Real-time Decision-Making via Data-Driven Optimization
Tremendous progress in optimization

Top500 peak CPU power

Hardware + Software

400 billion times speedups!

400,000 years → 30 seconds

Is it enough?

400,000 years  →  30 seconds
Is it enough?

Robotics

400,000 years 30 seconds

< 10 milliseconds
Is it enough?

Robotics

< 10 milliseconds

High-Frequency Trading

< 1 millisecond
Same problem with varying data

minimize \( f(x, \theta) \)
subject to \( g(x, \theta) \leq 0 \)
Same problem with varying data

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

decisions
Same problem with varying data

\[
\begin{align*}
data & \\
\downarrow & \\
\text{minimize} & & f(x, \theta) \\
\text{subject to} & & g(x, \theta) \leq 0 \\
\uparrow & \\
decisions & 
\end{align*}
\]
Same problem with varying data

Can we solve it in \textit{milliseconds} or \textit{microseconds}?
Challenges in real-time optimization

Extreme reliability
Challenges in real-time optimization

Extreme reliability

Real-Time

High performance

Limited resources
Today’s talk
Real-time Decision-Making via Data-Driven Optimization

OSQP Solver
Real-Time
Limited resources

Learning
Convex Optimization
Control Policies
Performance
Today’s talk
Real-time Decision-Making via Data-Driven Optimization
Still quadratic programming?

AN ALGORITHM FOR QUADRATIC PROGRAMMING

Marguerite Frank and Philip Wolfe
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!
First-order methods

Wide popularity

Pros

- Warm-starting
- Large-scale problems
- Embeddable
First-order methods

Wide popularity

**Pros**
- Warm-starting
- Large-scale problems
- Embeddable

**Cons**
- Low quality solutions
- Can't detect infeasibility
- Problem data dependent
First-order methods

Wide popularity

Pros
- Warm-starting
- Large-scale problems
- Embeddable

Cons
- Low quality solutions
- Can’t detect infeasibility
- Problem data dependent

OSQP
- High-quality solutions
- Detects infeasibility
- Robust
The problem

minimize \( \frac{1}{2} x^T P x + q^T x \)
subject to \( Ax \in C \)
The problem

minimize \( (1/2)x^T P x + q^T x \)
subject to \( Ax \in C \)

Quadratic program: \( C = [l, u] \)
ADMM
Alternating Direction Method of Multipliers

minimize \[ f(x) + g(x) \]  \quad \text{subject to} \quad \tilde{x} = x
ADMM
Alternating Direction Method of Multipliers

minimize \( f(x) + g(x) \) \quad \text{subject to} \quad \tilde{x} = x

Splitting

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

Splitting Iterations

\[
\begin{align*}
\tilde{x}^{k+1} &\leftarrow \text{argmin}_{\tilde{x}} \left( f(\tilde{x}) + \frac{\rho}{2} \| \tilde{x} - (x^k - y^k / \rho) \|^2 \right) \\
x^{k+1} &\leftarrow \text{argmin}_x \left( g(x) + \frac{\rho}{2} \| x - (\tilde{x}^{k+1} + y^k / \rho) \|^2 \right) \\
y^{k+1} &\leftarrow y^k + \rho \left( \tilde{x}^{k+1} - x^{k+1} \right)
\end{align*}
\]
How do we split the QP?

minimize \((1/2)x^T P x + q^T x\)

subject to \(Ax = z\)
\(z \in C\)
How do we split the QP?

minimize \((1/2)x^T Px + q^T x\)
subject to \(Ax = z\)
\(z \in C\)
How do we split the QP?

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

subject to

$Ax = z$

$z \in C$

$f$

$g$
How do we split the QP?

minimize \((1/2)x^TPx + q^Tx\)  
subject to \(Ax = z\)  
\(z \in C\)

Splitting formulation

minimize \((1/2)\tilde{x}^TP\tilde{x} + q^T\tilde{x} + \mathcal{I}_{Ax=z}(\tilde{x}, \tilde{z}) + \mathcal{I}_C(z)\)  
subject to \(\tilde{x} = x\)  
\(\tilde{z} = z\)
How do we split the QP?

\[
\text{minimize } (1/2)x^T P x + q^T x \\
\text{subject to } A x = z \\
z \in C
\]

Splitting formulation

\[
\text{minimize } (1/2)\tilde{x}^T P \tilde{x} + q^T \tilde{x} + \mathcal{I}_{A x = z}(\tilde{x}, \tilde{z}) + \mathcal{I}_{C}(z) \\
\tilde{x} = x \\
\tilde{z} = z
\]

Step sizes

\[
\sigma > 0 \\
\rho > 0
\]
ADMM iterations

\[
(x^{k+1}, \tilde{z}^{k+1}) \leftarrow \text{argmin}_{(x, z): Ax = z} \left( \frac{1}{2} x^T Px + q^T x + \sigma/2 \|x - x^k\|^2 + \rho/2 \|z - z^k + y^k / \rho\|^2 \right)
\]

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

\[
y^{k+1} \leftarrow y^k + \rho (\tilde{z}^{k+1} - \tilde{z}^{k+1})
\]
ADMM iterations

\[(x^{k+1}, z^{k+1}) \leftarrow \text{argmin}_{(x,z): Ax = z} \frac{1}{2}x^T Px + q^T x + \sigma/2 \|x - x^k\|^2 + \rho/2 \|z - z^k + y^k / \rho\|^2\]

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

\[y^{k+1} \leftarrow y^k + \rho \left( \tilde{z}^{k+1} - z^{k+1} \right)\]
ADMM iterations

$$(x^{k+1}, z^{k+1}) \leftarrow \argmin_{(x,z): Ax = z} \frac{1}{2}x^T Px + q^T x + \sigma / 2 \|x - x^k\|^2 + \rho / 2 \|z - z^k + y^k / \rho\|^2$$

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

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

Inner QP

Projection onto $C$
Solving the inner QP
Equality-constrained

minimize \((1/2)x^TPx + q^Tx + \sigma/2 \|x - x^k\|^2 + \rho/2 \|z - z^k + y^k/\rho\|^2\)
subject to \(Ax = z\)
Solving the inner QP
Equality-constrained

minimize \( (1/2)x^T P x + q^T x + \sigma / 2 \| x - x^k \|^2 + \rho / 2 \| z - z^k + y^k / \rho \|^2 \)
subject to \( Ax = z \)

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

minimize \( (1/2)x^TPx + q^T x + \sigma/2 \|x - x^k\|^2 + \rho/2 \|z - z^k + y^k/\rho\|^2 \)
subject to \( Ax = z \)

Reduced KKT system

Always solvable (robust)!

\[
\begin{bmatrix}
P + \sigma I & AT \\
A & -\frac{1}{\rho} I
\end{bmatrix}
\begin{bmatrix}
x \\
v
\end{bmatrix}
= 
\begin{bmatrix}
\sigma x^k - q \\
z^k - \frac{1}{\rho} y^k
\end{bmatrix}
\]
Solving the linear system
Direct method (small to medium scale)

\[
\begin{bmatrix}
P + \sigma I & A^T \\
A & -\frac{1}{\rho} I
\end{bmatrix}
\begin{bmatrix}
x \\
v
\end{bmatrix}
= 
\begin{bmatrix}
\sigma x^k - q \\
z^k - \frac{1}{\rho} y^k
\end{bmatrix}
\]
Solving the linear system
Direct method (small to medium scale)

Quasi-definite matrix

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

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

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)

\[(P + \sigma I + \rho A^T A) x = \sigma x^k - q + A^T (\rho z^k - y^k)\]
Solving the linear system
Indirect method (large scale)

Positive-definite matrix

\[(P + \sigma I + \rho A^T A) x = \sigma x^k - q + A^T (\rho z^k - y^k)\]
Solving the linear system
Indirect method (large scale)

Positive-definite matrix

\[(P + \sigma I + \rho A^T A)x = \sigma x^k - q + A^T (\rho z^k - y^k)\]

Conjugate gradient
Solve very large systems
Solving the linear system
Indirect method (large scale)

Positive-definite matrix

\[(P + \sigma I + \rho A^T A) x = \sigma x^k - q + A^T (\rho z^k - y^k)\]

Conjugate gradient

Solve very large systems

GPU implementation

[https://github.com/osqp/cuosqp]
Computing the projection

Quadratic program: \( C = [l, u] \)

Box projection

\( \Pi(v) = \max(\min(v, u), l) \)
Complete algorithm

Problem
minimize \((1/2)x^T P x + q^T x\)
subject to \(l \leq Ax \leq u\)
Complete algorithm

Problem

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

subject to \(l \leq Ax \leq u\)

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
minimize \((1/2)x^T P x + q^T x\)
subject to \(l \leq Ax \leq u\)

Algorithm

Linear system solve

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

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

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

\(y^{k+1} \leftarrow y^k + \rho (\tilde{z}^{k+1} - z^{k+1})\)
Complete algorithm

**Problem**

minimize \((1/2)x^TPx + q^Tx\)

subject to \(l \leq Ax \leq u\)

**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

Optimized C code

# Create OSQP object
m = osqp.OSQP()

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

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

It can be compiled into division-free
Compiled code size ~80kb (low footprint)

OSQP

GUROBI

300x Reduction!

CPLEX
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
\( r_{\text{prim}}^k = Ax^k - z^k \)

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
\( (P + \rho A^T A)x^{k+1} = -q + A^T (\rho z^k - y^k) \)
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
\[
(P + \rho A^T A) x^{k+1} = -q + A^T (\rho z^k - y^k)
\]

\[ \rho = \infty \]

\[ A^T Ax^{k+1} = A^T z^k \]

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

Dual residual
\[r_{\text{dual}}^k = Px^k + q + A^T y^k\]

Linear system in indirect method
\[(P + \rho A^T A)x^{k+1} = -q + A^T (\rho z^k - y^k)\]

- \[\rho = \infty\]
  \[A^T Ax^{k+1} = A^T z^k\]

- \[\rho = 0\]
  \[Px^{k+1} = -q - A^T y^k\]

Small primal residual
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
\[r_{\text{prim}}^k = Ax^k - z^k\]

Dual residual
\[r_{\text{dual}}^k = Px^k + q + A^T y^k\]

Linear system in indirect method
\[(P + \rho A^T A) x^{k+1} = -q + A^T (\rho z^k - y^k)\]

- \[\rho = \infty\]
  \[A^T Ax^{k+1} = A^T z^k\]

- \[\rho = 0\]
  \[Px^{k+1} = -q - A^T y^k\]

Small primal residual
Small dual residual

What’s the optimal \(\rho\)?
Extreme cases

Equality constrained QP

minimize \( (1/2) x^T P x + q^T x \)
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 \((1/2) x^T P x + q^T x\)
subject to \(Ax = b\)

Unconstrained QP
minimize \((1/2) x^T P x + q^T x\)

Solve in one ADMM step
\[
\begin{bmatrix}
P & A^T \\
A & 0
\end{bmatrix}
\begin{bmatrix}
x \\
v
\end{bmatrix}
= \begin{bmatrix}
-q \\
b
\end{bmatrix}
\]

Solve in one ADMM step
\[
\rho \approx \infty
\]
\[
P x = -q
\]
Extreme cases

Equality constrained QP
minimize $\frac{1}{2}x^T Px + q^T x$
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 $\frac{1}{2}x^T Px + q^T x$

Solve in one ADMM step
$\rho \approx 0$

$Px = -q$

We need different step sizes
Constraint-wise step size

\begin{align*}
\text{minimize} & \quad \frac{1}{2}x^T P x + q^T x \\
\text{subject to} & \quad l \leq Ax \leq u
\end{align*}

Tight constraints

\[ l_i = (Ax^*)_i \text{ or } (Ax^*)_i = u_i \]
Constraint-wise step size

minimize \( (1/2)x^T P x + q^T x \)
subject to \( l \leq Ax \leq u \)

Tight constraints
\( l_i = (Ax^*)_i \) or \( (Ax^*)_i = u_i \)

\( \rho = (\rho_1, \ldots, \rho_m) \) can be a vector
Constraint-wise step size

\[
\begin{align*}
\text{minimize} & \quad (1/2)x^TPx + q^Tx \\
\text{subject to} & \quad l \leq Ax \leq u \\
\end{align*}
\]

Tight constraints
\[
l_i = (Ax^*)_i \text{ or } (Ax^*)_i = u_i
\]

\[
\rho = (\rho_1, \ldots, \rho_m) \text{ can be a vector}
\]

Never tight
\[
l_i = -\infty \text{ and } u_i = \infty
\]
\[
\rho_i = 0
\]
Constraint-wise step size

minimize \( (1/2)x^T P x + q^T x \)
subject to \( l \leq A x \leq u \)

Tight constraints
\( l_i = (Ax^*)_i \) or \( (Ax^*)_i = u_i \)

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

Never tight
\( l_i = -\infty \) and \( u_i = \infty \)
\( \rho_i = 0 \)

Always tight
\( l_i = u_i \neq \infty \)
\( \rho_i = \infty \)
Constraint-wise step size

\[
\begin{align*}
\text{minimize} & \quad (1/2)x^T P x + q^T x \\
\text{subject to} & \quad l \leq Ax \leq u
\end{align*}
\]

Tight constraints

\[
l_i = (Ax^*)_i \text{ or } (Ax^*)_i = u_i
\]

\[\rho = (\rho_1, \ldots, \rho_m)\] can be a vector

Never tight

\[l_i = -\infty \text{ and } u_i = \infty\]

\[\rho_i = 0\]

Always tight

\[l_i \text{ or } u_i \text{ finite}\]

\[\rho_i = \infty\]

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!
Performance benchmarks

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

Maroz-Meszaros

[github.com/osqp/osqp_benchmarks]
Constraint-wise step size

minimize \( (1/2)x^T Px + q^T x \)
subject to \( l \leq Ax \leq u \)

Vector step size
\[ \rho = (\rho_1, \ldots, \rho_m) \]

Tight constraints
\[ l_i = (Ax^*)_i \text{ or } (Ax^*)_i = u_i \]

Never tight
\[ l_i = -\infty \text{ and } u_i = \infty \]
\[ \rho_i = 0 \]

Otherwise
Balance residuals
\[ \rho_i^{k+1} \leftarrow \rho_i^k \sqrt{\frac{\|r_{\text{prim}}\|}{\|r_{\text{dual}}\|}} \]

Always tight
\[ l_i = u_i \neq \infty \]
\[ \rho_i = \infty \]
Constraint-wise step size

minimize \((1/2)x^T P x + q^T x\)
subject to \(l \leq Ax \leq u\)

Vector step size
\[\rho = (\rho_1, \ldots, \rho_m)\]

Tight constraints
\[l_i = (Ax^*)_i \text{ or } (Ax^*)_i = u_i\]

Never tight
\[l_i = -\infty \text{ and } u_i = \infty\]
\[\rho_i = 0\]

Always tight
\[l_i = u_i \neq \infty\]
\[\rho_i = \infty\]

Balance residuals
\[\rho_i^{k+1} \leftarrow \rho_i^k \sqrt{\|r_{\text{prim}}\|/\|r_{\text{dual}}\|}\]

Can we learn a better update rule from data?
**Step size choice as a control problem**

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

\[
J = E \sum_{k=1}^{\infty} \gamma^k \ell(s^k)
\]
Step size choice as a control problem

\[ s^k = (x^k, z^k, y^k) \]

controller

\[ \rho = \phi(s) \]

algorithm

\[ s^{k+1} = \text{ADMM}_\rho(s^k) \]

Stage cost

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

Cumulative cost

\[ J = \mathbb{E} \sum_{k=1}^{\infty} \gamma^k \ell(s^k) \]

Train with

Deep Policy Gradient methods (TD3)
Constraint-wise control policy

Per-constraint update rule

\[ \rho_i = \phi_c(s_i) \]

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

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

\[ s_i = \begin{bmatrix} \min(z_i - l_i, u_i - z_i) \\ (Ax)_i - z_i \\ y_i \end{bmatrix} \text{ slacks} \]
Constraint-wise control policy

Per-constraint update rule

\[ \rho_i = \phi_c(s_i) \]

Per-constraint state

\[ s_i = \begin{bmatrix} \min(z_i - l, u_i - z_i) \\ (Ax)_i - z_i \\ y_i \end{bmatrix} \]

slacks
infeasibility
Constraint-wise control policy

Per-constraint update rule
\[ \rho_i = \phi_c(s_i) \]

Per-constraint state
\[ s_i = \begin{bmatrix} \min(z_i - l_i, u_i - z_i) \\ (Ax)_i - z_i \\ y_i \end{bmatrix} \]

\[ \phi(s) = \begin{bmatrix} \phi_c(s_1) \\ \phi_c(s_2) \\ \vdots \\ \phi_c(s_m) \end{bmatrix} \]
**Constraint-wise control policy**

Per-constraint update rule
\[
\rho_i = \phi_c(s_i)
\]

Per-constraint state
\[
s_i = \begin{bmatrix}
\min(z_i - l_i, u_i - z_i) \\
(Ax)_i - z_i \\
y_i
\end{bmatrix}
\]

Generalize to different dimensions

Low-dimensional state per constraint

Small NN policy
\[
\phi_c(s_i)
\]
Visualize learned policy

Interpretable policy

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

$z_i = Ax_i^5$
Performance with step size learning

Timings for high-accuracy convergence criteria

Timings for high-accuracy convergence criteria
Performance with step size learning

Timings for high-accuracy convergence criteria

- Gurobi
- OSQP
- RLQP (scalar)
- RLQP (vector)

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

- 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

- Code generation from C to C
- CVXPY integration
OSQP 1.0 (this summer!)

**Improved embedded code generation**

- Code generation from C to C
- CVXPY integration

**Differentiable layers**

[PyTorch]
OSQP 1.0 (this summer!)

Improved embedded code generation

• Code generation from C to C
• CVXPY integration

Differentiable layers

Modular linear algebra

- NVIDIA
- Intel
- Math Kernel Library (MKL)
- FPGAs
- CUDA

---
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

\[ x_t \quad \text{state} \]
\[ u_t \quad \text{input} \]
\[ w_t \quad \text{(random) disturbance} \]

\[ \phi(x_t) \quad \text{control policy} \]
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: \( x^TQx + u^TRu \)
**Explicit vs implicit control policies**

**Explicit**
Complete control specification

**Implicit (optimization-based)**
Designer specifies goal and requirements
Optimizer computes the action

**Example:** PI Controller
\[
  u_t = -K_P e_t - K_I \sum_{\tau=0}^{t} e_{\tau}
\]

**Example:** LQR Controller
Dynamics: \( x_{k+1} = Ax_t + Bu_t + w_t \)
Stage cost: \( x^T Q x + u^T R u \)

\[
  u_t = \arg\min_u u^T R u + (Ax_t + Bu)^T P (Ax_t + Bu)
  = K x_t
\]
Convex optimization control policies (COCPs)

\[ u_t = \underset{u}{\text{argmin}} \quad f(x_t, u, \theta) \]
subject to \[ g(x_t, u, \theta) \leq 0 \]
\[ A(x_t, \theta)u = b(x_t, \theta) \]

- \( x_t \) state
- \( \theta \) parameters to tune
- \( f, g \) convex functions
Convex optimization control policies (COCPs)

\[ u_t = \text{argmin}_u f(x_t, u, \theta) \]

subject to
\[ g(x_t, u, \theta) \leq 0 \]
\[ A(x_t, \theta)u = b(x_t, \theta) \]

\( x_t \) state
\( \theta \) parameters to tune
\( f, g \) convex functions
Convex optimization control policies (COCPs)

\[ u_t = \arg \min_u f(x_t, u, \theta) \]

subject to

\[ g(x_t, u, \theta) \leq 0 \]
\[ A(x_t, \theta)u = b(x_t, \theta) \]

\( x_t \) state
\( \theta \) 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 \textit{stochastic process}

\textbf{Trajectories}

\begin{align*}
X &= (x_0, \ldots, x_{T-1}, x_T) \\
U &= (u_0, \ldots, u_{T-1}) \\
W &= (w_0, \ldots, w_{T-1})
\end{align*}
Judging COCPs

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

<table>
<thead>
<tr>
<th>Trajectories</th>
<th>Policy cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X = (x_0, \ldots, x_{T-1}, x_T)$</td>
<td>$J(\theta) = \mathbb{E} \psi(X, U, W)$</td>
</tr>
<tr>
<td>$U = (u_0, \ldots, u_{T-1})$</td>
<td></td>
</tr>
<tr>
<td>$W = (w_0, \ldots, w_{T-1})$</td>
<td></td>
</tr>
</tbody>
</table>
Judging COCPs

Given a policy, state and input trajectories form a \textit{stochastic process}.

\begin{align*}
\text{Trajectories} & \\
X &= (x_0, \ldots, x_{T-1}, x_T) \\
U &= (u_0, \ldots, u_{T-1}) \\
W &= (w_0, \ldots, w_{T-1}) \\
\text{Policy cost} & \\
J(\theta) &= \mathbf{E} \psi(X, U, W)
\end{align*}

\text{Approximate } J(\theta) \text{ from data (monte carlo simulation)}

\[ \hat{J}(\theta) = \frac{1}{K} \sum_{i=1}^{K} \psi(X^i, U^i, W^i) \]
COCO 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 \( T \to \infty \)

\[ \phi(x_t) = \arg\min_{u} \mathbb{E}(g(x_t, u, w_t) + V(f(x_t, u, w_t))) \]
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 \( T \to \infty \)
\[ \phi(x_t) = \arg\min_u \mathbb{E} \left( g(x_t, u, w_t) + V(f(x_t, u, w_t)) \right) \]

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 $T \to \infty$

$$\phi(x_t) = \arg\min_u E\left( g(x_t, u, w_t) + V\left( f(x_t, u, w_t) \right) \right)$$

Value function

COCP if

- $f$ affine in $x$ and $u$
- $g$ convex in $x$ and $u$
- $V$ is convex
COCP Example: approximate dynamic programming

\[ \phi(x_t) = \arg\min_u E \left( g(x_t, u, w_t) + \hat{V}(f(x_t, u, w_t)) \right) \]
COCP Example: approximate dynamic programming

$$\phi(x_t) = \arg\min_u E \left( g(x_t, u, w_t) + \hat{V}(f(x_t, u, w_t)) \right)$$

Approximate value function
COCP Example: approximate dynamic programming

\[
\phi(x_t) = \operatorname{argmin}_u \mathbb{E} \left( g(x_t, u, w_t) + \hat{V}(f(x_t, u, w_t)) \right)
\]

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

\[ \theta^{k+1} = \theta^k - \epsilon^k \nabla_{\theta} \hat{J}(\theta^k) \]

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 $\phi_\theta$?
Differentiating through convex optimization problems

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

- \( x \) variable
- \( \theta \) parameter
Differentiating through convex optimization problems

\[ \begin{align*}
\text{minimize} & \quad f(x, \theta) \\
\text{subject to} & \quad g(x, \theta) \leq 0
\end{align*} \]

- \( x \) variable
- \( \theta \) parameter

**Optimality conditions**

\[ \nabla_x f(x, \theta) + D_x g(x, \theta)^T y = 0 \]
\[ \text{diag}(y) g(x, \theta) = 0 \]
\[ g(x^*, \theta) \leq 0 \]
\[ y^* \geq 0 \]
Differentiating through convex optimization problems

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

\[ \nabla_x f(x, \theta) + D_x g(x, \theta)^T y = 0 \]
\[ \text{diag}(y)g(x, \theta) = 0 \]
\[ g(x^*, \theta) \leq 0 \]
\[ y^* \geq 0 \]

- \( x \) variable
- \( \theta \) parameter

stationarity
**Differentiating through convex optimization problems**

minimize \( f(x, \theta) \)

subject to \( g(x, \theta) \leq 0 \)

- \( x \) variable
- \( \theta \) parameter

**Optimality conditions**

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

\[ \text{diag}(y)g(x, \theta) = 0 \]

\[ g(x^*, \theta) \leq 0 \]

\[ y^* \geq 0 \]
Differentiating through convex optimization problems

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

\[
\nabla_x f(x, \theta) + D_x g(x, \theta)^T y = 0
\]
\[
\text{diag}(y) g(x, \theta) = 0
\]
\[
g(x^*, \theta) \leq 0
\]
\[
y^* \geq 0
\]

• \( x \) variable
• \( \theta \) parameter

Optimality conditions

\( F(z^*(\theta), \theta) = 0 \)

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

stationarity
complementary slackness
Differentiating through convex optimization problems

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

Optimality conditions

\[ \nabla_x f(x, \theta) + D_x g(x, \theta)^T y = 0 \]
\[ \text{diag}(y) g(x, \theta) = 0 \]
\[ g(x^*, \theta) \leq 0 \]
\[ y^* \geq 0 \]

\[ F(z^*(\theta), \theta) = 0 \]

Goal
Compute \( Dz^*(\theta) \)
Differentiating through convex optimization problems

\[ F(z^*(\theta), \theta) = 0 \]

primal/dual solution

\[ z^* = (x^*(\theta), y^*(\theta)) \]
Differentiating through convex optimization problems

\[ F(z^*(\theta), \theta) = 0 \]

**Primal/Dual Solution**

\[ z^* = (x^*(\theta), y^*(\theta)) \]

**Implicit Function Theorem**

\[ D_z F(z^*, \theta) Dz^*(\theta) + D_\theta F(z^*, \theta) = 0 \]
Differentiating through convex optimization problems

\[ F(z^*(\theta), \theta) = 0 \]  

primal/dual solution

\[ z^* = (x^*(\theta), y^*(\theta)) \]

Implicit function theorem

\[ D_z F(z^*, \theta) Dz^*(\theta) + D_\theta F(z^*, \theta) = 0 \]

\[ Dz^*(\theta) = - (D_z F(z^*, \theta))^{-1} D_\theta F(z^*, \theta) \quad (D_z F(z^*, \theta) \text{ must be invertible}) \]  

one linear system solution
Differentiating through convex optimization problems

\[ F(z^*(\theta), \theta) = 0 \]

primal/dual solution
\[ z^* = (x^*(\theta), y^*(\theta)) \]

Implicit function theorem
\[
D_z F(z^*, \theta) Dz^*(\theta) + D_\theta F(z^*, \theta) = 0
\]

\[
Dz^*(\theta) = -\left( D_z F(z^*, \theta) \right)^{-1} D_\theta F(z^*, \theta)
\]

must be invertible
One linear system solution

We plug \( Dz^*(\theta) \) in AD
(automatic differentiation)
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: \( x_t^T Q x_t + u_t^T Ru_t \)

COCP Policy (QP)

\[
 u_t = \arg\min_u u^T Ru + \|\theta(A x_t + B u)\|_2^2
\]

subject to \( \|u\|_\infty \leq 1 \)
Box-constrained LQR

Problem setup

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

COCP Policy (QP)

\[
    u_t = \arg\min_{u} \quad u^T Ru + \|\theta(Ax_t + Bu)\|_2^2 \\
    \text{subject to} \quad \|u\|_\infty \leq 1
\]
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

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

Input: $u_t = (b_t, s_t, z_t)$
Supply chain distribution

<table>
<thead>
<tr>
<th>quantity held</th>
<th>supplier price</th>
<th>consumer demand</th>
</tr>
</thead>
</table>

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

Input: \( u_t = (b_t, s_t, z_t) \)

Dynamics

\[
h_{t+1} = h_t + \left( A_{\text{in}} - A_{\text{out}} \right) u_t
\]

\( p_{t+1} \) 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

quantity held  supplier price  consumer demand

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

Input: \( u_t = (b_t, s_t, z_t) \)

\begin{align*}
\text{Dynamics} \\
& h_{t+1} = h_t + (A^{\text{in}} - A^{\text{out}}) u_t \\
& p_{t+1} \text{ and } d_{t+1} \text{ are log-normal}
\end{align*}

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

Network example
Supply chain distribution
Cost and constraints

Stage cost

\[ p_t^T b_t - r^T s_t + \tau^T z_t + \alpha^T h_t + \beta^T h_t^2 + I(x_t, u_t) \]
Supply chain distribution
Cost and constraints

\[ p_t^T b_t - r_t^T s_t + \tau_t^T z_t + \alpha_t^T h_t + \beta t^2 + I(x_t, u_t) \]

sale revenues

suppliers payment

Stage cost

shipment cost
Supply chain distribution
Cost and constraints

Stage cost

\[ p_t^T b_t - r_t^T s_t + \tau_t^T z_t + \alpha_t^T h_t + \beta_t^T h_t^2 + \mathcal{I}(x_t, u_t) \]

sale revenues
suppliers payment
shipment cost
storage cost
Supply chain distribution
Cost and constraints

Stage cost

\[ p_t^T b_t - r_t^T s_t + \tau_t^T z_t + \alpha_t^T h_t + \beta_t^T h_t^2 + \mathcal{I}(x_t, u_t) \]

sale revenues
suppliers payment
shipment cost
storage cost

constraints
Supply chain distribution
Cost and constraints

Stage cost

\[ p_t^T b_t - r_t^T s_t + r_t^T z_t + \alpha_t^T h_t + \beta_t^T h_t^2 + \mathcal{I}(x_t, u_t) \]

sale revenues
suppliers payment
shipment cost
storage cost

Constraints

\[ 0 \leq h_t \leq h_{\text{max}}, \quad 0 \leq u_t \leq u_{\text{max}} \]

\[ A^{\text{out}} u_t \leq h_t, \quad s \leq d_t \]
Supply chain distribution COCP

COCP

QP problem

\[ u_t = (b_t, s_t, z_t) = \text{argmin} \quad p^T b - r^T s + \tau^T z + \|Sh^+\|_2^2 + q^T h^+ \]

subject to

\[ h^+ = h_t + (A^{in} - A^{out})(b, s, z) \]
\[ 0 \leq h^+ \leq h_{\text{max}}, \quad 0 \leq (b, s, z) \leq u_{\text{max}} \]
\[ A^{out}(b, s, z) \leq h_t, \quad s \leq d_t \]
Supply chain distribution COCP

COCP

\[ u_t = (b_t, s_t, z_t) = \text{argmin} \]

subject to \[
p_t^T b - r^T s + \tau^T z + \| S h^+ \|_2^2 + q^T h^+
\]

\[ h^+ = h_t + (A^\text{in} - A^\text{out})(b, s, z) \]

\[ 0 \leq h^+ \leq h_{\text{max}}, \quad 0 \leq (b, s, z) \leq u_{\text{max}} \]

\[ A^\text{out}(b, s, z) \leq h_t, \quad s \leq d_t \]
Supply chain distribution

Results

Validation loss
Supply chain distribution

Results

Validation loss

Normalized shipments
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

\[
\begin{align*}
\text{minimize} & \quad \|Gx - h\|^2 \\
\text{subject to} & \quad x \geq 0 \\
\theta = (G, h)
\end{align*}
\]

```python
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](https://github.com/cvxgrp/cocp))


Mathematical Programming Computation 2020

[Infeasibility detection in the alternating direction method of multipliers for convex optimization.](https://pypi.org/project/cvxpygen/) Banjac, Goulart, Stellato, and Boyd.

Journal of Optimization Theory and Applications 2019

[Embedded code generation using the OSQP solver.](https://github.com/cvxgrp/cocp) Stellato, Banjac, Stellato, Moehle, Goulart, Bemporad, and Boyd.

IEEE Conf. on Decision and Control 2017

**Learning COCPs**

[Embedded Code Generation with CVXPY.](https://pypi.org/project/cvxpygen/) Schaller, Banjac, Diamond, Agrawal, Stellato, and Boyd.

IEEE Conf. on Decision and Control 2022 (submitted)]


Learning for Dynamics and Control (L4DC) 2020


NeurIPS 2019


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