Real-time Decision-Making via Data-Driven Optimization

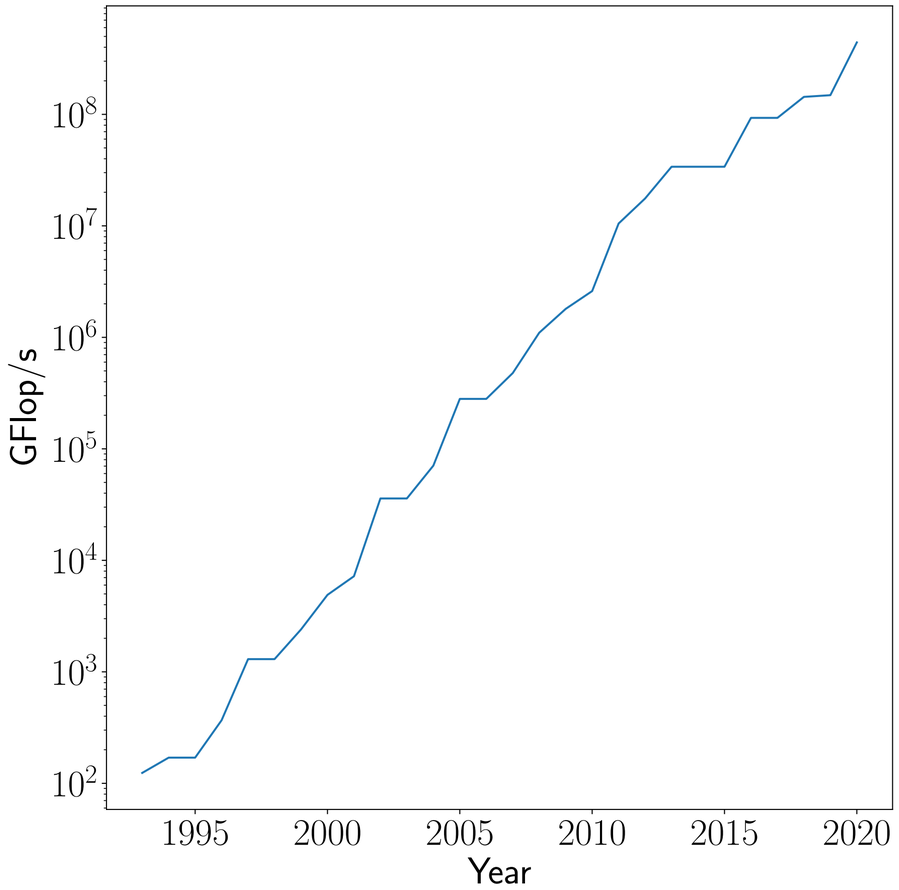
PRINCETON UNIVERSITY

ORFE

Bartolomeo Stellato — Cornell Tech, March 21 2022

Tremendous progress in optimization

Top500 peak CPU power



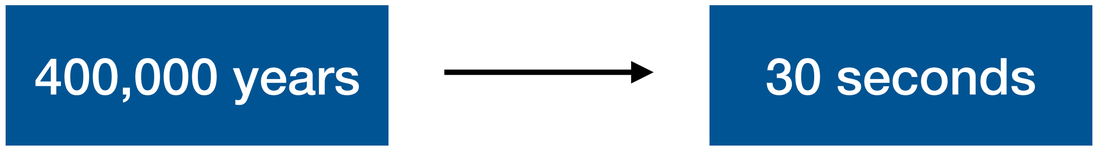
Hardware + Software

400 billion times speedups!

400,000 years → 30 seconds

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

Is it enough?

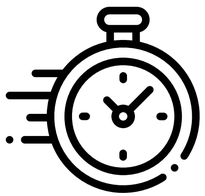


Is it enough?

400,000 years → 30 seconds

Robotics

< 10 milliseconds

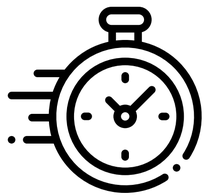


Is it enough?

400,000 years → 30 seconds

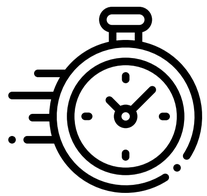
Robotics

< 10 milliseconds



High-Frequency Trading

< 1 millisecond



Same problem with varying data

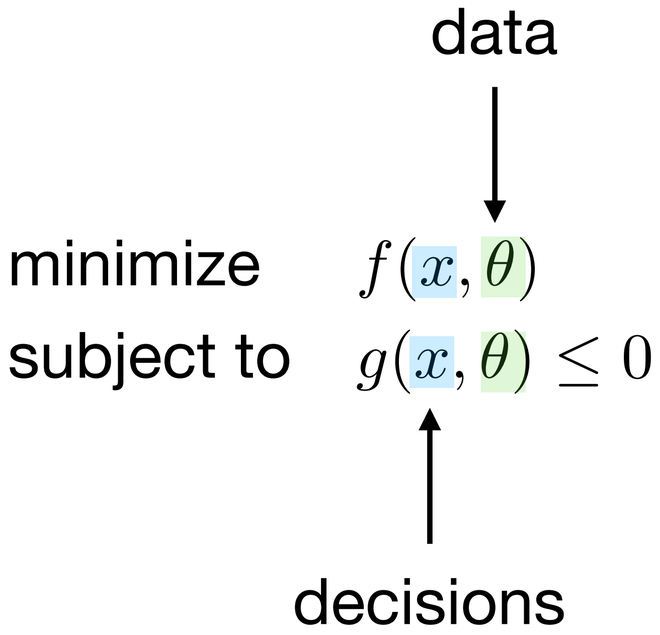
$$\begin{aligned} \text{minimize} \quad & f(x, \theta) \\ \text{subject to} \quad & g(x, \theta) \leq 0 \end{aligned}$$

Same problem with varying data

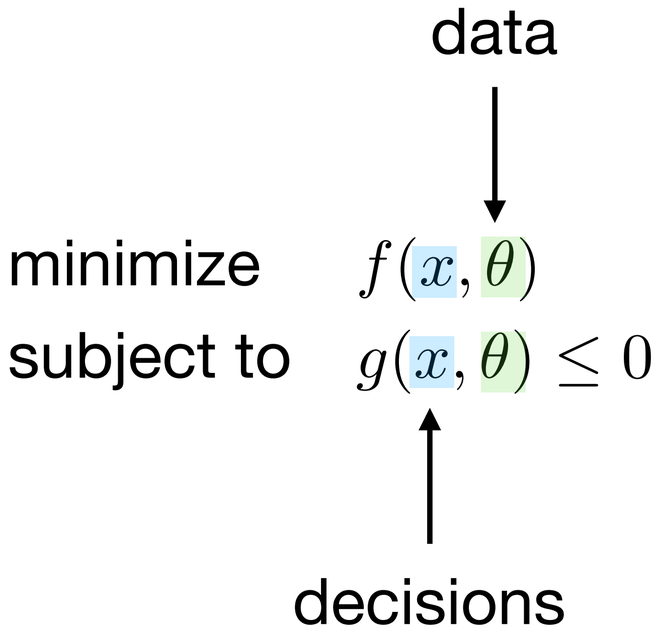
$$\begin{aligned} \text{minimize} \quad & f(x, \theta) \\ \text{subject to} \quad & g(x, \theta) \leq 0 \end{aligned}$$

decisions

Same problem with varying data



Same problem with varying data



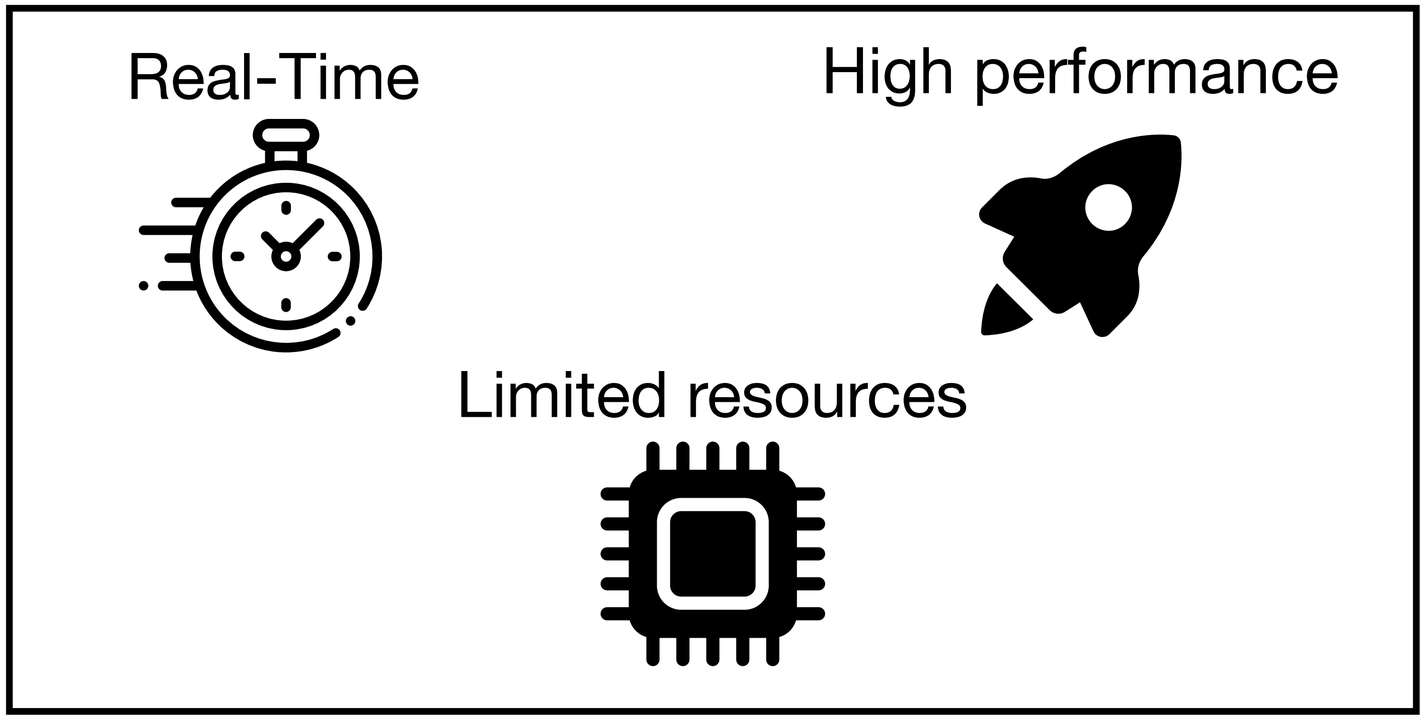
Can we solve it in **milliseconds** *or* **microseconds***?*

Challenges in real-time optimization

Extreme reliability

Challenges in real-time optimization

Extreme reliability



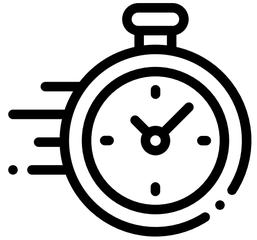
Today's talk

Real-time Decision-Making via Data-Driven Optimization

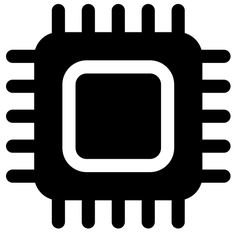
OSQP
Solver

Learning
Convex Optimization
Control Policies

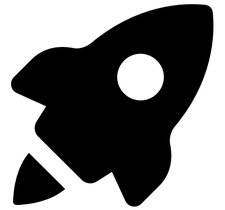
Real-Time



Limited resources



Performance

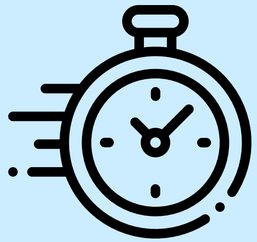


Today's talk

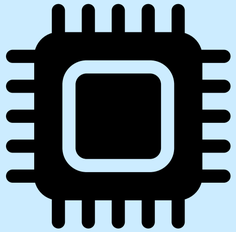
Real-time Decision-Making via Data-Driven Optimization

OSQP
Solver

Real-Time

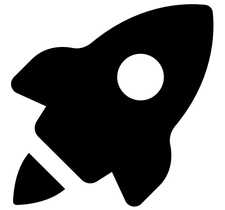


Limited resources

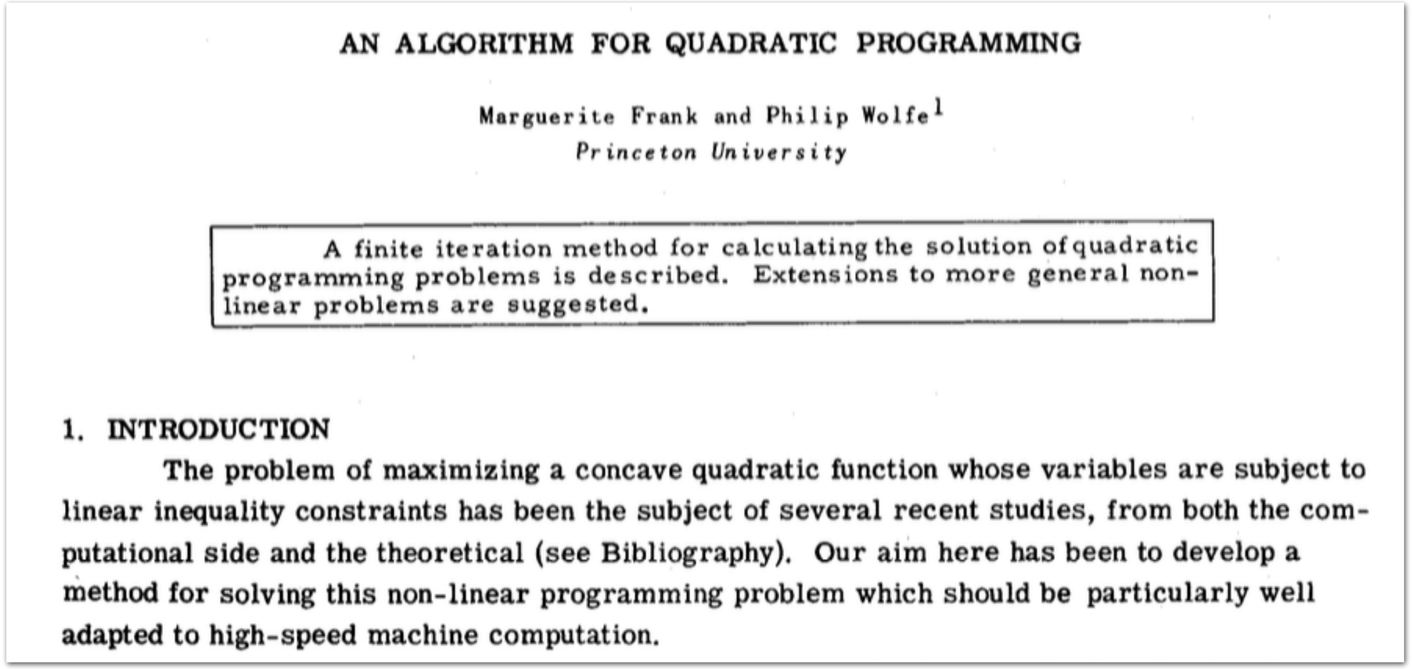


Learning
Convex Optimization
Control Policies

Performance



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

- Warm-starting
- Large-scale problems
- Embeddable

First-order methods

Wide popularity

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

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

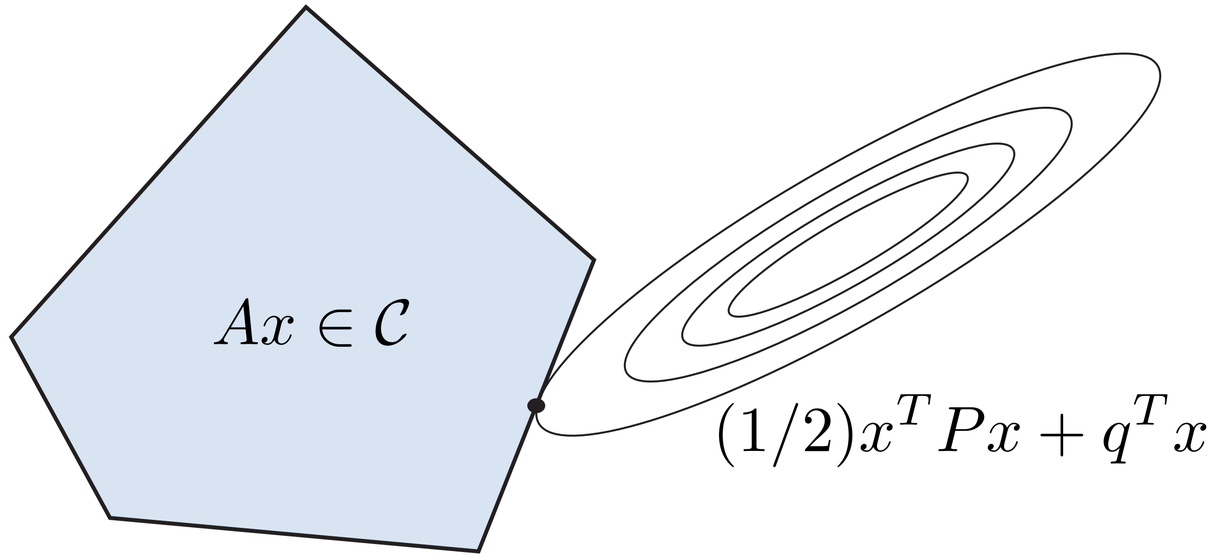
$$\begin{array}{ll} \text{minimize} & (1/2)x^T Px + q^T x \\ \text{subject to} & Ax \in \mathcal{C} \end{array}$$

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

minimize $f(\tilde{x}) + g(x)$

subject to $\tilde{x} = x$

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?

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

How do we split the QP?

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

How do we split the QP?

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

subject to $Ax = z$

$z \in \mathcal{C}$ g

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

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

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

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

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

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

ADMM iterations

Inner QP

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

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

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

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

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

Solving the inner QP

Equality-constrained

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

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 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 (robust)!

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)

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

Solving the linear system

Direct method (small to medium scale)

Quasi-definite matrix

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

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

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/osqp/qdldl]

Solving the linear system

Indirect method (large scale)

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

Solving the linear system

Indirect method (large scale)

Positive-definite matrix

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

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

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

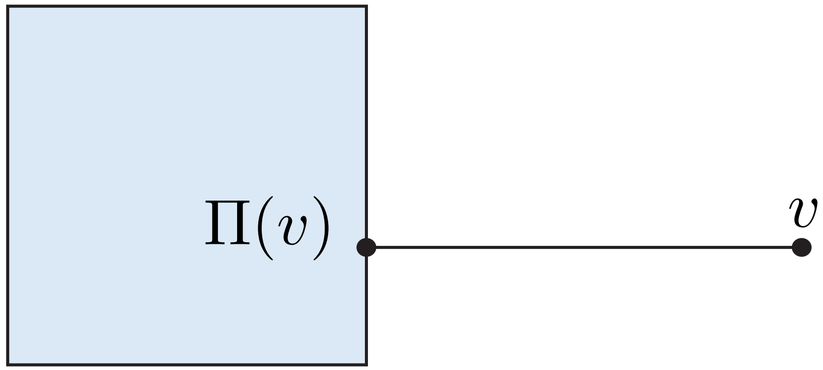
[https://github.com/osqp/cuosqp]

Computing the projection

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

Box projection

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



Complete algorithm

Problem

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

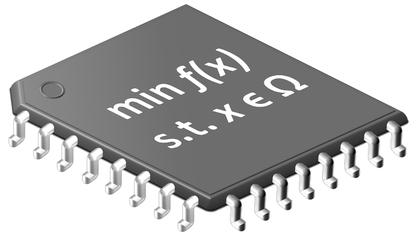
Code generation with OSQP

```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

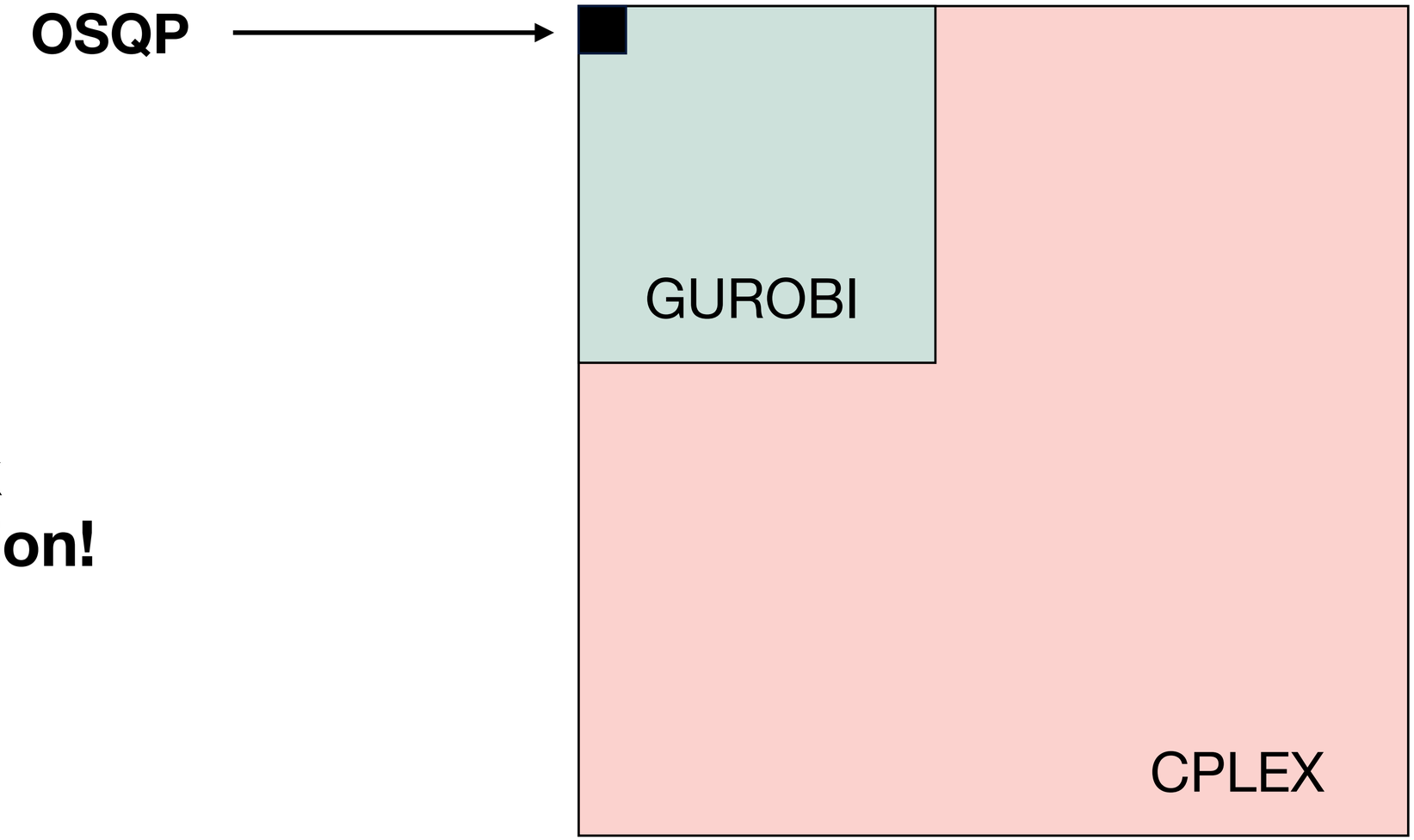
```c
// Main ADMM algorithm
for (iter = 1; iter <= work->settings->max_iter; iter ++) {
    // Update x_prev, z_prev (preallocated, no malloc)
    swap_vectors(&(work->x), &(work->x_prev));
    swap_vectors(&(work->z), &(work->z_prev));

    /* 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?

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

Primal residual

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

Dual residual

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

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

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

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

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

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 $Ax = b$

Solve in one ADMM step

$$\rho \approx \infty$$

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

Extreme cases

Equality constrained QP

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

subject to $Ax = b$

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

Solve in one ADMM step

$\rho \approx 0$

$$Px = -q$$

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 $(Ax^\star)_i = u_i$

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**

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 $u_i = \infty$

$\downarrow$

$\rho_i = 0$

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 $u_i = \infty$

$\downarrow$

$\rho_i = 0$

Always tight

$l_i = u_i \neq \infty$

$\downarrow$

$\rho_i = \infty$

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

$l_i = u_i \neq \infty$

$\downarrow$

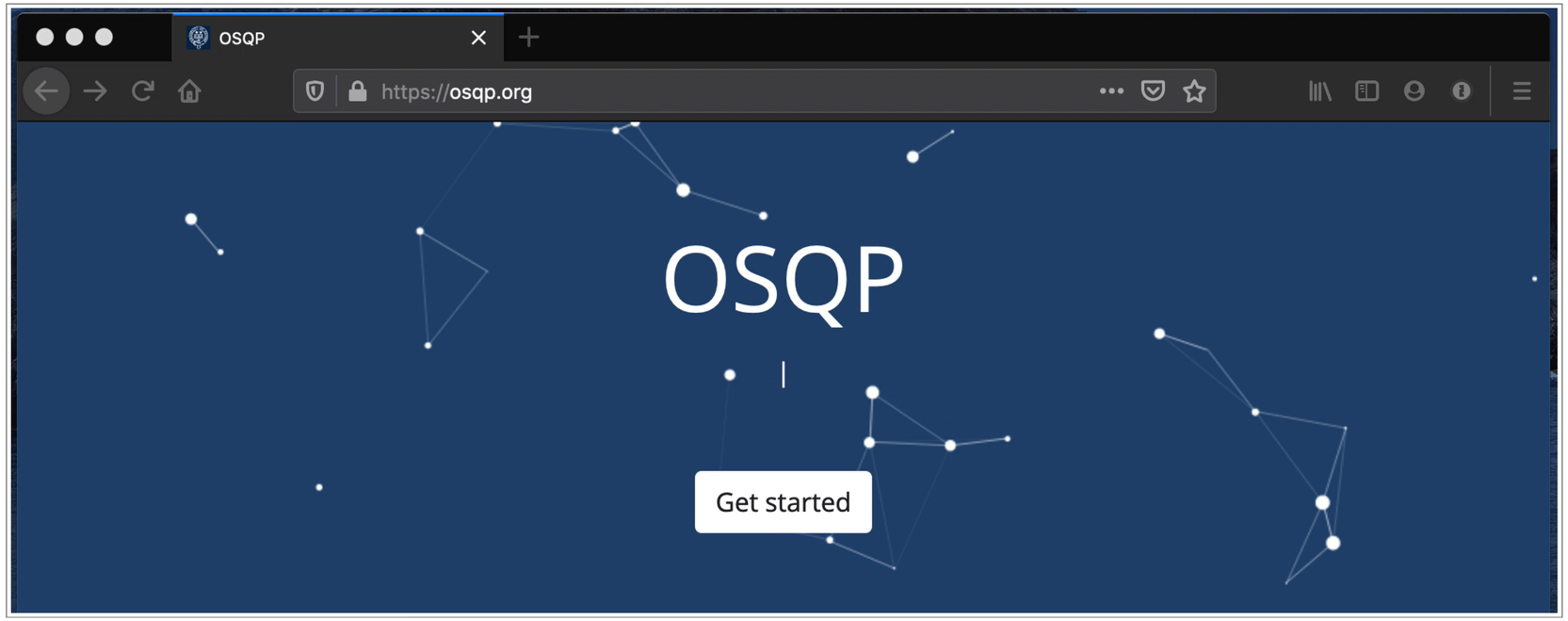
$\rho_i = \infty$

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!)

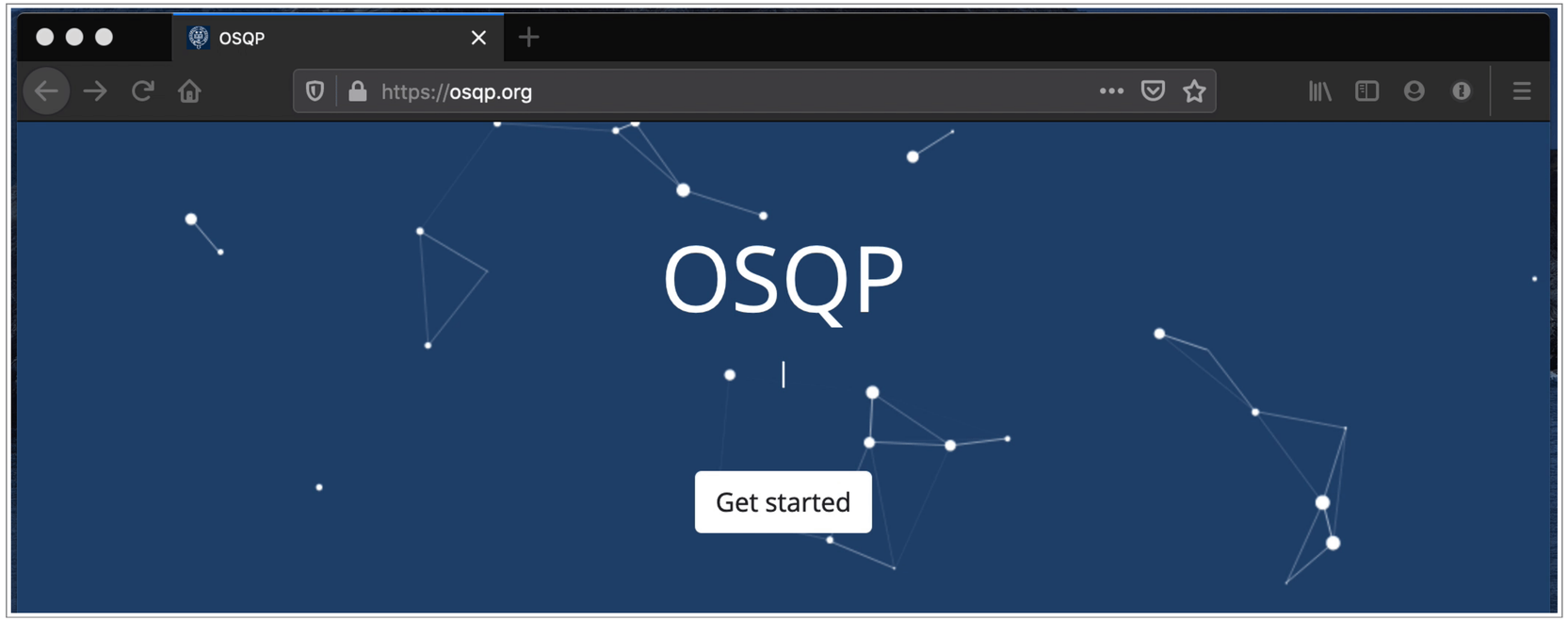
Supports
warm-starting

Detects
infeasibility

Solves large-scale
problems

OSQP

Operator Splitting solver for Quadratic Programs



Embeddable
(can be division free!)

Supports
warm-starting

Detects
infeasibility

Solves large-scale
problems

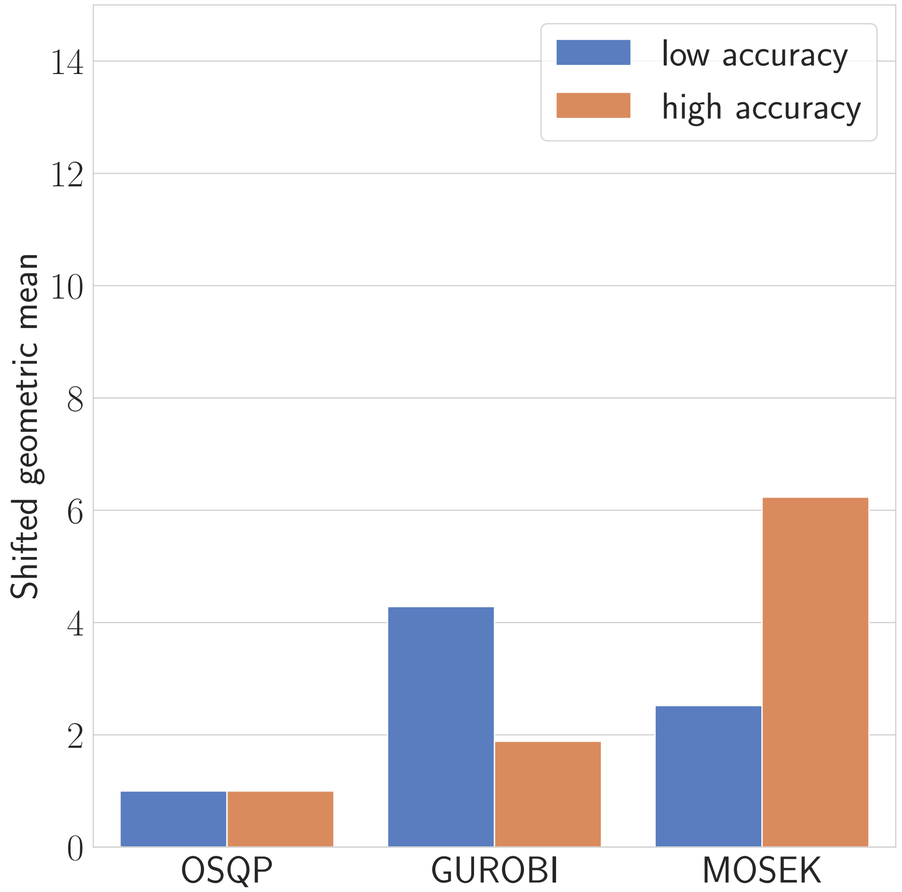
Users

More than 7 million downloads!

[pepy.tech/project/osqp]

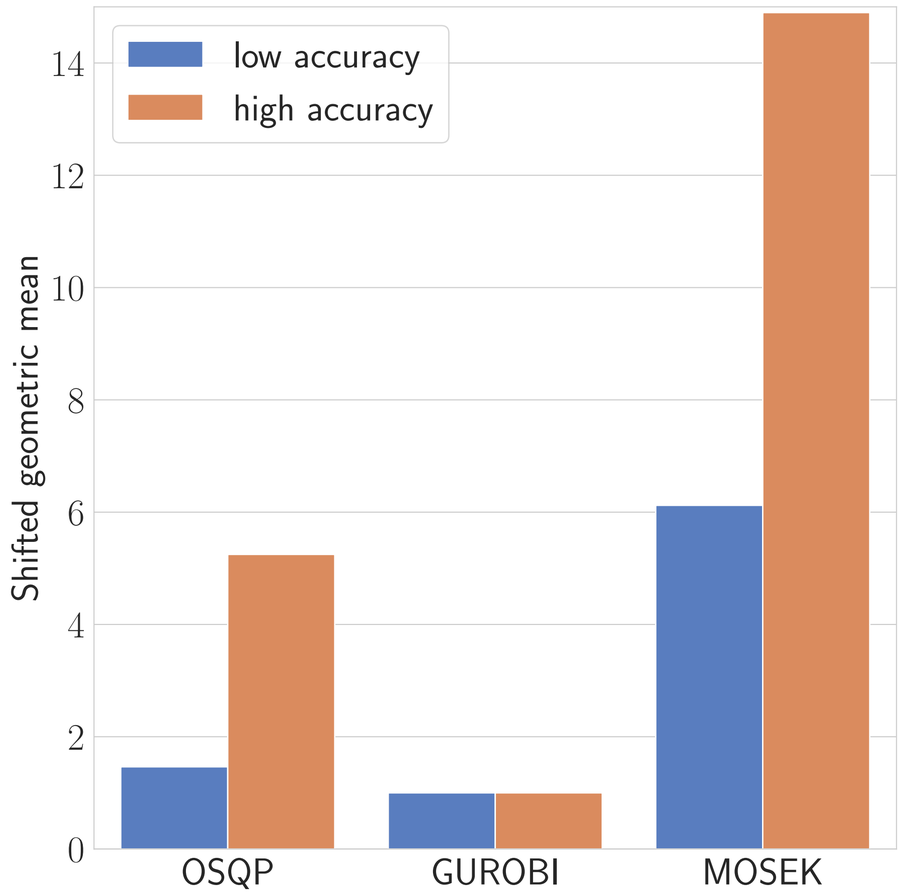
Performance benchmarks

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



[github.com/osqp/osqp_benchmarks]

Maroz-Meszaros



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 $(Ax^\star)_i = u_i$

Vector step size

$\rho = (\rho_1, \ldots, \rho_m)$

Never tight

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

↓

$\rho_i = 0$

Otherwise

↓

Balance residuals

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

Always tight

$l_i = u_i \neq \infty$

↓

$\rho_i = \infty$

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 $(Ax^\star)_i = u_i$

Vector step size

$\rho = (\rho_1, \ldots, \rho_m)$

Never tight

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

↓

$\rho_i = 0$

Otherwise

↓

Balance residuals

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

Always tight

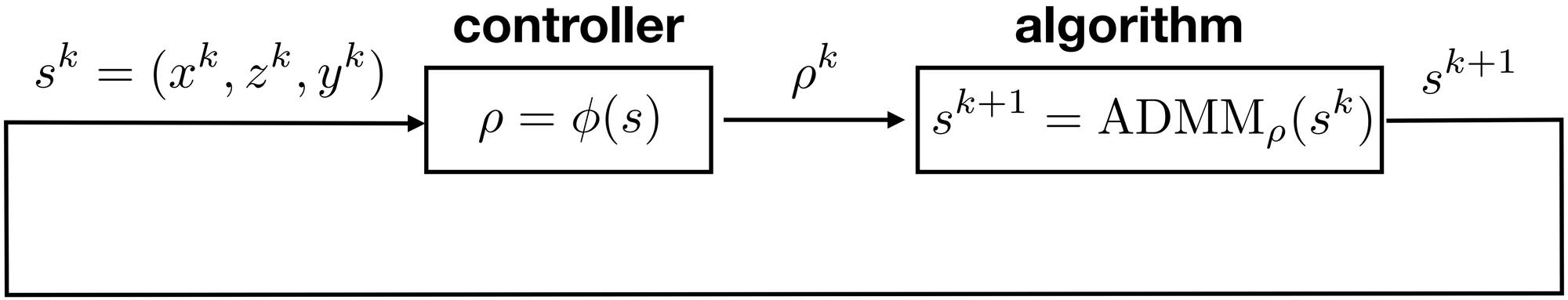
$l_i = u_i \neq \infty$

↓

$\rho_i = \infty$

Can we learn a better update rule from data?

Step size choice as a control problem



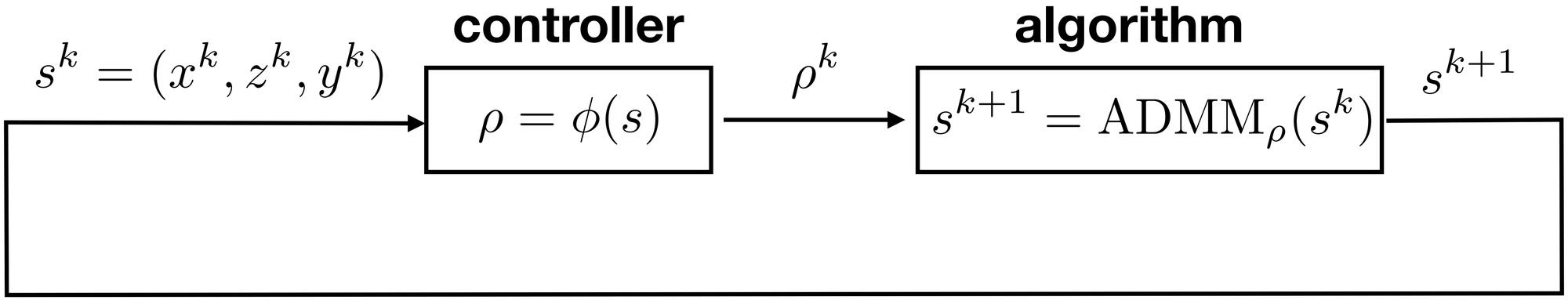
Stage cost

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

Cumulative cost

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

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
Deep Policy Gradient methods (TD3)

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

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

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

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

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

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

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

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

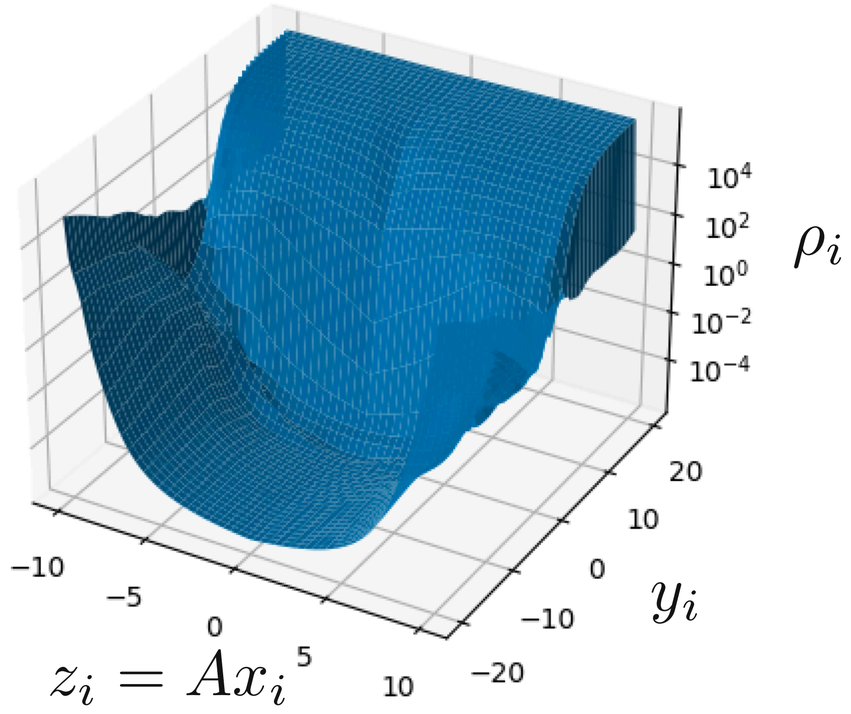
dual variable

Generalize to different dimensions

Low-dimensional state per constraint

Small NN policy $\phi_c(s_i)$

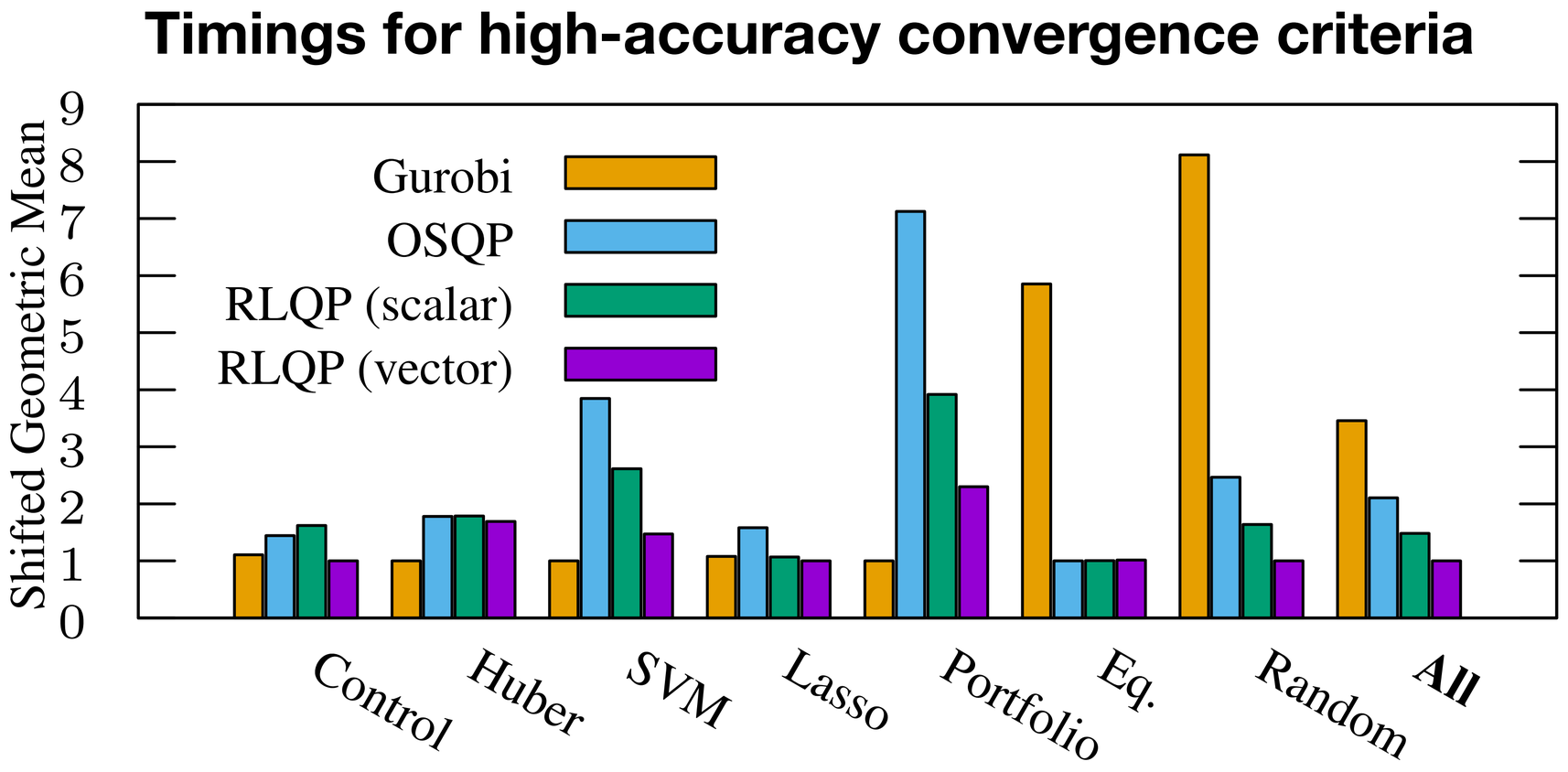
Visualize learned policy



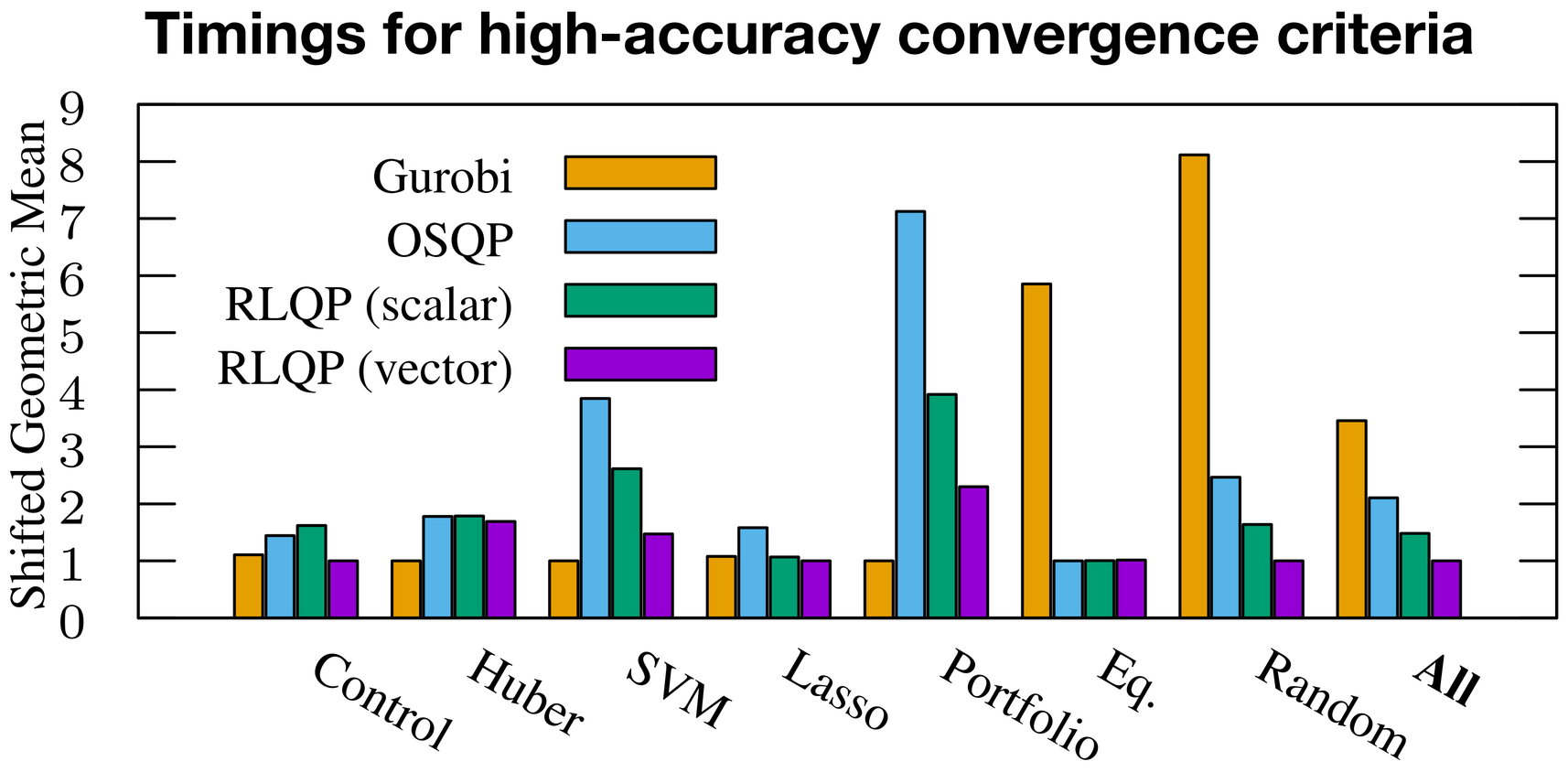
Interpretable policy

High step size ρ_i
when we reach bounds
$z_i \approx l_i$ and $z_i \approx u_i$.

Performance with step size learning



Performance with step size learning



Up to 3x faster than Gurobi

OSQP features

Features

- Robust
- Embeddable (can be division free!)
- Supports warm-starting
- Detects infeasibility
- Can improve with data

OSQP features

Features

Robust

Embeddable (can be division free!)

Supports warm-starting

Detects infeasibility

Can improve with data

Learning for optimization

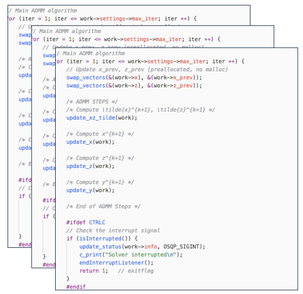
Integrate RL and ADMM to dynamically tune parameters

- Faster convergence
- Very low overhead
- Interpretable policy

OSQP 1.0 (this summer!)

Improved embedded code generation

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

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

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



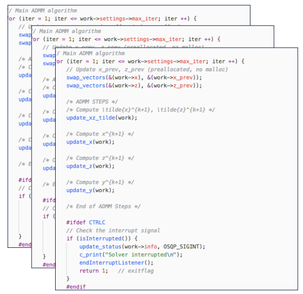
Embedded Hardware

- Code generation from C to C
- CVXPY integration

OSQP 1.0 (this summer!)

Improved embedded code generation

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

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

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



Embedded Hardware

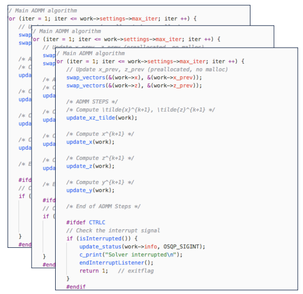
Differentiable layers

- Code generation from C to C
- CVXPY integration

OSQP 1.0 (this summer!)

Improved embedded code generation

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

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

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



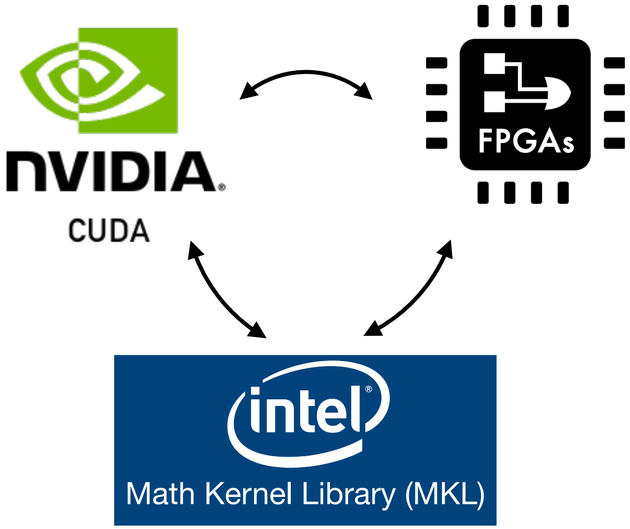
Embedded Hardware

- Code generation from C to C
- CVXPY integration

Differentiable layers

PyTorch

Modular linear algebra



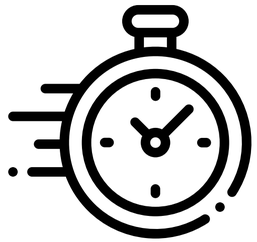
Today's talk

Real-time Decision-Making via Data-Driven Optimization

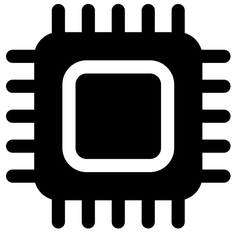
OSQP
Solver

Learning
Convex Optimization
Control Policies

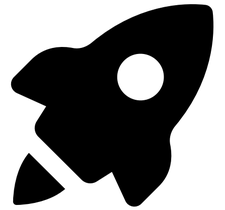
Real-Time



Limited resources



Performance

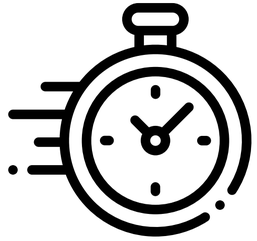


Today's talk

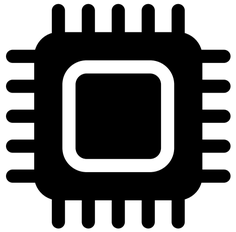
Real-time Decision-Making via Data-Driven Optimization

OSQP
Solver

Real-Time

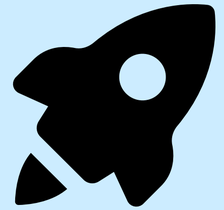


Limited resources

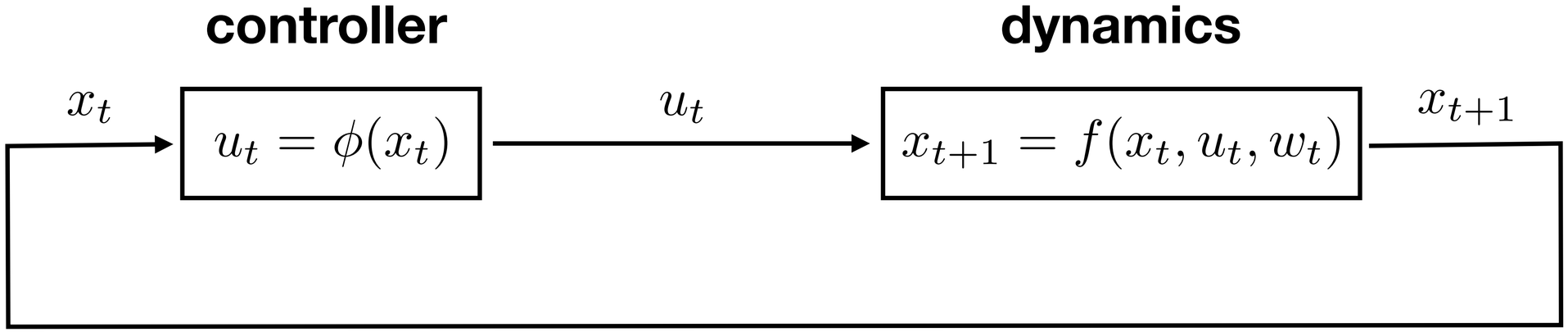


Learning
Convex Optimization
Control Policies

Performance



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

Explicit vs implicit control policies

Explicit

Complete control specification

Example: PI Controller

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

Explicit vs implicit control policies

Explicit

Complete control specification

Implicit (optimization-based)

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

Example: PI Controller

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

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

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

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

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

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***

Judging COCPs

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

Trajectories

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

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

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

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

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

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 $T \to \infty$

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

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 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

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

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 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

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

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

Learning scheme

Auto-tuning

Stochastic gradient descent

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

step size

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

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

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

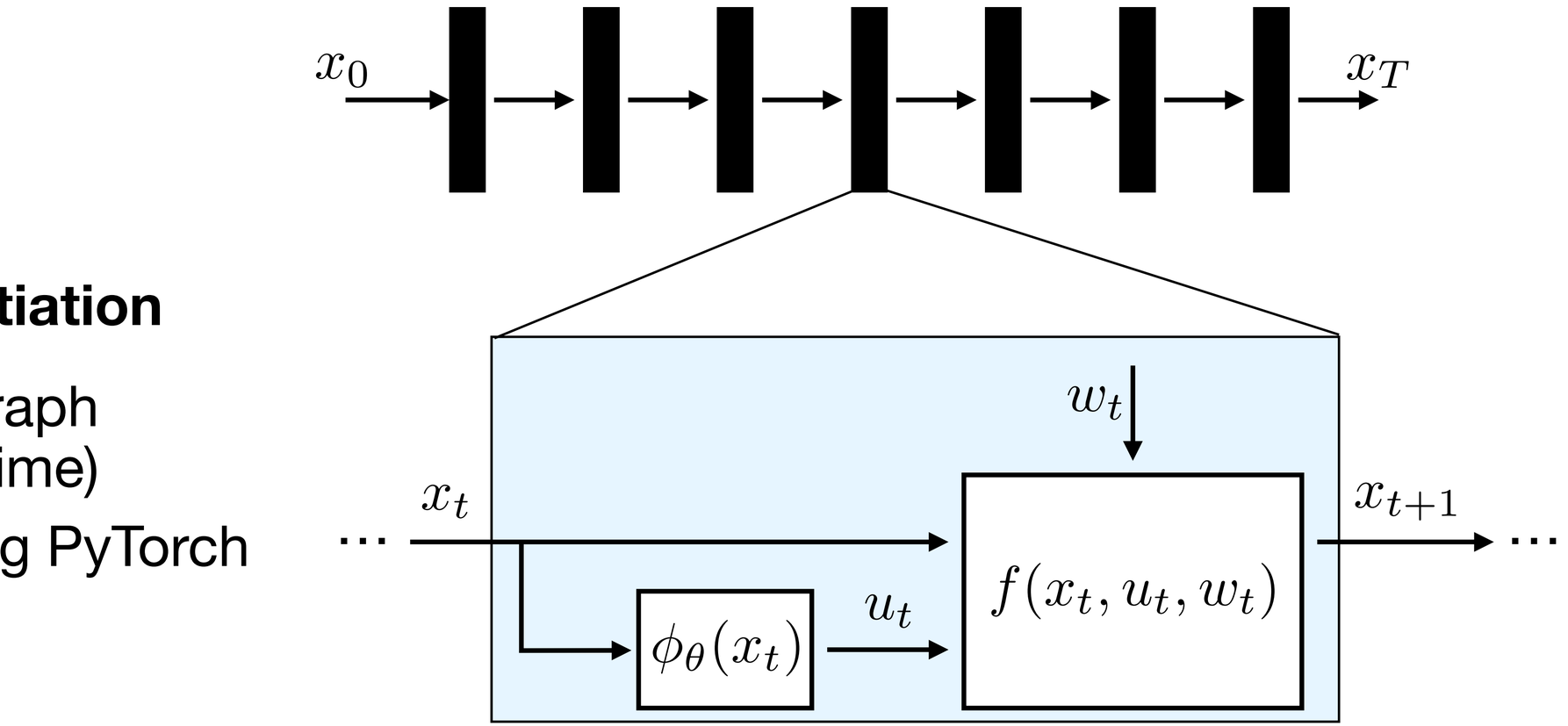
Non differentiable $\hat{J}(\theta)$?

Still get a descent direction (common in NN community)

Implementation

Automatic differentiation

- **Build** computation graph (simulate forward in time)
- **Backpropagate** using PyTorch



How do you backpropagate through ϕ_θ ?

Differentiating through convex optimization problems

$$\begin{array}{ll} \text{minimize} & f(x, \theta) \\ \text{subject to} & g(x, \theta) \leq 0 \end{array}$$

- x variable
- θ parameter

Differentiating through convex optimization problems

$$\begin{array}{ll} \text{minimize} & f(x,\theta) \\ \text{subject to} & g(x,\theta) \leq 0 \end{array}$$

- x variable
- θ parameter

Optimality conditions

$$\begin{aligned} \nabla_x f(x,\theta) + D_x g(x,\theta)^T y &= 0 \\ \mathbf{diag}(y)g(x,\theta) &= 0 \\ g(x^\star,\theta) &\leq 0 \\ y^\star &\geq 0 \end{aligned}$$

Differentiating through convex optimization problems

$$\begin{array}{ll} \text{minimize} & f(x,\theta) \\ \text{subject to} & g(x,\theta) \leq 0 \end{array}$$

- x variable
- θ parameter

Optimality conditions

stationarity →

$$\nabla_x f(x,\theta) + D_x g(x,\theta)^T y = 0$$

$$\mathbf{diag}(y)g(x,\theta) = 0$$

$$g(x^\star,\theta) \leq 0$$

$$y^\star \geq 0$$

Differentiating through convex optimization problems

$$\begin{array}{ll} \text{minimize} & f(x,\theta) \\ \text{subject to} & g(x,\theta) \leq 0 \end{array}$$

- x variable
- θ parameter

Optimality conditions

stationarity → $\nabla_x f(x,\theta) + D_x g(x,\theta)^T y = 0$

complementary slackness → $\mathbf{diag}(y)g(x,\theta) = 0$

$$g(x^\star,\theta) \leq 0$$

$$y^\star \geq 0$$

Differentiating through convex optimization problems

minimize $f(x, \theta)$
subject to $g(x, \theta) \leq 0$

- x variable
- θ parameter

Optimality conditions

stationarity → $\nabla_x f(x, \theta) + D_x g(x, \theta)^T y = 0$

complementary slackness → $\mathbf{diag}(y) g(x, \theta) = 0$

$g(x^\star, \theta) \leq 0$

$y^\star \geq 0$

→ $F(z^\star(\theta), \theta) = 0$

primal/dual solution

$z^\star = (x^\star(\theta), y^\star(\theta))$

Differentiating through convex optimization problems

minimize $f(x, \theta)$
subject to $g(x, \theta) \leq 0$

- x variable
- θ parameter

Optimality conditions

stationarity → $\nabla_x f(x, \theta) + D_x g(x, \theta)^T y = 0$

complementary slackness → $\mathbf{diag}(y) g(x, \theta) = 0$

$g(x^\star, \theta) \leq 0$

$y^\star \geq 0$

→ $F(z^\star(\theta), \theta) = 0$

primal/dual solution
$z^\star = (x^\star(\theta), y^\star(\theta))$

Goal

Compute $Dz^\star(\theta)$

Differentiating through convex optimization problems

$$F(z^\star(\theta), \theta) = 0$$

primal/dual solution

$$z^\star = (x^\star(\theta), y^\star(\theta))$$

Differentiating through convex optimization problems

$$F(z^\star(\theta), \theta) = 0$$

primal/dual solution

$$z^\star = (x^\star(\theta), y^\star(\theta))$$

Implicit function theorem

$$D_z F(z^\star, \theta) Dz^\star(\theta) + D_\theta F(z^\star, \theta) = 0$$

Differentiating through convex optimization problems

$$F(z^\star(\theta), \theta) = 0$$

primal/dual solution

$$z^\star = (x^\star(\theta), y^\star(\theta))$$

Implicit function theorem

$$D_z F(z^\star, \theta) Dz^\star(\theta) + D_\theta F(z^\star, \theta) = 0$$

$$\downarrow$$

$$Dz^\star(\theta) = -\left(D_z F(z^\star, \theta)\right)^{-1} D_\theta F(z^\star, \theta)$$

($D_z F(z^\star, \theta)$ must be invertible)

one linear system solution

Differentiating through convex optimization problems

$$F(z^\star(\theta), \theta) = 0$$

primal/dual solution

$$z^\star = (x^\star(\theta), y^\star(\theta))$$

Implicit function theorem

$$D_z F(z^\star, \theta) Dz^\star(\theta) + D_\theta F(z^\star, \theta) = 0$$

$$\downarrow$$

$$Dz^\star(\theta) = -\left(D_z F(z^\star, \theta)\right)^{-1} D_\theta F(z^\star, \theta)$$

($D_z F(z^\star, \theta)$ must be invertible)

one linear system solution

We plug $Dz^\star(\theta)$ in AD (automatic differentiation)

PyTorch

Box-constrained LQR

Problem setup

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

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)

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

Box-constrained LQR

Problem setup

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

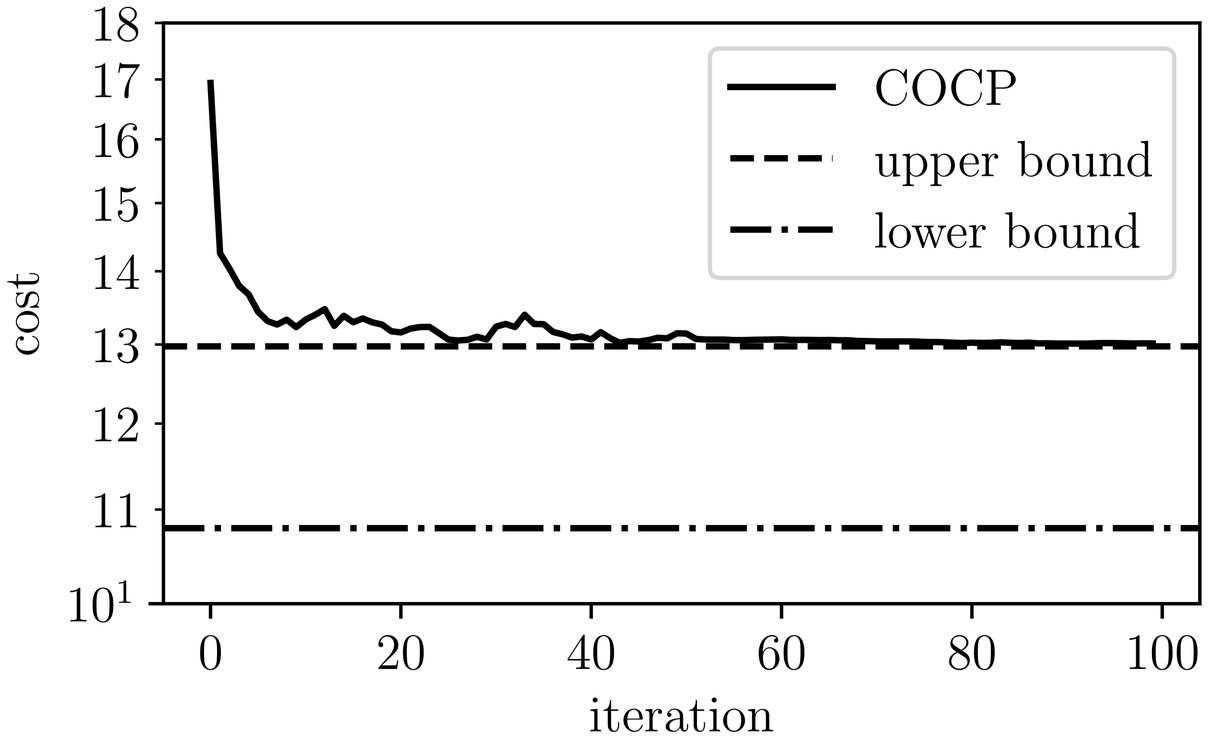
COCP Policy (QP)

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

parameters

Box-constrained LQR

Performance

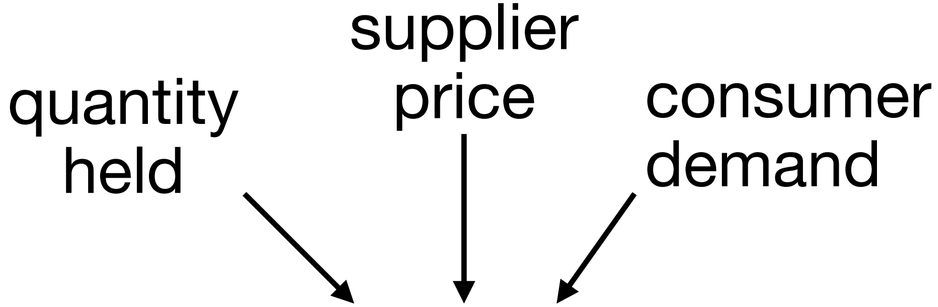


Standard upper/lower bounds from SDPs

↓

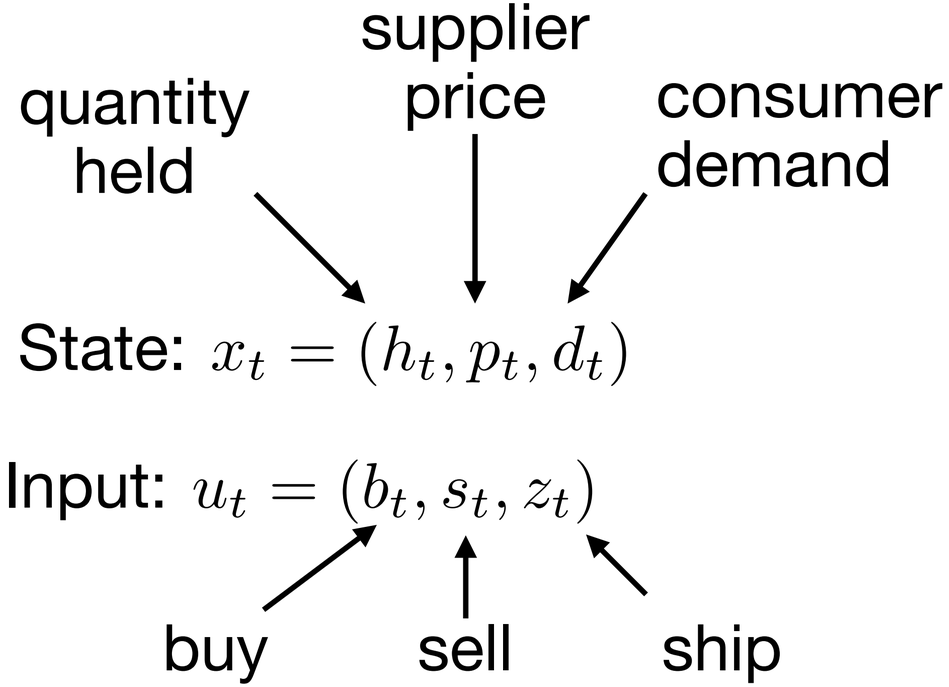
Hard to generalize
(other dynamics, disturbances, etc)

Supply chain distribution

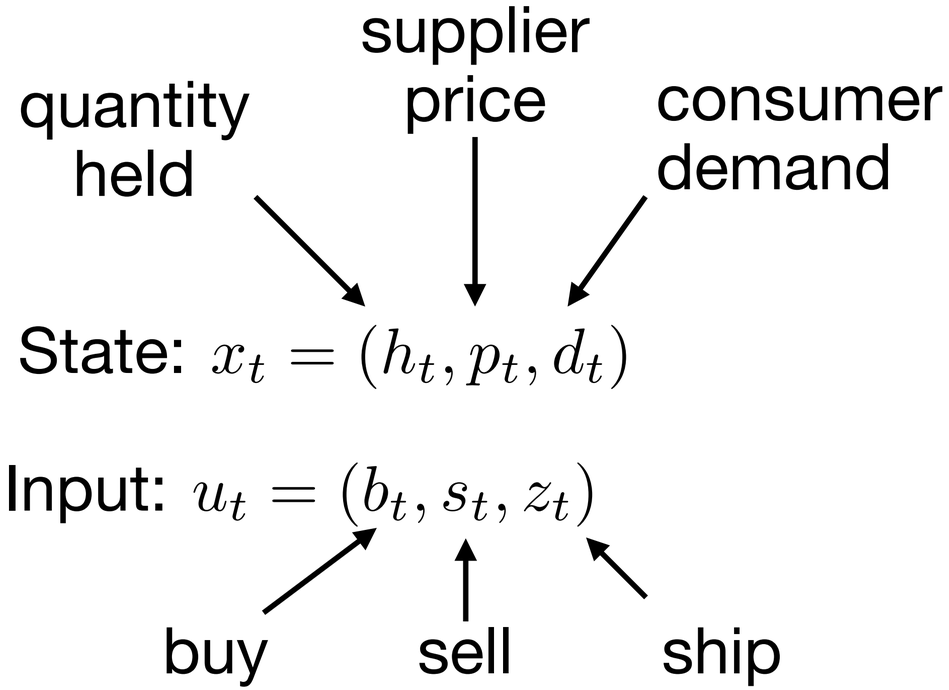


State: $x_t = (h_t, p_t, d_t)$

Supply chain distribution



Supply chain distribution



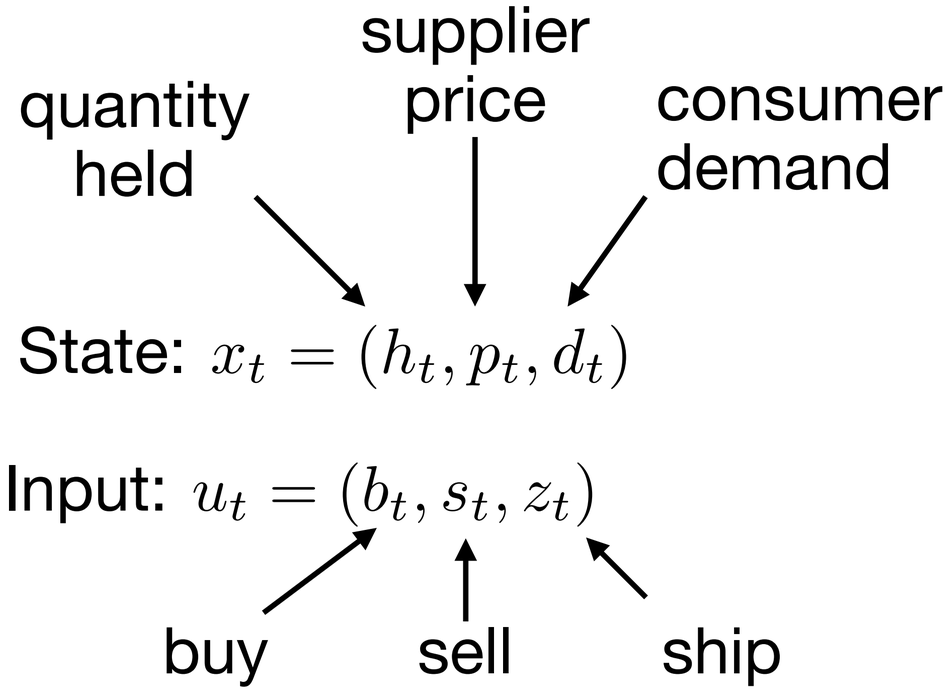
Dynamics

$$h_{t+1} = h_t + (A^{\text{in}} - A^{\text{out}})u_t$$

p_{t+1} and d_{t+1} are log-normal

$$A_{ij}^{\text{in(out)}} = \begin{cases} 1 & \text{if link } j \text{ enters (exits) node } i \\ 0 & \text{otherwise} \end{cases}$$

Supply chain distribution



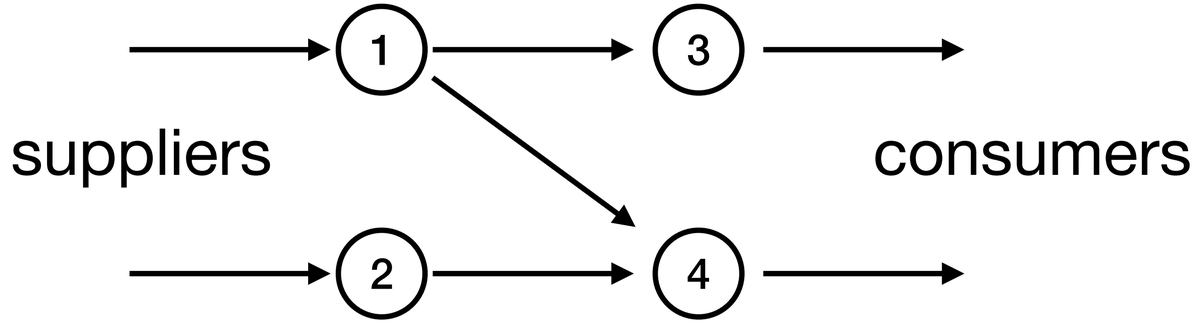
Dynamics

$$h_{t+1} = h_t + (A^{\text{in}} - A^{\text{out}})u_t$$

p_{t+1} and d_{t+1} are log-normal

$$A_{ij}^{\text{in(out)}} = \begin{cases} 1 & \text{if link } j \text{ enters (exits) node } i \\ 0 & \text{otherwise} \end{cases}$$

Network example



Supply chain distribution

Cost and constraints

Stage cost

$$p_t^T b_t - r^T s_t + \tau^T z_t + \alpha^T h_t + \beta^T h_t^2 + \mathcal{I}(x_t, u_t)$$

Supply chain distribution

Cost and constraints

sale revenues

Stage cost

$$p_t^T b_t - r^T s_t + \tau^T z_t + \alpha^T h_t + \beta^T h_t^2 + \mathcal{I}(x_t, u_t)$$

suppliers payment

shipment cost

Supply chain distribution

Cost and constraints

sale revenues

Stage cost

$$p_t^T b_t - r^T s_t + \tau^T z_t + \alpha^T h_t + \beta^T h_t^2 + \mathcal{I}(x_t, u_t)$$

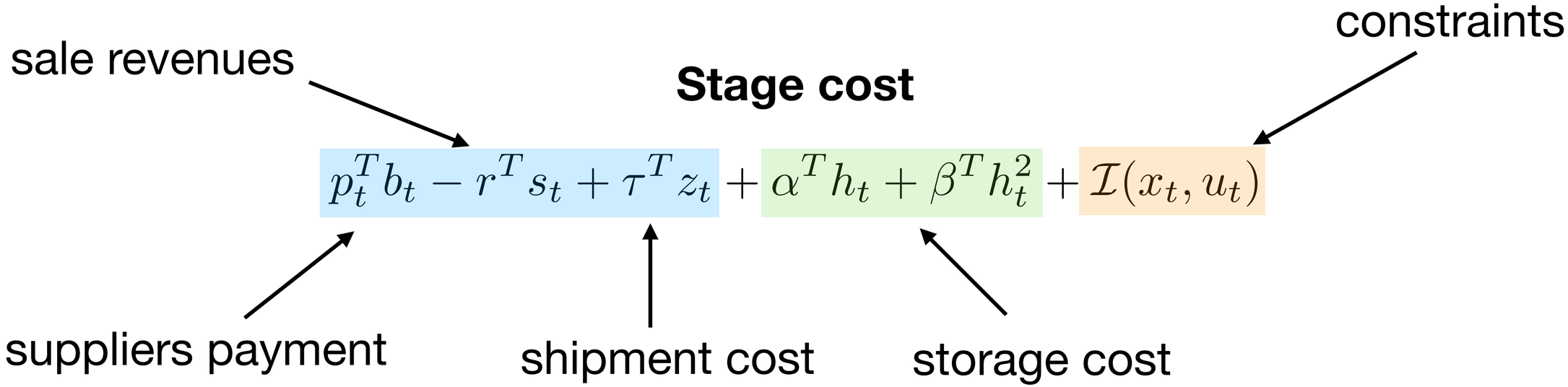
suppliers payment

shipment cost

storage cost

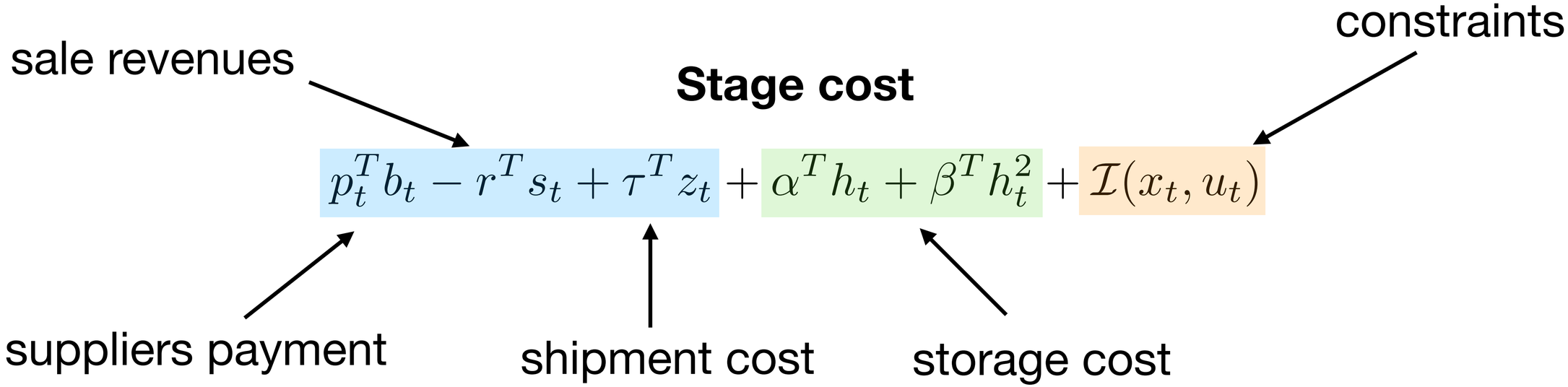
Supply chain distribution

Cost and constraints



Supply chain distribution

Cost and constraints



Constraints

$$0 \leq h_t \leq h_{\max}, \qquad 0 \leq u_t \leq u_{\max}$$

$$A^{\text{out}} u_t \leq h_t, \qquad s \leq d_t$$

Supply chain distribution COCP

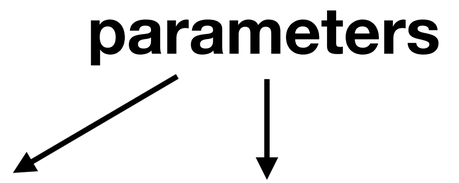
COCP

QP problem

$$
\begin{aligned}
u_t = (b_t, s_t, z_t) = \quad & \text{argmin} \quad && p_t^T b - r^T s + \tau^T z + \|Sh^+\|_2^2 + q^T h^+ \\
& \text{subject to} && h^+ = h_t + (A^{\text{in}} - A^{\text{out}})(b, s, z) \\
& && 0 \le h^+ \le h_{\max}, \quad 0 \le (b, s, z) \le u_{\max} \\
& && A^{\text{out}}(b, s, z) \le h_t, \quad s \le d_t
\end{aligned}
$$

Supply chain distribution COCP

COCP

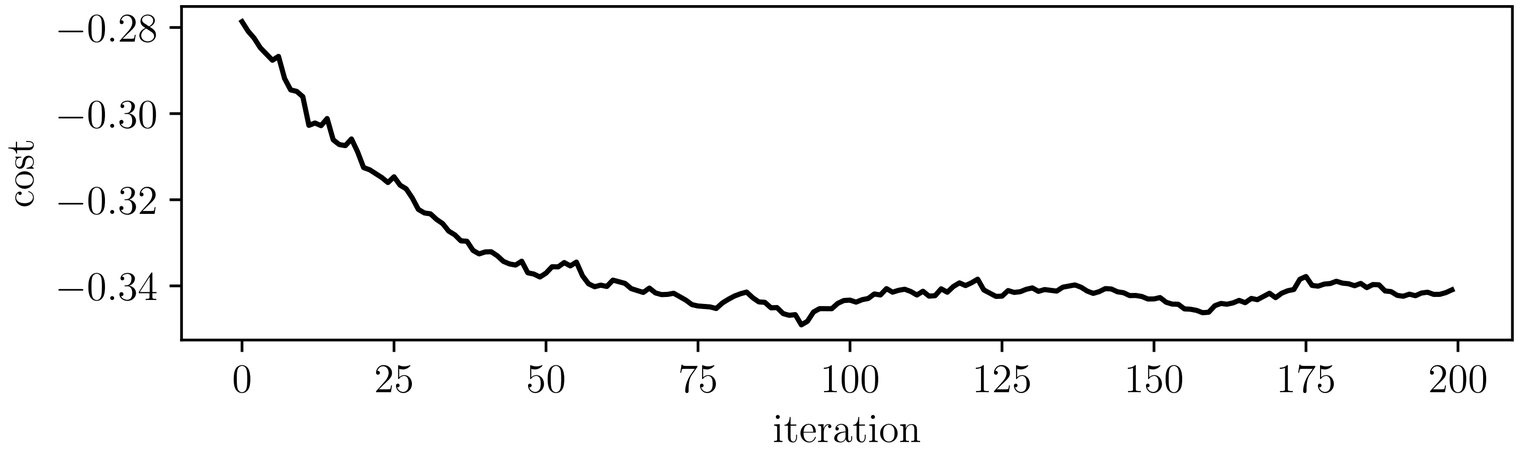


$$
\begin{aligned}
u_t = (b_t, s_t, z_t) = \quad & \text{argmin} \quad && p_t^T b - r^T s + \tau^T z + \|Sh^+\|_2^2 + q^T h^+ \\
& \text{subject to} && h^+ = h_t + (A^{\text{in}} - A^{\text{out}})(b, s, z) \\
& && 0 \le h^+ \le h_{\max}, \quad 0 \le (b, s, z) \le u_{\max} \\
& && A^{\text{out}}(b, s, z) \le h_t, \quad s \le d_t
\end{aligned}
$$

Supply chain distribution

Results

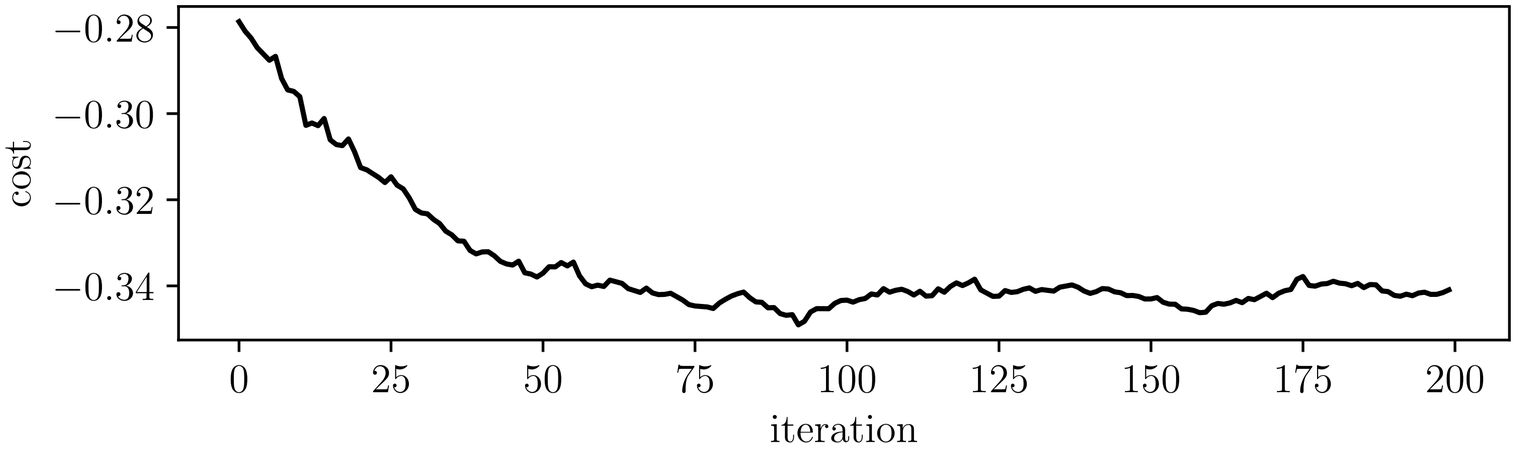
Validation loss



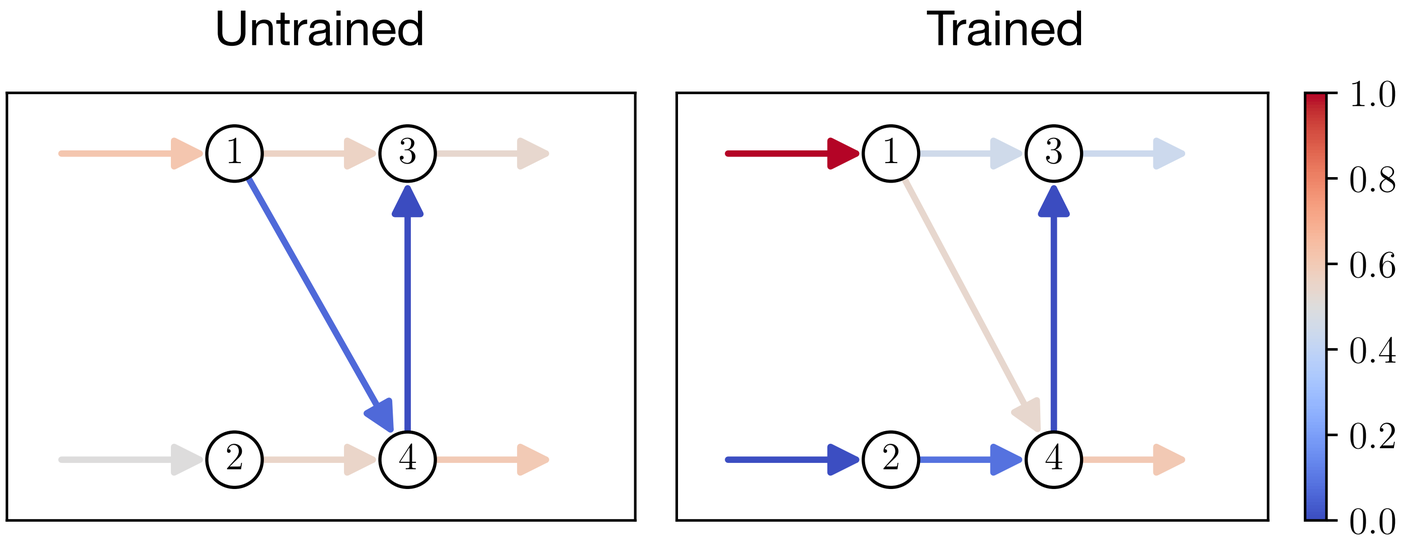
Supply chain distribution

Results

Validation loss



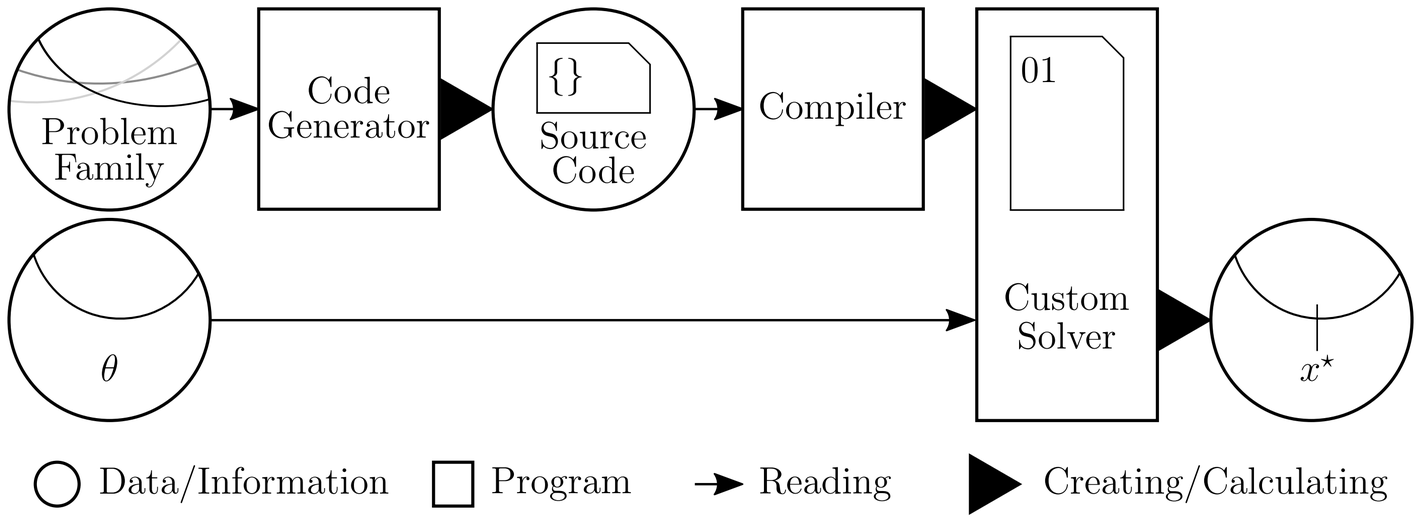
Normalized shipments



Code generation for parametric convex optimization

CVXPYgen

`https://pypi.org/project/cvxpygen/`

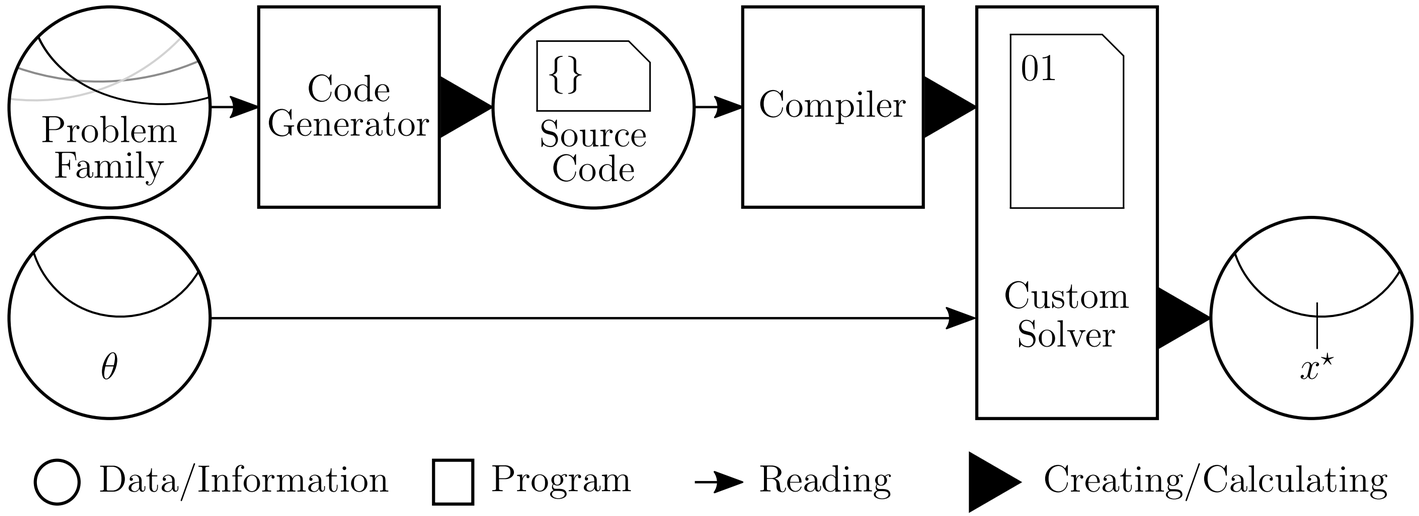


It supports quadratic (OSQP),
and conic (ECOS, SCS) optimization.

Code generation for parametric convex optimization

CVXPYgen

https://pypi.org/project/cvxpygen/



It supports quadratic (OSQP), and conic (ECOS, SCS) optimization.

Example

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

$$\theta = (G, h)$$

```
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')
p = cp.Problem(cp.Minimize(cp.sum_squares(G@x-h)),
               [x>=0])
# generate code
cpg.generate_code(p)
```

Learning COCPs summary

Interpretable

Satisfy constraints

Handle varying dynamics

Efficient and reliable (code generation)

Easy to learn from data

Learning COCPs summary

Interpretable

Satisfy constraints

Handle varying dynamics

Efficient and reliable (code generation)

Easy to learn from data

Future work

- Hybrid (mixed-integer) control policies
- Stochastic policies with safety-guarantees

Conclusions

Acknowledgements

Goran Banjac

Paul Goulart

Alberto Bemporad

Ken Goldberg

Stephen Boyd

Akshay Agrawal

Shane Barratt

Jeff Ichnowski

Max Schaller

Paras Jain

Francesco Borrelli

References

OSQP (osqp.org)

[**Accelerating Quadratic Optimization with Reinforcement Learning.** Ichnowski, Jain, *Stellato*, Banjac, Luo, Gonzalez, Stoica, Borrelli, and Goldberg. NeurIPS 2021]

[**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]

Learning COCPs

[**Embedded Code Generation with CVXPY.** Schaller, Banjac, Diamond, Agrawal, *Stellato*, and Boyd. IEEE Conf. on Decision and Control 2022 (submitted)]

[**Learning Convex Optimization Control Policies.** Agrawal, Amos, Barratt, Boyd, and Stellato. Learning for Dynamics and Control (L4DC) 2020]

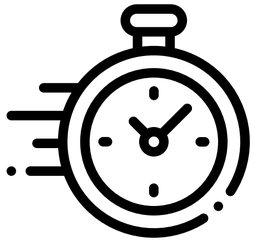
[Differentiable convex optimization layers. Agrawal, Amos, Barratt, Boyd, Diamond, and Kolter. NeurIPS 2019]

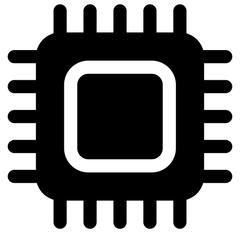
(https://pypi.org/project/cvxpygen/)

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

Conclusions

Real-time and embedded optimization





will soon **be applied everywhere**

Thanks to

Efficient and reliable optimizers

Data-driven control policies

stellato.io

bstellato@princeton.edu

@b_stellato

github.com/bstellato