatrix}$$

Easy operations

$$\tilde{z}^{k+1} \leftarrow z^k + (\nu^{k+1} - y^k)/\rho$$

$$z^{k+1} \leftarrow \Pi\left(\tilde{z}^{k+1} + y^k/\rho\right)$$

$$y^{k+1} \leftarrow y^k + \rho\left(\tilde{z}^{k+1} - z^{k+1}\right)$$

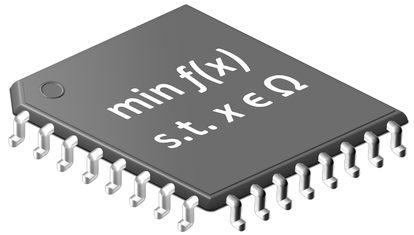
Code generation

```python
# Create OSQP object
m = osqp.OSQP()

# Initialize solver
m.setup(P, q, A, l, u,
        settings)

# Generate C code
m.codegen('folder_name')
```

Optimized C code

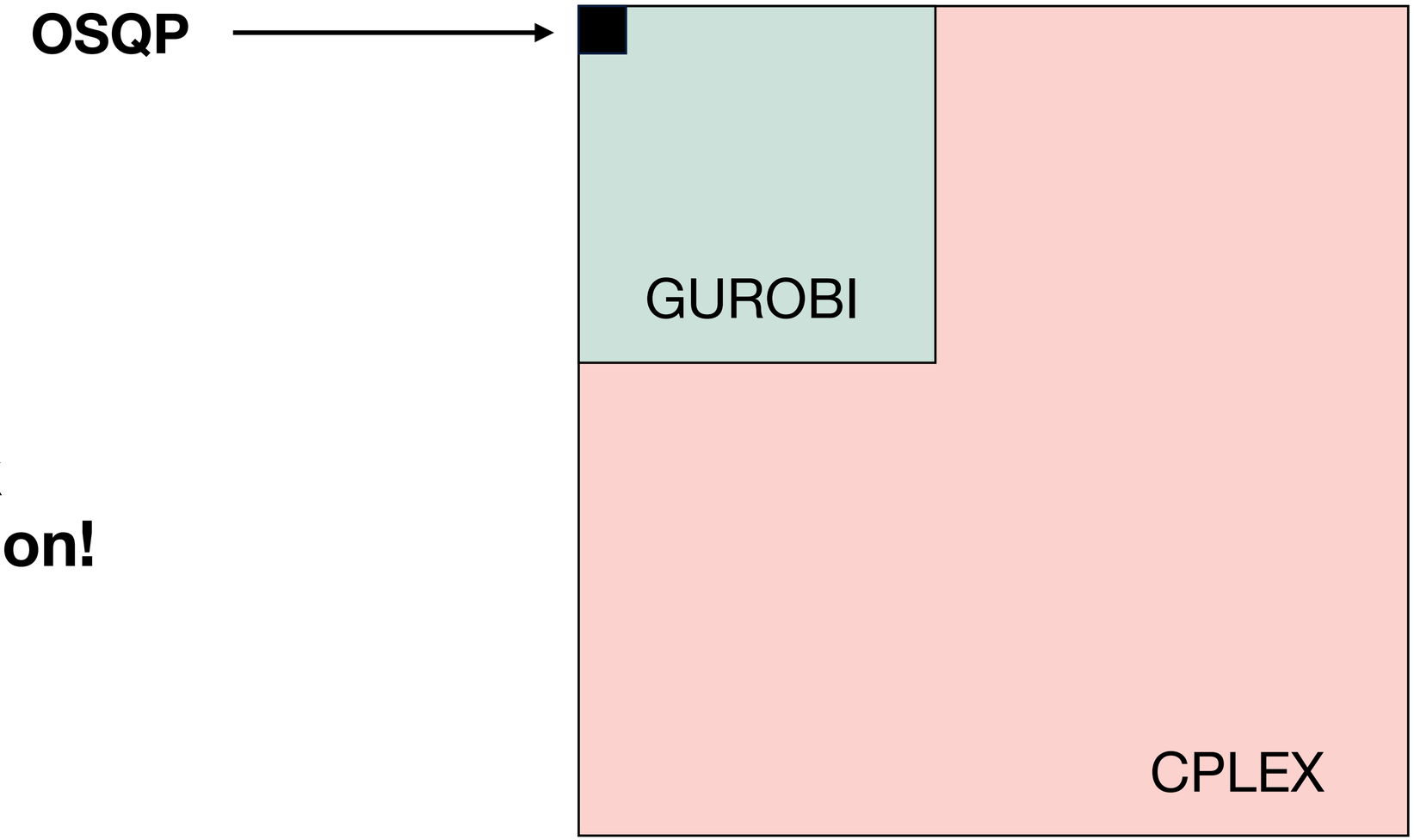
```c
// Main ADMM algorithm
for (iter = 1; iter <= work->settings->max_iter; iter ++) {
    // Update x_prev, z_prev (preallocated, no malloc)
    swap_vectors(&(work->x), &(work->x_prev));
    swap_vectors(&(work->z), &(work->z_prev));

    /* ADMM STEPS */
    /* Compute \tilde{x}^{k+1}, \tilde{z}^{k+1} */
    update_xz_tilde(work);

    /* Compute x^{k+1} */
    update_x(work);

    /* Compute z^{k+1} */
    update_z(work);

    /* Compute y^{k+1} */
    update_y(work);

    /* End of ADMM Steps */

    #ifdef CTRLC
    // Check the interrupt signal
    if (isInterrupted()) {
        update_status(work->info, OSQP_SIGINT);
        c_print("Solver interrupted\n");
        endInterruptListener();
        return 1;   // exitflag
    }
    #endif
```

Embedded Hardware



It can be compiled into division-free

Compiled code size ~80kb (low footprint)



300x Reduction!

How do we ensure fast convergence?

minimize $(1/2)x^T Px + q^T x$

subject to $Ax = z$

$l \leq z \leq u$

Primal residual

$$r^k_{\text{prim}} = Ax^k - z^k$$

Dual residual

$$r^k_{\text{dual}} = Px^k + q + A^T y^k$$

Linear system in indirect method

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

$\rho = \infty$

$$A^T A x^{k+1} = A^T z^k$$

Small primal residual

$\rho = 0$

$$Px^{k+1} = -q - A^T y^k$$

Small dual residual

What's the optimal ρ?

Extreme cases

Equality constrained QP

minimize $(1/2)x^T Px + q^T x$

subject to $Ax = b$

$\longrightarrow$

Solve in one ADMM step

$\rho \approx \infty$

$$\begin{bmatrix} P & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x \\ \nu \end{bmatrix} = \begin{bmatrix} -q \\ b \end{bmatrix}$$

Unconstrained QP

minimize $(1/2)x^T Px + q^T x$

$\longrightarrow$

Solve in one ADMM step

$\rho \approx 0$

$$Px = -q$$

We need different step sizes

Constraint-wise step size

minimize $(1/2)x^T Px + q^T x$

subject to $l \leq Ax \leq u$

Tight constraints

$l_i = (Ax^\star)_i$ or $(Ax^\star)_i = u_i$

$\rho = (\rho_1, \ldots, \rho_m)$ **can be a vector**

Never tight

$l_i = -\infty$ and $u_i = \infty$

$\downarrow$

$\rho_i = 0$

l_i or u_i finite

$\downarrow$

Always tight

$l_i = u_i \neq \infty$

$\downarrow$

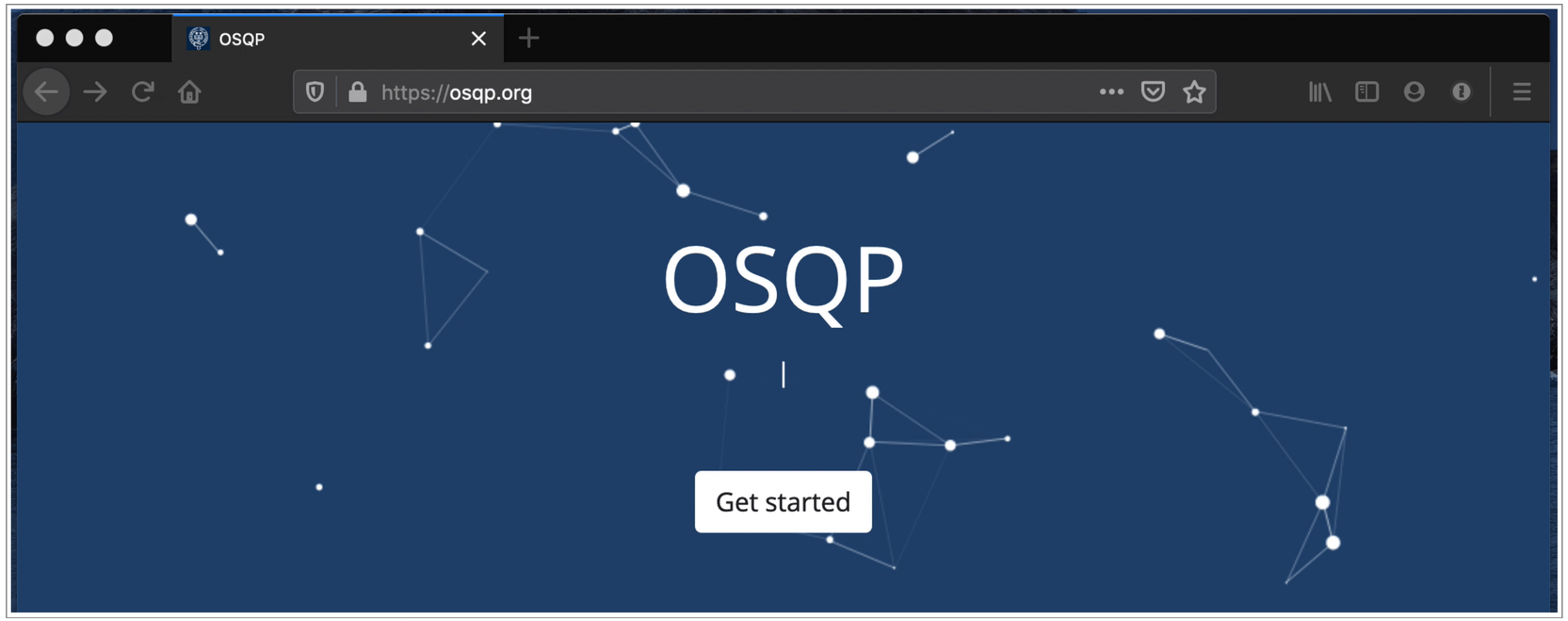
$\rho_i = \infty$

Balance residuals

$$\rho_i^{k+1} \leftarrow \rho_i^k \sqrt{\|r_{\text{prim}}\| / \|r_{\text{dual}}\|}$$

OSQP

Operator Splitting solver for Quadratic Programs



Embeddable
(can be division free!)

Supports
warm-starting

Detects
infeasibility

Solves large-scale
problems

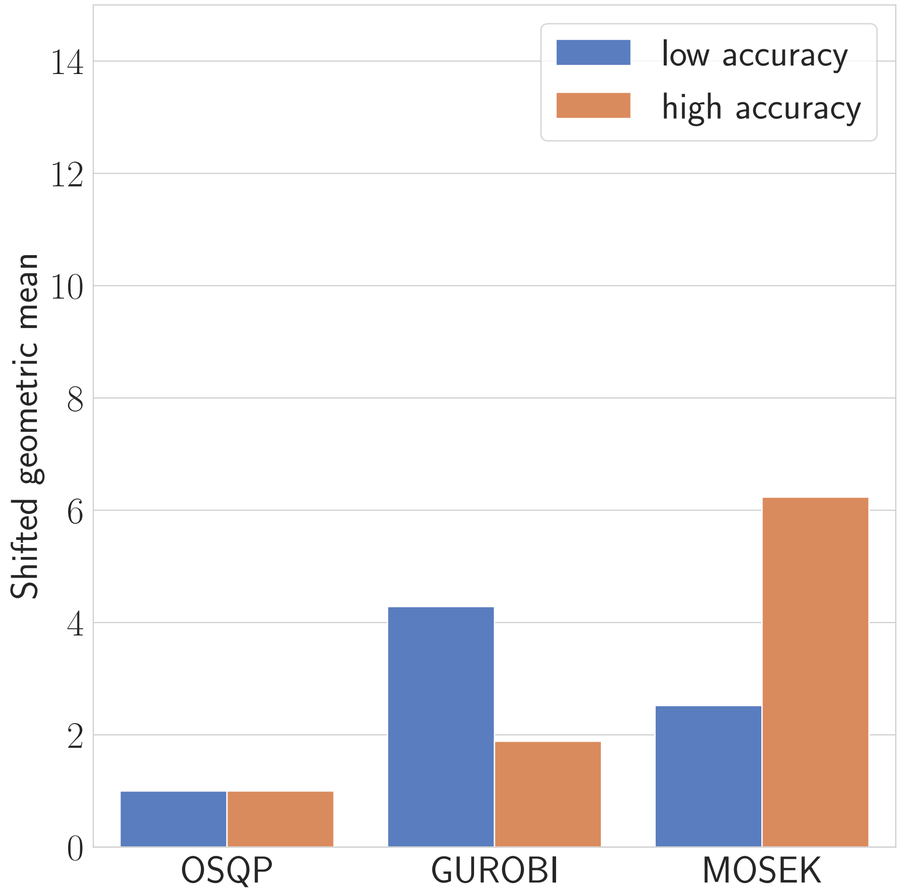
Users

More than 3 million downloads!

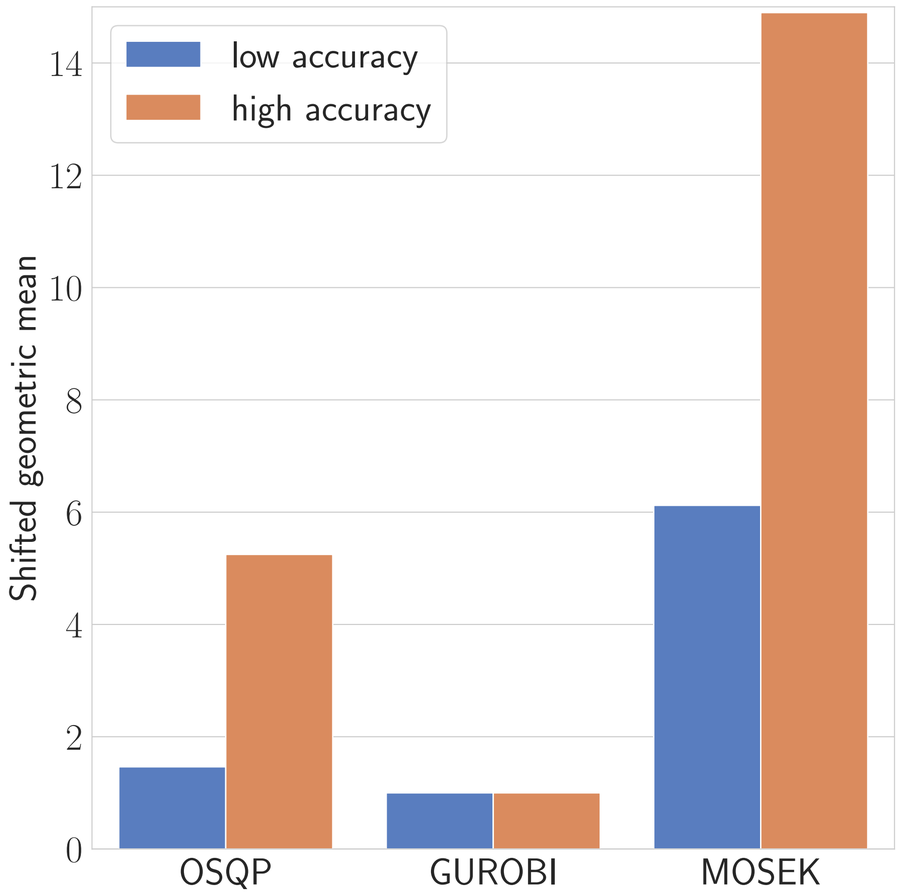
[pepy.tech/project/osqp]

Performance benchmarks

OSQP Benchmarks
(control, portfolio, lasso, SVM, etc.)



Maroz-Meszaros



[github.com/oxfordcontrol/osqp_benchmarks]

Improve the step size update

Step-size update rule

$l_i = -\infty$ and $u_i = \infty$

$\rho_i = 0$

l_i or u_i finite

Balance residuals

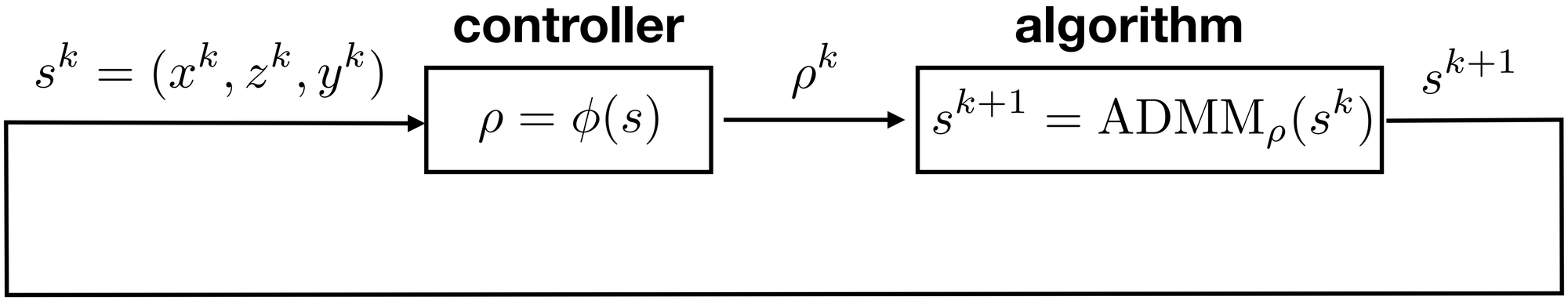
$$\rho_i^{k+1} \leftarrow \rho_i^k \sqrt{\|r_{\text{prim}}\| / \|r_{\text{dual}}\|}$$

$l_i = u_i \neq \infty$

$\rho_i = \infty$

Can we learn the update from data?

Step size choice as a control problem



Stage cost

$$\ell(s) = \begin{cases} 1 & \text{if not converged} \\ 0 & \text{if converged} \end{cases}$$

Cumulative cost

$$J = \mathbf{E} \sum_{k=1}^{\infty} \gamma^k \ell(s^k)$$

Train with
Deterministic Policy Gradient (DPG)

Constraint-wise control policy

$$\phi(s) = \begin{bmatrix} \phi_c(s_1) \\ \phi_c(s_2) \\ \vdots \\ \phi_c(s_m) \end{bmatrix}$$

Per-constraint update rule

$$\rho_i = \phi_c(s_i)$$

Per-constraint state

$$s_i = \begin{bmatrix} \min(z_i - l_i, u_i - z_i) \\ (Ax)_i - z_i \\ y_i \end{bmatrix}$$

slacks

infeasibility

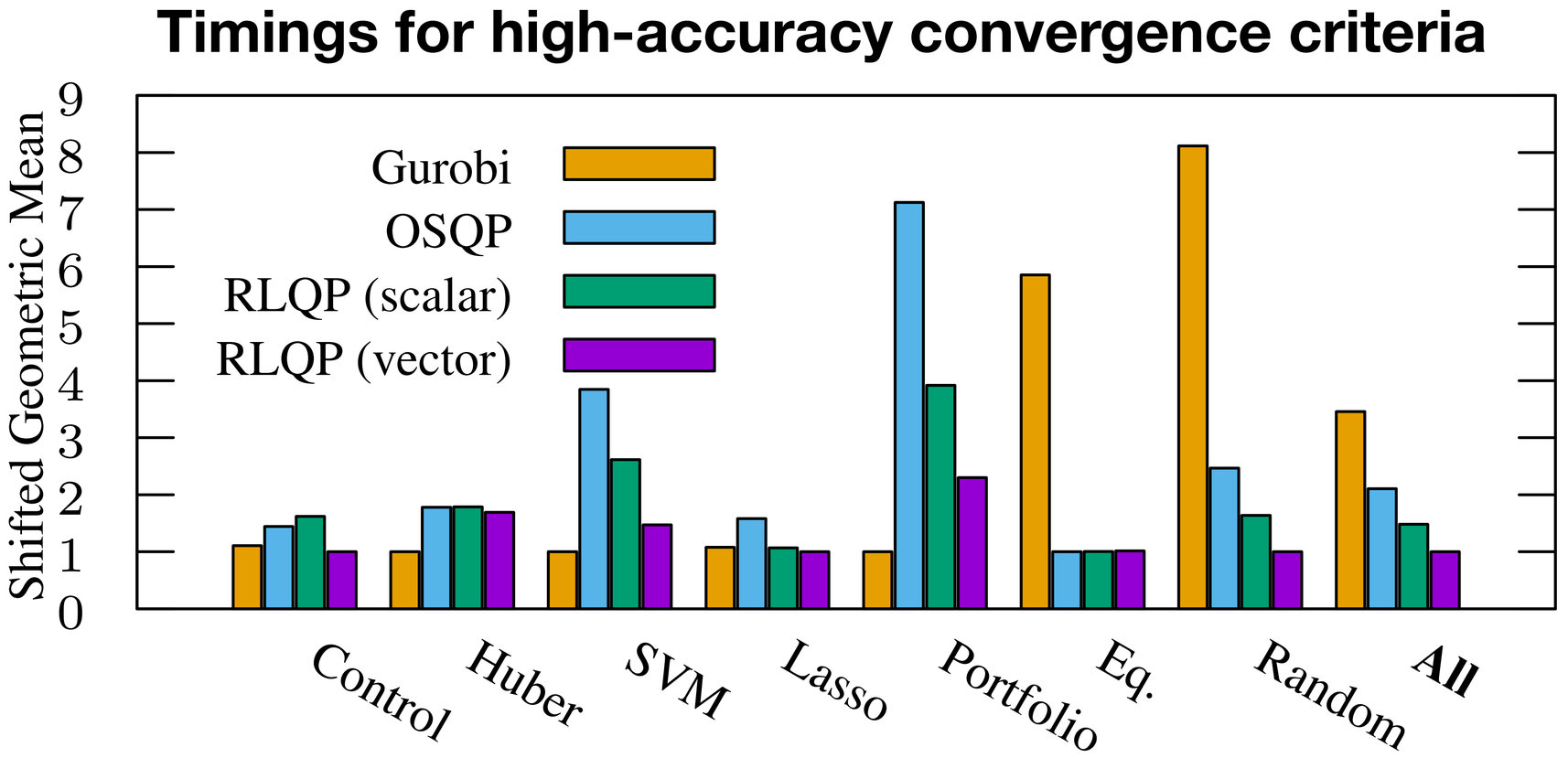
dual variable

Generalize to different dimensions

Low-dimensional state per constraint

Small NN policy $\phi_c(s_i)$

Performance with step size learning



Comparable or better performance than interior-point methods

[Accelerating Quadratic Optimization with Reinforcement Learning.
Ichnowski, Jain, Stellato, Banjac, Luo, Gonzalez, Stoica, Borrelli, and Goldberg. In prep. (2021)]

OSQP summary

Robust

Embeddable
(can be division free!)

Supports
warm-starting

Detects
infeasibility

Can improve with
data

Future work

Algorithms

- Improvements: learning & acceleration
- Semidefinite optimization (SDP)
- Sequential quadratic programming (SQP)
- Mixed-integer optimization

Architecture

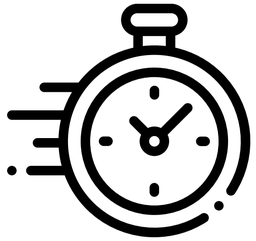
- New linear algebra
- New linear system solvers
- New languages

Today's talk

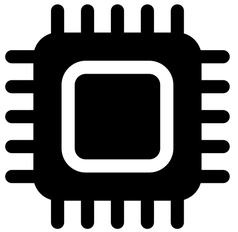
Data-Driven Embedded Optimization for Control

OSQP Solver

Real-Time



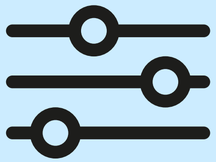
Limited resources



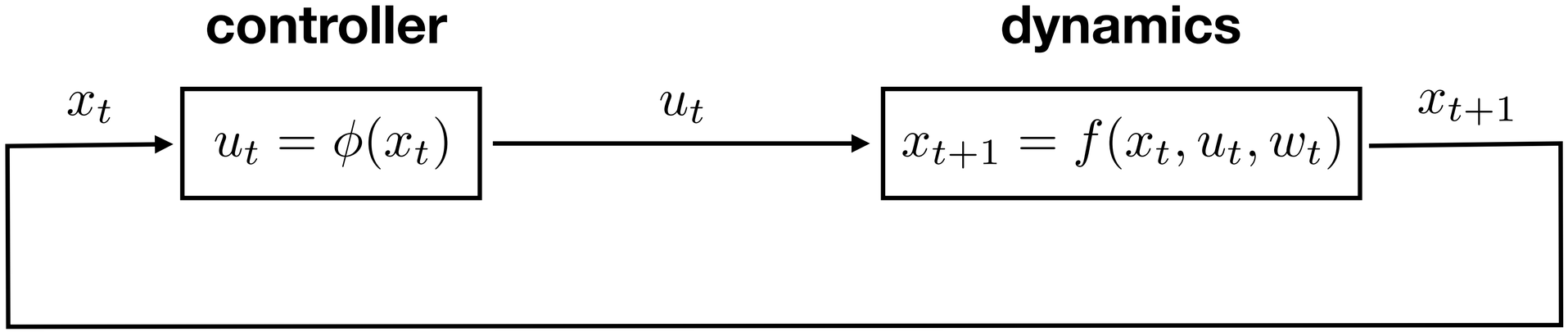
Reliability

Learning
Convex Optimization
Control Policies

Interpretable tuning



Control loop



x_t state
u_t input
w_t (random) disturbance

$\phi(x_t)$ **control policy**

Explicit vs implicit control policies

Explicit

Complete control specification

Example: PI Controller

$$u_t = -K_P e_t - K_I \sum_{\tau=0}^{t} e_\tau$$

Implicit (optimization-based)

Designer specifies goal and requirements → **Optimizer** computes the action

Example: LQR Controller

dynamics: $x_{k+1} = Ax_t + Bu_t + w_t$

stage cost: $x^T Q x + u^T R u$

$$\begin{aligned} u_t &= \underset{u}{\text{argmin}}\, u^T R u + (Ax_t + Bu)^T P (Ax_t + Bu) \\ &= Kx_t \end{aligned}$$

Convex optimization control policies (COCPs)

$$
\begin{array}{lll}
u_t = & \underset{u}{\text{argmin}} & f(x_t, u, \theta) \\
& \text{subject to} & g(x_t, u, \theta) \leq 0 \\
& & A(x_t, \theta)u = b(x_t, \theta)
\end{array}
$$

x_t state
θ parameters to tune
f, g convex functions

Many control policies are COCPs

Examples

- Linear Quadratic Regulator (LQR)
- Model predictive control (MPC)
- Actuator allocation
- Resource allocation
- Portfolio trading

Advantages

Interpretable

Satisfy constraints

Handle varying dynamics

Efficient and reliable (even division-free: OSQP)

Judging COCPs

Given a policy, state and input trajectories form a ***stochastic process***

Trajectories

$$X = (x_0, \ldots, x_{T-1}, x_T)$$
$$U = (u_0, \ldots, u_{T-1})$$
$$W = (w_0, \ldots, w_{T-1})$$

Policy cost

$$J(\theta) = \mathbf{E}\,\psi(X, U, W)$$

Approximate $J(\theta)$ from data (monte carlo simulation)

$$\hat{J}(\theta) = \frac{1}{K}\sum_{i=1}^{K} \psi(X^i, U^i, W^i)$$

COCP Example: dynamic programming

Time-separable cost

$$\psi(X, U, W) = \sum_{t=0}^{T-1} g(x_t, u_t, w_t)$$

Optimal policy as $T \to \infty$

$$\phi(x_t) = \operatorname*{argmin}_u \mathbf{E}\left(g(x_t, u, w_t) + V(f(x_t, u, w_t))\right)$$

Value function

COCP if

- f affine in x and u
- g convex in x and u
- V is convex

COCP Example: approximate dynamic programming

$$\phi(x_t) = \underset{u}{\text{argmin}}\, \mathbf{E}\left(g(x_t, u, w_t) + \hat{V}(f(x_t, u, w_t))\right)$$

Approximate value function

COCP if

- f affine in x and u
- g convex in x and u
- $\hat{V}$ is convex

(even when V is not)

Controller tuning problem

Goal

minimize $J(\theta)$

Nonconvex and difficult to solve

Traditional approaches

- **Hand-tuning** (few parameters, simple dependencies)
- **Derivative-free method** (very slow)

Learning scheme

Auto-tuning

Stochastic gradient descent

$$\theta^{k+1} = \theta^k - t^k \nabla_\theta \hat{J}(\theta^k)$$

step size

stochastic gradient from simulation

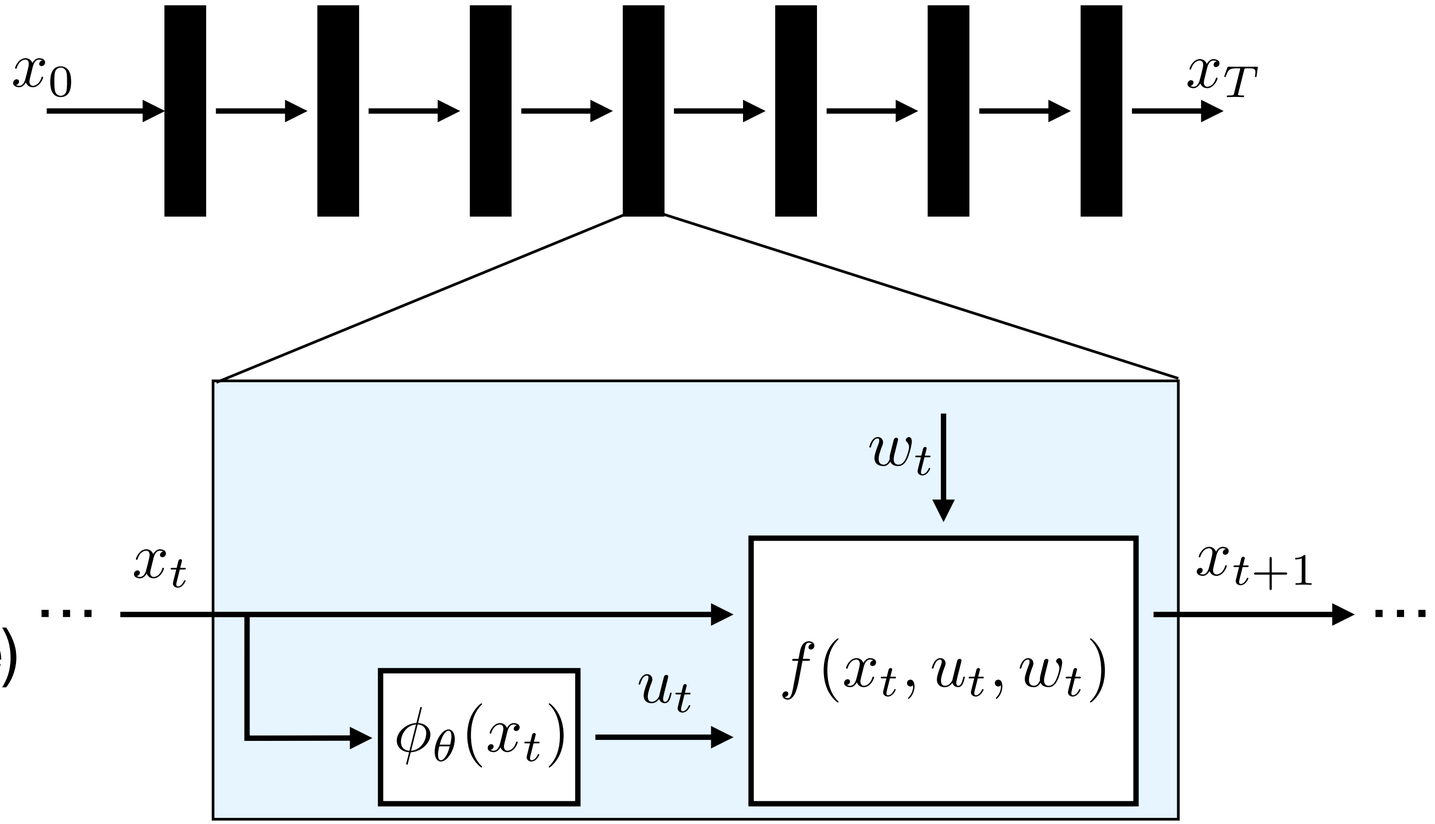
Generalization

Split simulation data in training, validation and testing

Non differentiable $\hat{J}(\theta)$?

Still get a descent direction (common in NN community)

Implementation



Automatic differentiation

- **Build** computation graph (simulate)
- **Backpropagate** using PyTorch

CVXPYLayers

Backpropagate through COCPs
(differentiate KKT optimality conditions)

[https://github.com/cvxgrp/cvxpylayers/]

[Differentiable convex optimization layers. Agrawal, Amos, Barratt, Boyd, Diamond, and Kolter. NeurIPS 2019]

[Differentiable optimization-based modeling for machine learning. Amos. PhD thesis 2019]

Box-constrained LQR

Problem setup

- dynamics: $x_{t+1} = Ax_t + Bu_t + w_t$
- actuator limit: $\|u_t\|_\infty \leq 1$
- stage cost: $x_t^T Q x_t + u_t^T R u_t$

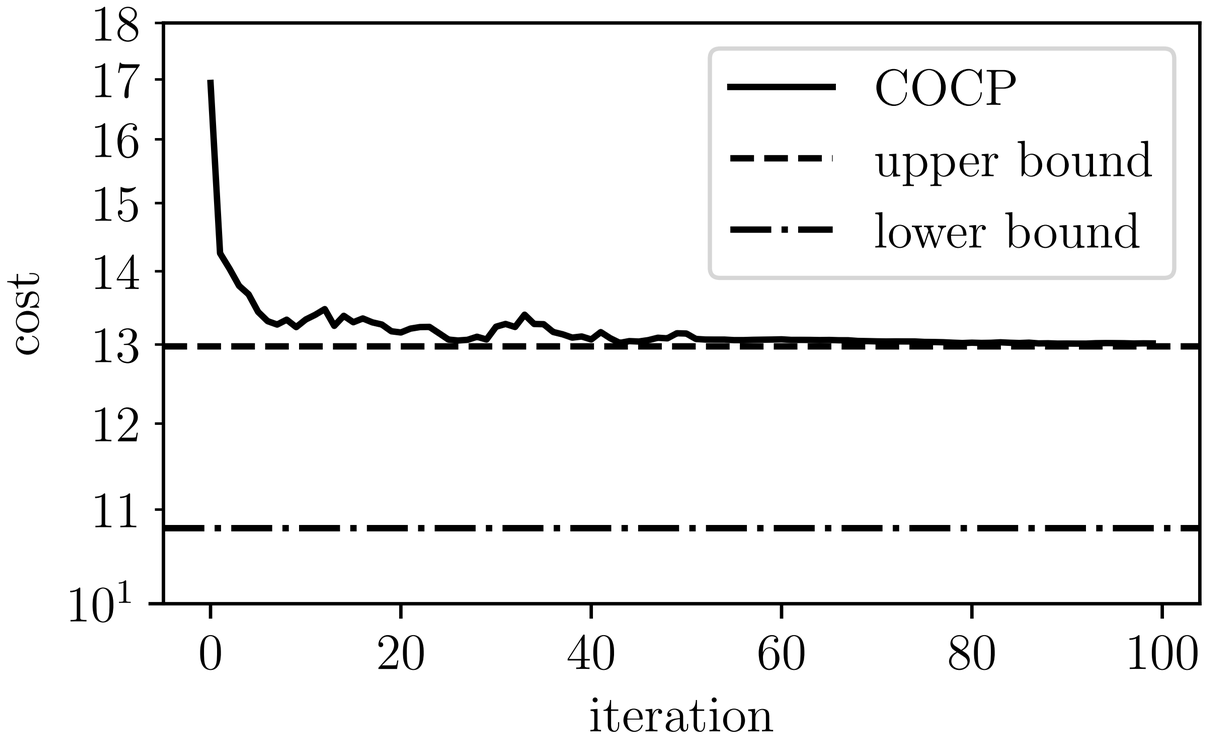
COCP Policy (QP)

$$u_t = \begin{array}{ll} \underset{u}{\text{argmin}} & u^T R u + \|\theta(Ax_t + Bu)\|_2^2 \\ \text{subject to} & \|u\|_\infty \leq 1 \end{array}$$

parameters

Box-constrained LQR

Performance



Standard upper/lower bounds from SDPs

↓

Hard to generalize
(other dynamics, disturbances, etc)

Vehicle tracking curved paths

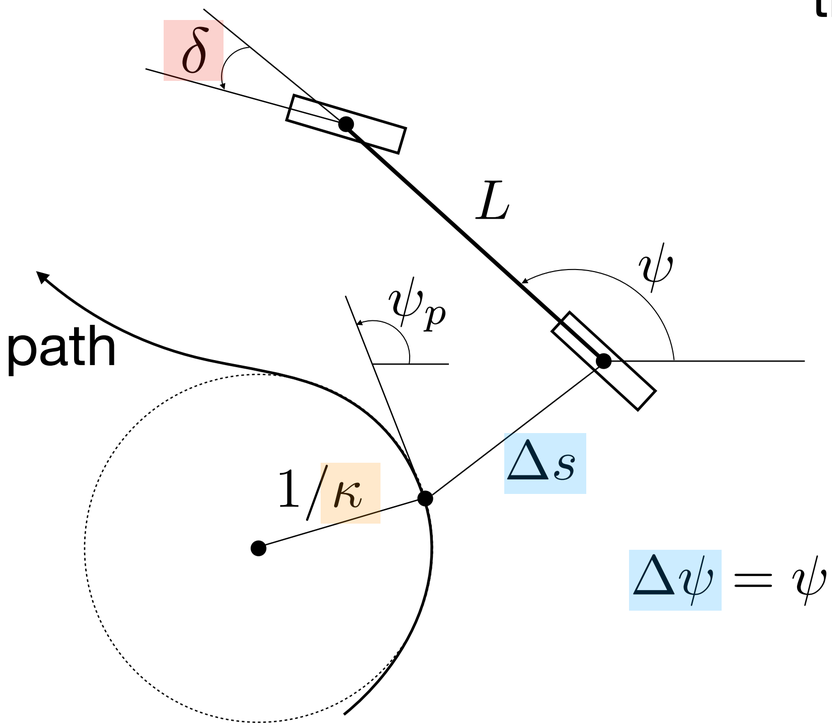
lateral deviation

heading deviation

path curvature

State: $x_t = (\Delta s_t, \Delta\psi_t, v_t, v_t^{\text{des}}, \kappa_t)$

true and desired velocities



$\Delta\psi = \psi - \psi_p$

acceleration

Input: $u_t = (a_t, z_t)$

turn

$z_t = \tan(\delta_t) - L\kappa_t$

Dynamics

$x_{t+1} = f(x_t, u_t, w_t)$

Vehicle tracking curved paths

Cost and constraints

track velocity

Stage cost

small control effort

$$(v_t - v_t^{\text{des}})^2 + \Delta s_t^2 + \Delta \psi_t^2 + \lambda(|a_t| + z_t^2)$$

track path

Constraints

maximum acceleration

$$|a_t| \leq a_{\max}$$

maximum turn

$$|z + L\kappa_t| \leq \tan(\delta_{\max})$$

Vehicle tracking curved paths

COCP as a Convex QP

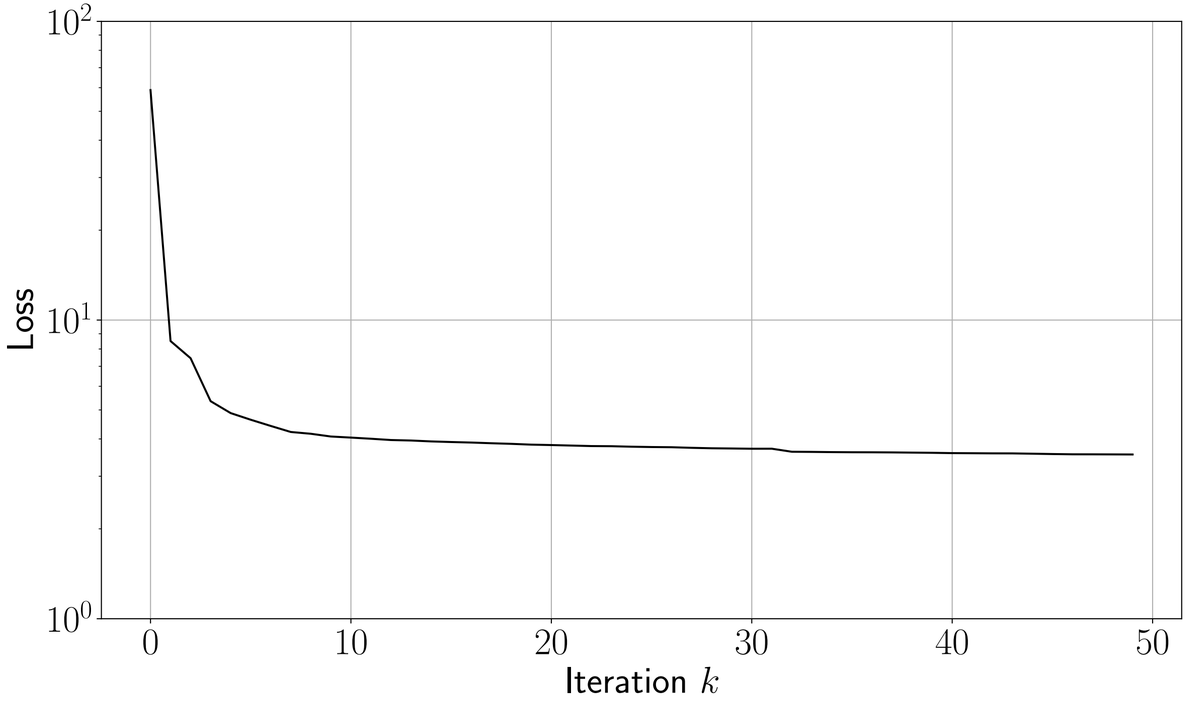
parameters

$$
\begin{aligned}
u_t = (a_t, z_t) = \phi(x_t) = \quad & \underset{u}{\text{argmin}} && |a| + z^2 + \|Sy\|_2^2 + q^T y \\
& \text{subject to} && x_t = (\Delta s_t, \Delta\psi_t, v_t, v_t^{\text{des}}, \kappa_t) \\
& && u = (a, z) \\
\text{next state} \longrightarrow & && y = f(x_t, u, 0) \\
& && |a| \leq a_{\max} \\
& && |z + L\kappa_t| \leq \tan(\delta_{\max})
\end{aligned}
$$

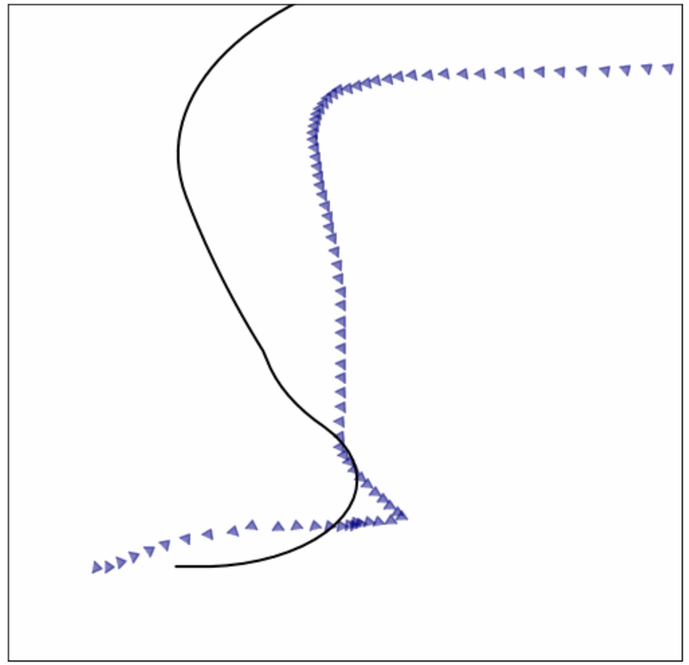
Vehicle tracking curved paths

Results

Validation loss



Tracking behavior



Good trajectory in very few iterations

Learning COCPs summary

Interpretable

Satisfy constraints

Handle varying dynamics

Efficient and reliable (even division-free: OSQP)

Easy to tune from data

Future work

- Hybrid (mixed-integer) control policies
- Integrate tuning and deployment with code generation (e.g., OSQP)
- Stochastic policies with safety-guarantees

Conclusions

Acknowledgements

Goran Banjac

Paul Goulart

Alberto Bemporad

Ken Goldberg

Stephen Boyd

Akshay Agrawal

Shane Barratt

Jeff Ichnowski

Paras Jain

Francesco Borrelli

References

OSQP (osqp.org)

[OSQP: An Operator Splitting Solver for Quadratic Programs. Stellato, Banjac, Goulart, Bemporad, and Boyd. Mathematical Programming Computation 2020]

[Infeasibility detection in the alternating direction method of multipliers for convex optimization. Banjac, Goulart, Stellato, and Boyd. Journal of Optimization Theory and Applications 2019]

[Embedded code generation using the OSQP solver. Stellato, Banjac, Stellato, Moehle, Goulart, Bemporad, and Boyd. IEEE Conf. on Decision and Control 2017]

[Embedded mixed-integer quadratic optimization using the OSQP solver. Stellato, Naik, Bemporad, Goulart, and Boyd. European Control Conference, 2018]

[Accelerating Quadratic Optimization with Reinforcement Learning. Ichnowski, Jain, Stellato, Banjac, Luo, Gonzalez, Stoica, Borrelli, and Goldberg. (in prep) 2021]

Learning COCPs (https://github.com/cvxgrp/cocp)

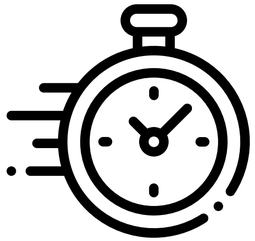
[Learning Convex Optimization Control Policies. Agrawal, Amos, Barratt, Boyd, and Stellato. L4DC 2020]

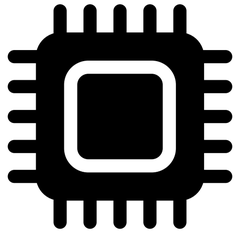
[Differentiable convex optimization layers. Agrawal, Amos, Barratt, Boyd, Diamond, and Kolter. NeurIPS 2019]

[Differentiable optimization-based modeling for machine learning. Amos. PhD thesis 2019]

Conclusions

Real-time and embedded optimization



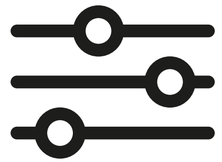


will soon **be applied everywhere**

Thanks to

Efficient and reliable optimizers

Easy-to-tune control policies




stellato.io


bstellato@princeton.edu


@b_stellato


github.com/bstellato

Backup

Constraint-wise Deep Deterministic Policy Gradient

Algorithm 2 Collaborative-agent DDPG for (ρ vector)

Input: exploration noise σ, buffer size $\texttt{rs}$
$\pi, Q \leftarrow$ initialize policy and critic (see DDPG)
$\mathcal{D} \leftarrow$ replay buffer w/ ($\texttt{rs}\times$(mean constraint count))
$\texttt{env}, \mathbf{s}^{(0)}, m \leftarrow$ new QP, its state, constraint count
for $t \in \{0, \ldots, T\}$ **do**
  $\rho_i^{(t)} \leftarrow \pi(s_i^{(t)}) + \epsilon,\ \epsilon \sim \mathcal{N}(0, \sigma)$ for all $i \in [1\ ..\ m]$
  $\mathbf{s}^{(t+1)}, r^{(t)}, \texttt{done}^{(t)} \leftarrow \texttt{step}(\texttt{env}, \mathbf{s}^{(t)}, \boldsymbol{\rho}^{(t)})$
  store $(s_i^{(t)}, \rho_i^{(t)}, r^{(t)}, s_i^{(t+1)})$ in $\mathcal{D}$ for all $i \in [1\ ..\ m]$
  if $\texttt{done}^{(t)}$ **then**
    $\texttt{env}, \mathbf{s}^{(t)}, m \leftarrow$ new QP, its state, constraint count
  end if
  update π and Q using data sampled from $\mathcal{D}$
end for