

Real-Time Embedded Convex Optimization

Stephen Boyd

joint work with Michael Grant, Jacob Mattingley, Yang Wang
Electrical Engineering Department, Stanford University

Outline

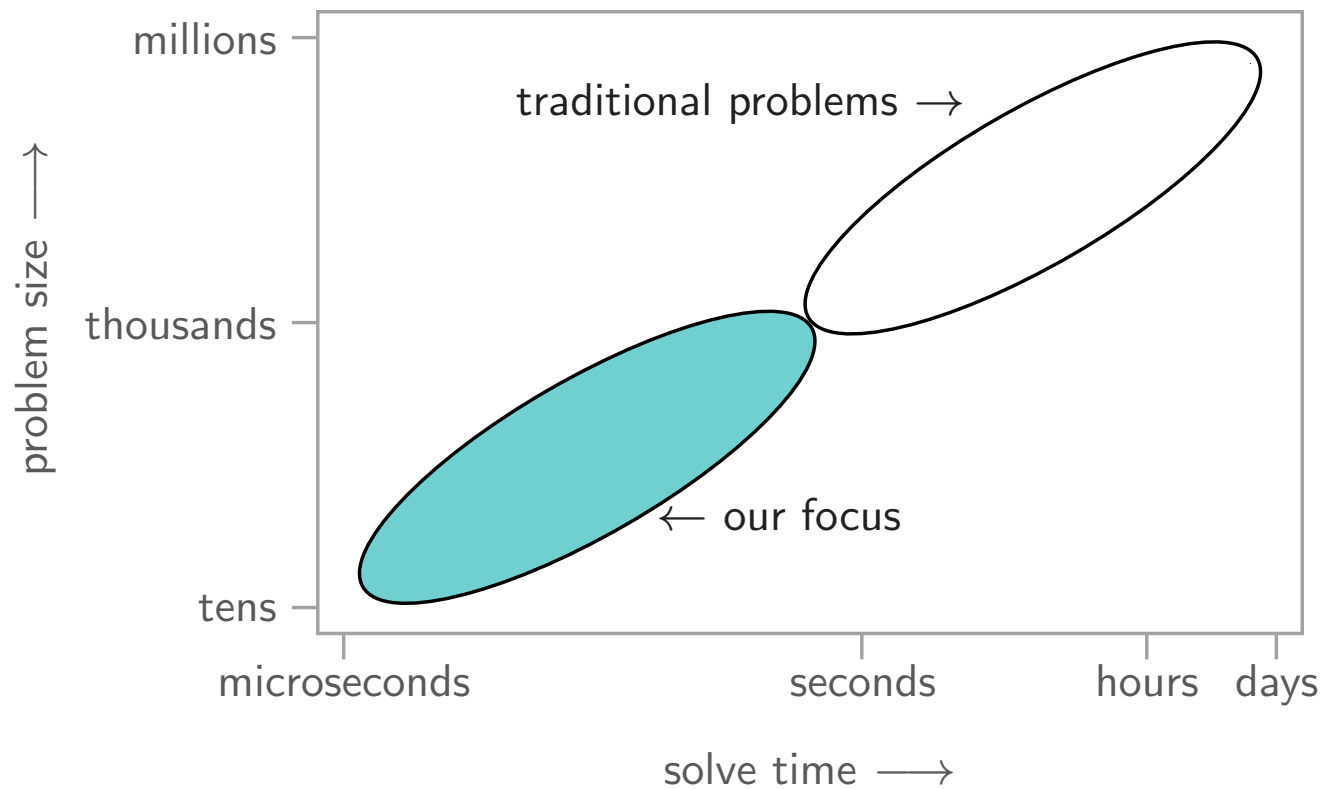
- Real-time embedded convex optimization
- Examples
- Parser/solvers for convex optimization
- Code generation for real-time embedded convex optimization

Embedded optimization

- embed solvers in real-time applications
- *i.e.*, **solve an optimization problem at each time step**
- used now for applications with hour/minute time-scales
 - process control
 - supply chain and revenue management
 - trading

What's new

embedded optimization at **millisecond/microsecond time-scales**



Applications

- real-time resource allocation
 - update allocation as objective, resource availabilities change
- signal processing
 - estimate signal by solving optimization problem over sliding window
 - replace least-squares estimates with robust (Huber, ℓ_1) versions
 - re-design (adapt) coefficients as signal/system model changes
- control
 - closed-loop control via rolling horizon optimization
 - real-time trajectory planning
- all of these done now, on long (minutes or more) time scales
but could be done on millisecond/microsecond time scales

Outline

- Real-time embedded convex optimization
- **Examples**
- Parser/solvers for convex optimization
- Code generation for real-time embedded convex optimization

Multi-period processor speed scheduling

- processor adjusts its speed $s_t \in [s^{\min}, s^{\max}]$ in each of T time periods
- energy consumed in period t is $\phi(s_t)$; total energy is $E = \sum_{t=1}^T \phi(s_t)$
- n jobs
 - job i available at $t = A_i$; must finish by deadline $t = D_i$
 - job i requires total work $W_i \geq 0$
- $S_{ti} \geq 0$ is effective processor speed allocated to job i in period t

$$s_t = \sum_{i=1}^n S_{ti}, \quad \sum_{t=A_i}^{D_i} S_{ti} \geq W_i$$

Minimum energy processor speed scheduling

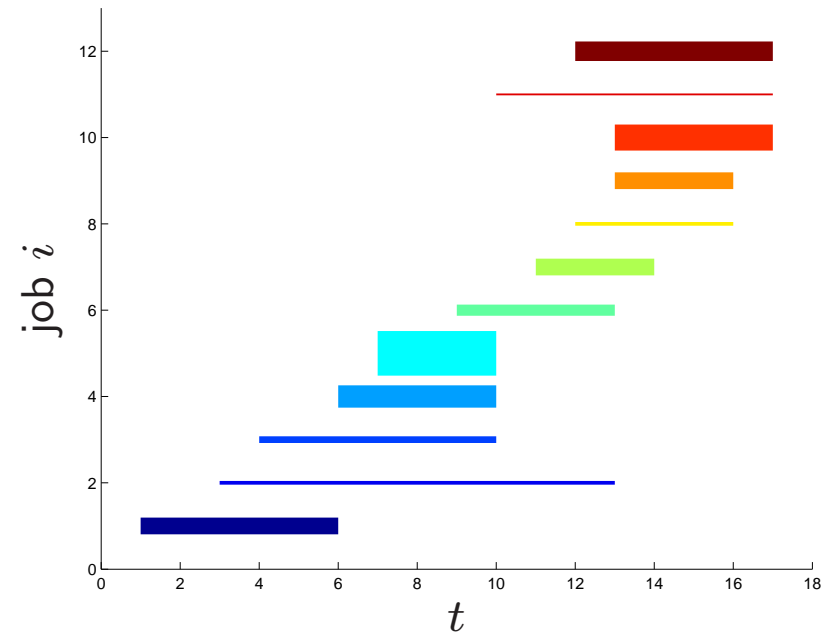
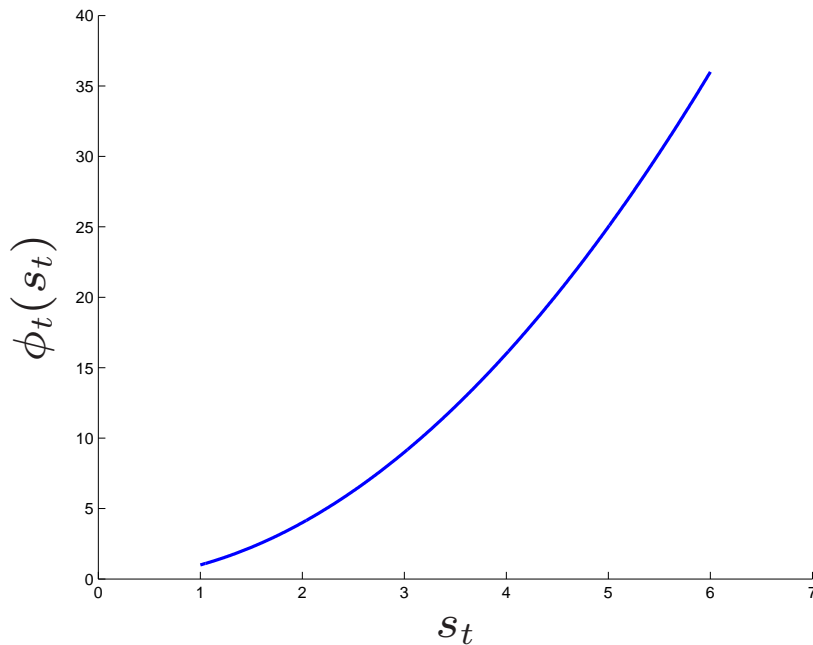
- choose speeds s_t and allocations S_{ti} to minimize total energy E

$$\begin{aligned} \text{minimize} \quad & E = \sum_{t=1}^T \phi(s_t) \\ \text{subject to} \quad & s^{\min} \leq s_t \leq s^{\max}, \quad t = 1, \dots, T \\ & s_t = \sum_{i=1}^n S_{ti}, \quad t = 1, \dots, T \\ & \sum_{t=A_i}^{D_i} S_{ti} \geq W_i, \quad i = 1, \dots, n \end{aligned}$$

- a convex problem when ϕ is convex
- more sophisticated versions include
 - multiple processors
 - other constraints (thermal, speed slew rate, . . .)
 - stochastic models for (future) data

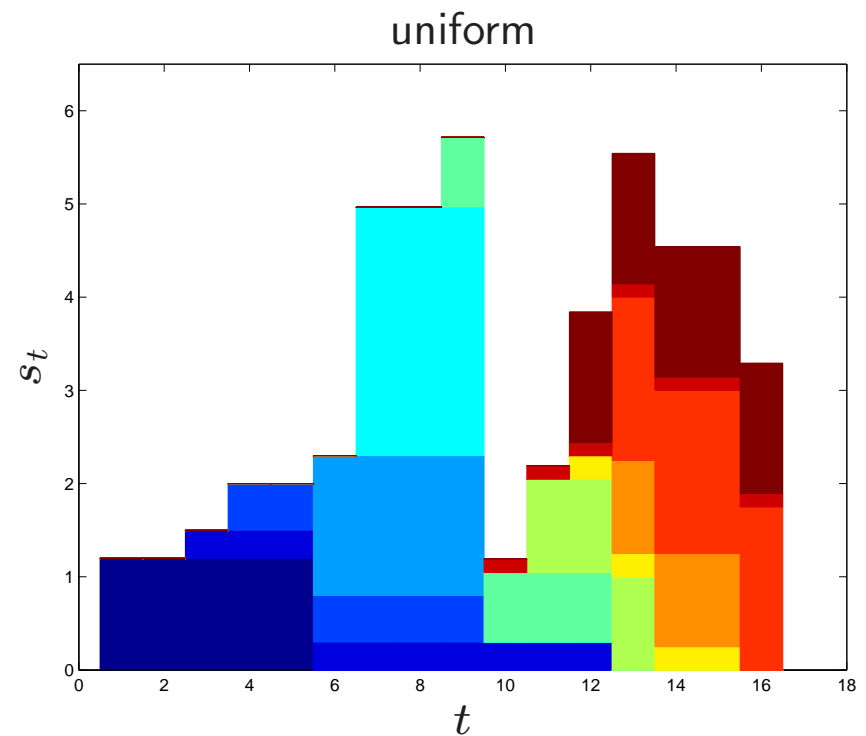
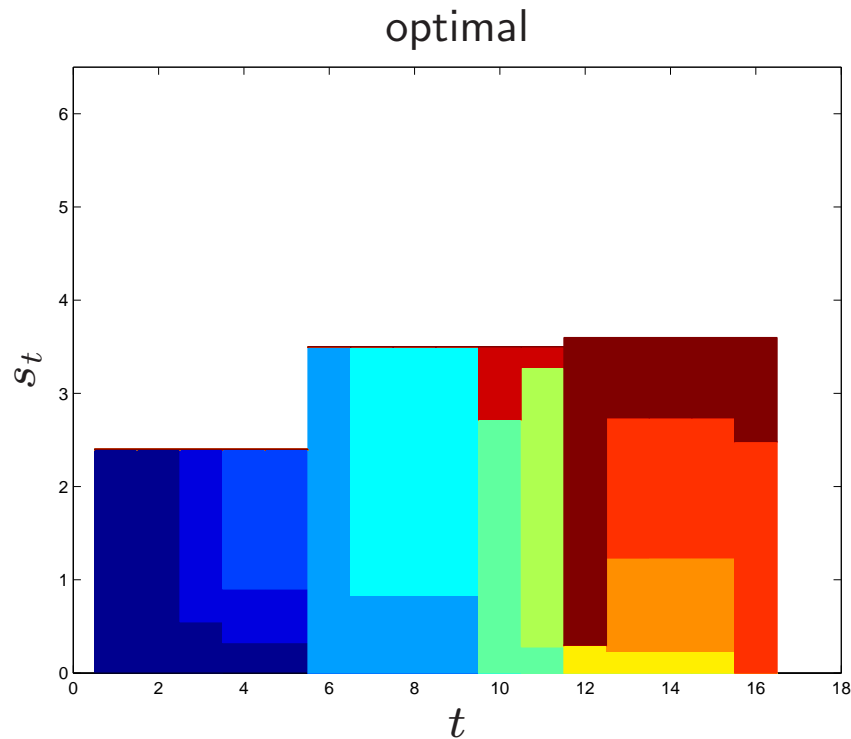
Example

- $T = 16$ periods, $n = 12$ jobs
- $s^{\min} = 1$, $s^{\max} = 6$, $\phi(s_t) = s_t^2$
- jobs shown as bars over $[A_i, D_i]$ with area $\propto W_i$

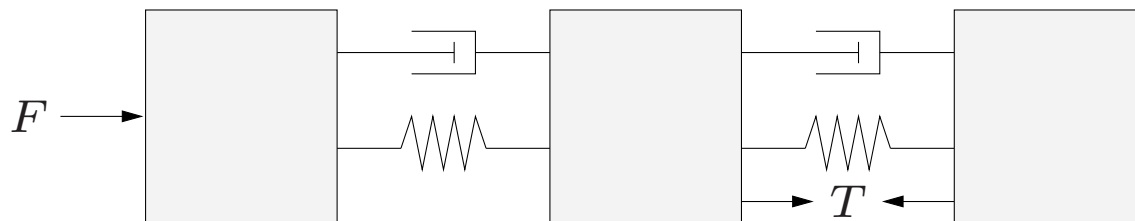


Optimal and uniform schedules

- uniform schedule: $S_{ti} = W_i / (D_i - A_i)$; gives $E^{\text{unif}} = 374.1$
- optimal schedule S_{ti}^* gives $E^* = 167.1$



Minimum time control with active vibration suppression

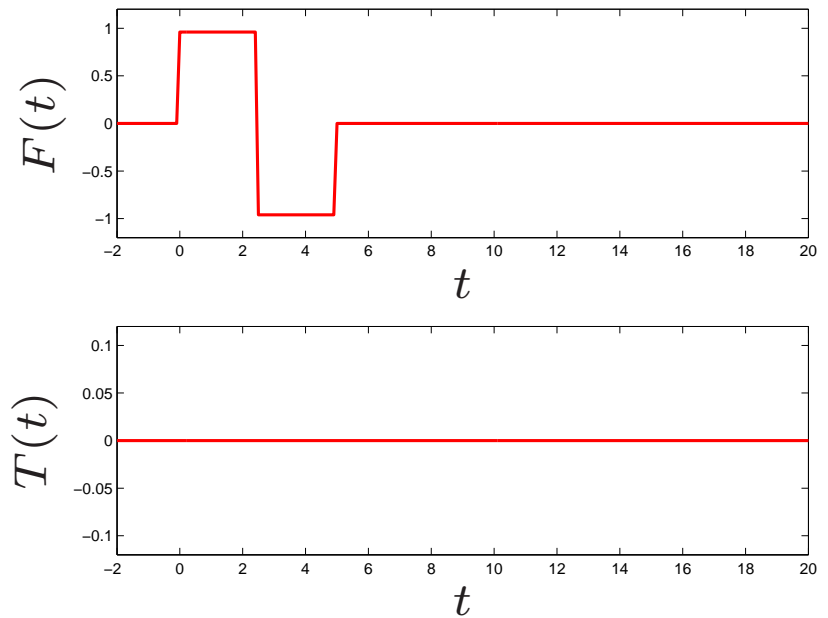
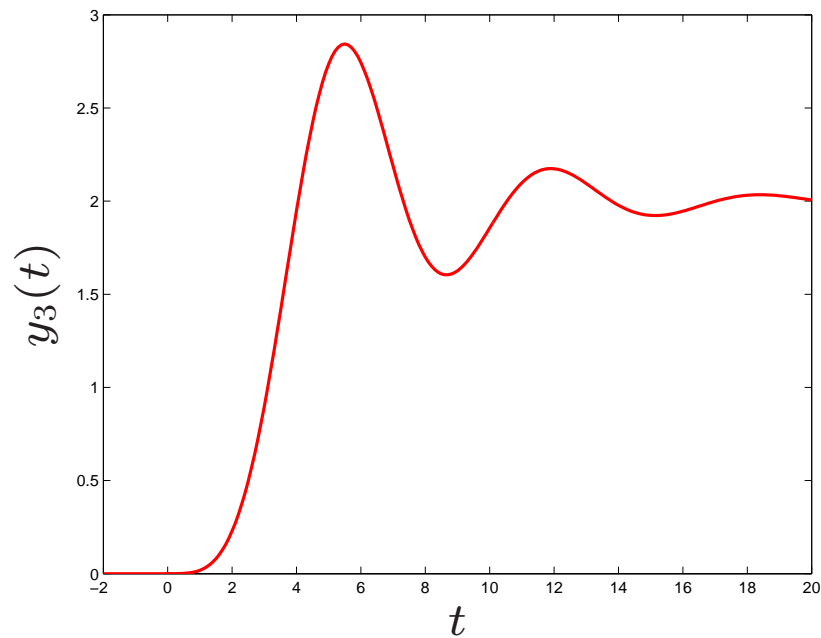


- force $F(t)$ moves object modeled as 3 masses (2 vibration modes)
- tension $T(t)$ used for active vibration suppression
- goal: move object to commanded position as quickly as possible, with

$$|F(t)| \leq 1, \quad |T(t)| \leq 0.1$$

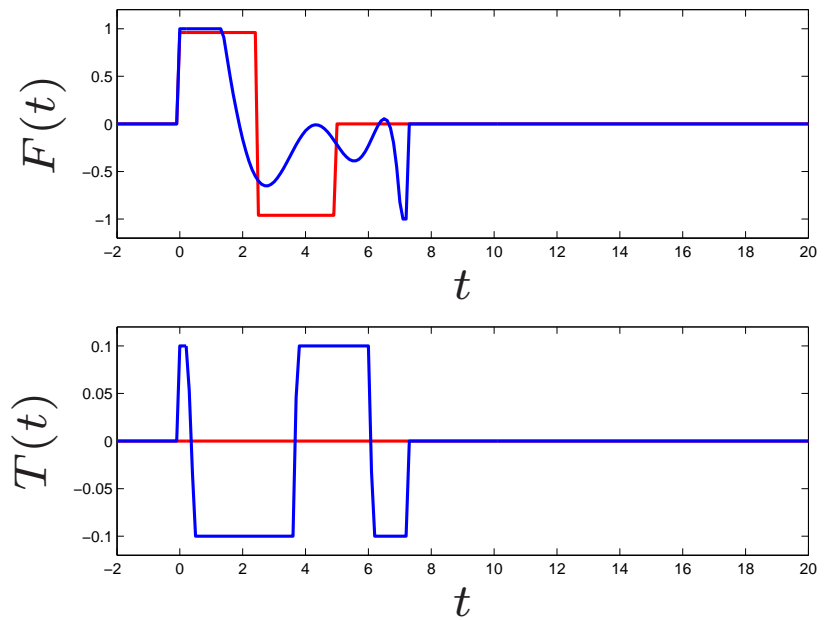
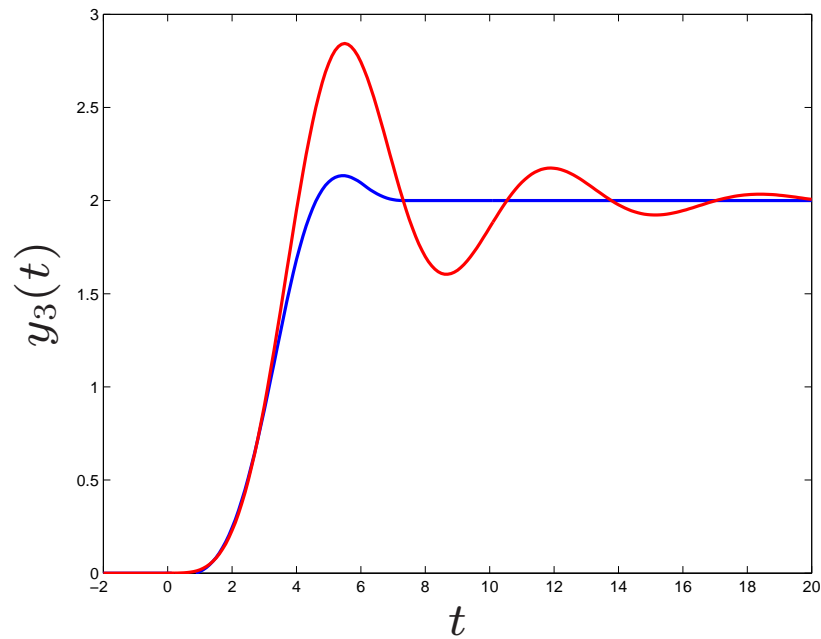
Ignoring vibration modes

- treat object as single mass; apply only F
- analytical ('bang-bang') solution



With vibration modes

- no analytical solution, but reduces to a quasiconvex problem
- can be solved by solving a small number of convex problems



Grasp force optimization

- choose K grasping forces on object to
 - resist external wrench (force and torque)
 - respect friction cone constraints
 - minimize maximum grasp force
- convex problem (second-order cone program or SOCP):

$$\text{minimize } \max_i \|f^{(i)}\|_2$$

max contact force

$$\text{subject to } \sum_i Q^{(i)} f^{(i)} = f^{\text{ext}}$$

force equilibrium

$$\sum_i p^{(i)} \times (Q^{(i)} f^{(i)}) = \tau^{\text{ext}}$$

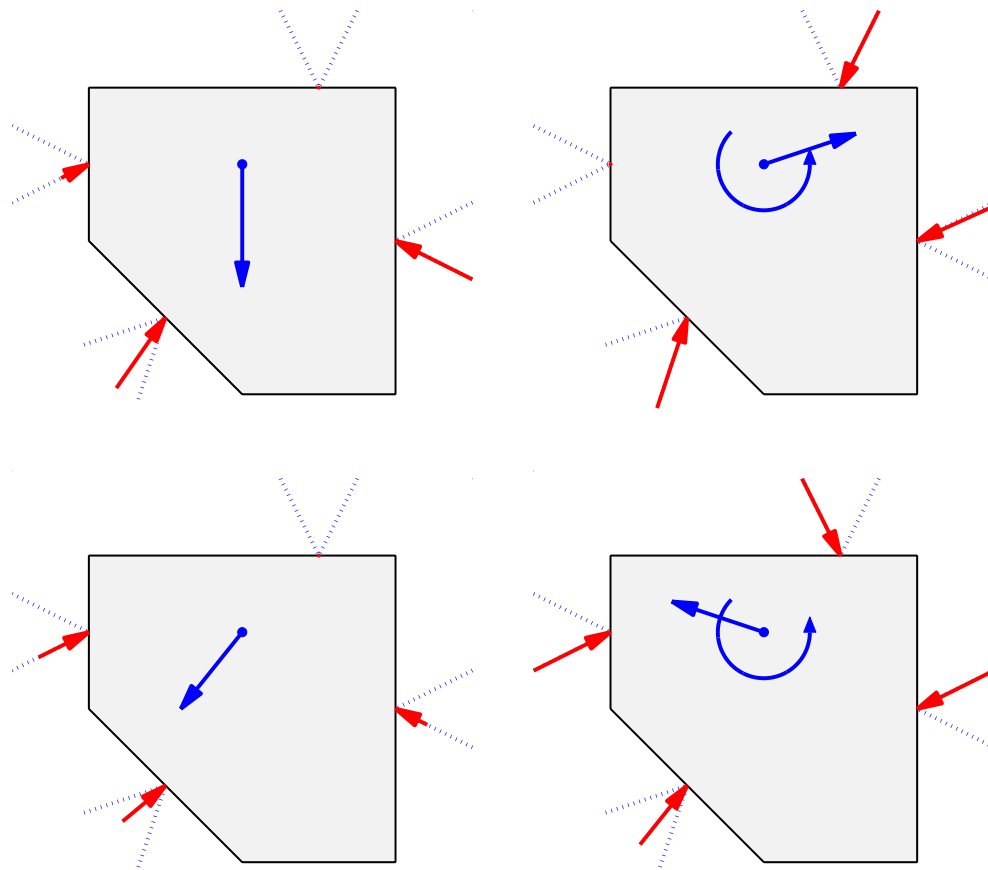
torque equilibrium

$$\mu_i f_z^{(i)} \geq \left(f_x^{(i)2} + f_y^{(i)2} \right)^{1/2}$$

friction cone constraints

variables $f^{(i)} \in \mathbf{R}^3$, $i = 1, \dots, K$ (contact forces)

Example



Grasp force optimization solve times

- example with $K = 5$ fingers (grasp points)
- reduces to SOCP with 15 vars, 6 eqs, 5 3-dim SOCs
- custom code solve time: $50\mu s$ (SDPT3: 100ms)

Robust Kalman filtering

- estimate state of a linear dynamical system driven by IID noise
- sensor measurements have occasional outliers (failures, jamming, . . .)
- model: $x_{t+1} = Ax_t + w_t, \quad y_t = Cx_t + v_t + z_t$
 - $w_t \sim \mathcal{N}(0, W), \quad v_t \sim \mathcal{N}(0, V)$
 - z_t is **sparse**; represents outliers, failures, . . .
- (steady-state) Kalman filter (for case $z_t = 0$):
 - time update: $\hat{x}_{t+1|t} = A\hat{x}_{t|t}$
 - measurement update: $\hat{x}_{t|t} = \hat{x}_{t|t-1} + L(y_t - C\hat{x}_{t|t-1})$
- we'll replace measurement update with robust version to handle outliers

Measurement update via optimization

- standard KF: $\hat{x}_{t|t}$ is solution of quadratic problem

$$\begin{aligned} \text{minimize} \quad & v^T V^{-1} v + (x - \hat{x}_{t|t-1})^T \Sigma^{-1} (x - \hat{x}_{t|t-1}) \\ \text{subject to} \quad & y_t = Cx + v \end{aligned}$$

with variables x, v (simple analytic solution)

- robust KF: choose $\hat{x}_{t|t}$ as solution of convex problem

$$\begin{aligned} \text{minimize} \quad & v^T V^{-1} v + (x - \hat{x}_{t|t-1})^T \Sigma^{-1} (x - \hat{x}_{t|t-1}) + \lambda \|z\|_1 \\ \text{subject to} \quad & y_t = Cx + v + z \end{aligned}$$

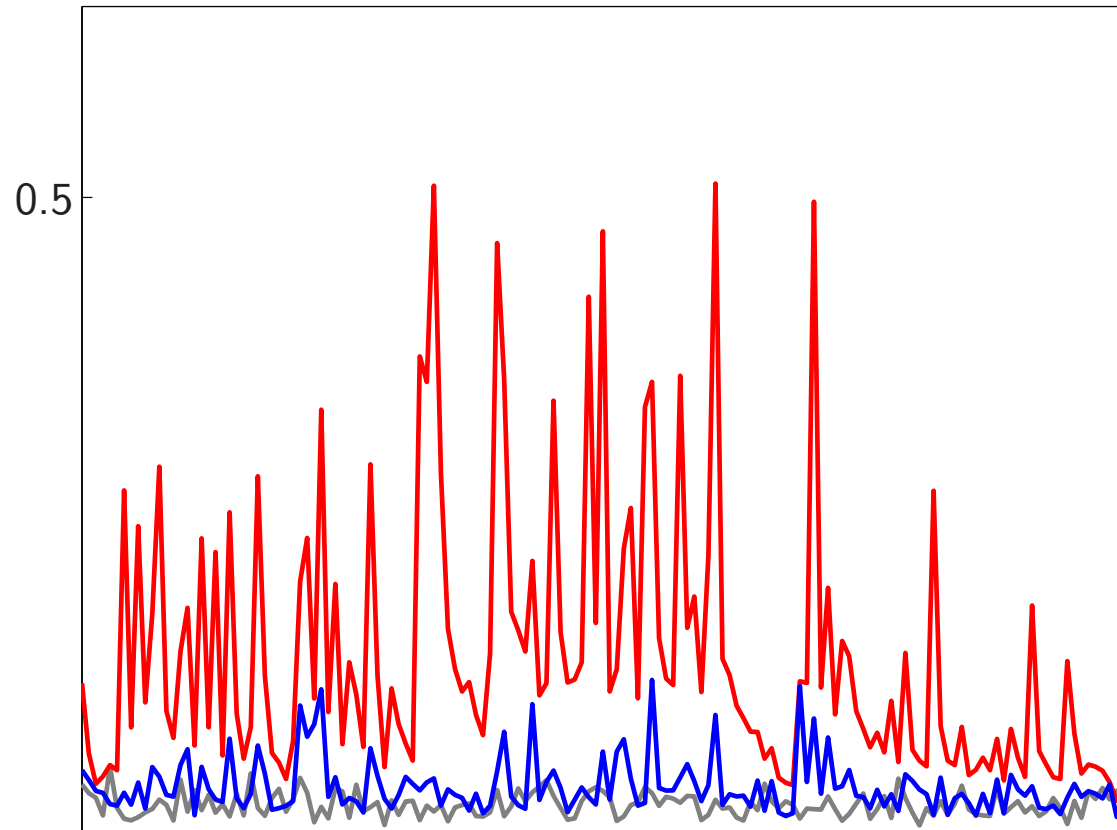
with variables x, v, z (requires solving a QP)

Example

- 50 states, 15 measurements
- with prob. 5%, measurement components replaced with $(y_t)_i = (v_t)_i$
- so, get a flawed measurement (*i.e.*, $z_t \neq 0$) every other step (or so)

State estimation error

$\|x - \hat{x}_{t|t}\|_2$ for KF (red); robust KF (blue); KF with $z = 0$ (gray)

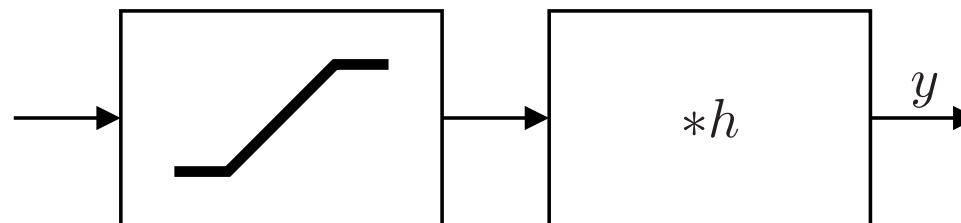


Robust Kalman filter solve time

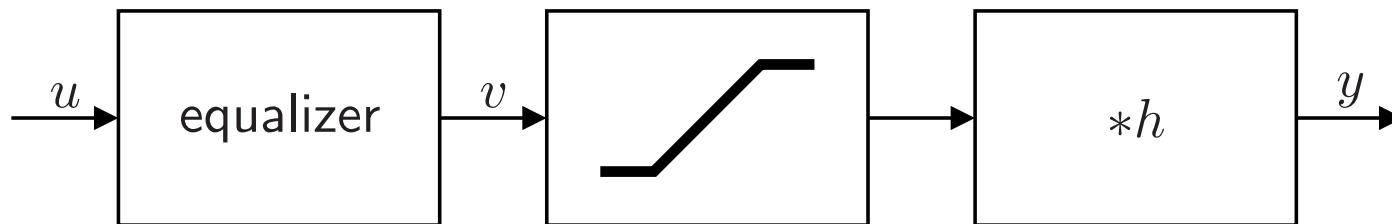
- robust KF requires solution of QP with 95 vars, 15 eqs, 30 ineqs
- automatically generated code solves QP in $120 \mu\text{s}$ (SDPT3: 120 ms)
- standard Kalman filter update requires $10 \mu\text{s}$

Linearizing pre-equalizer

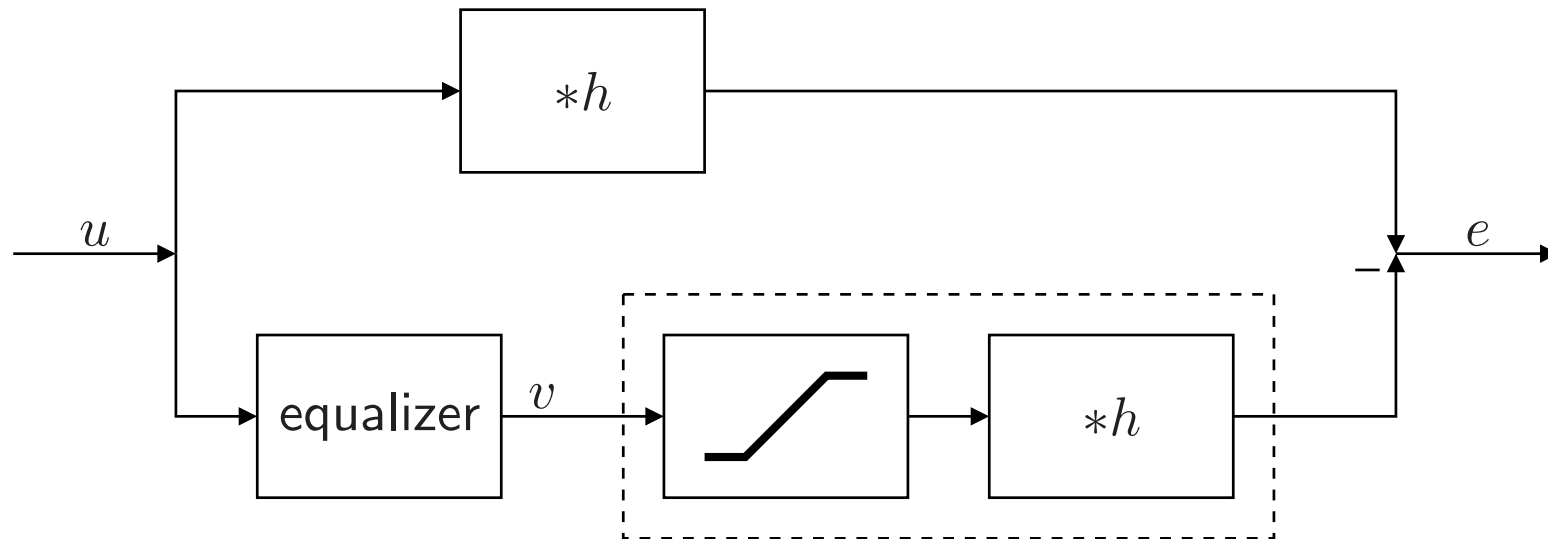
- linear dynamical system with input saturation



- we'll design pre-equalizer to compensate for saturation effects



Linearizing pre-equalizer



- goal: minimize error e (say, in mean-square sense)
- pre-equalizer has T sample look-ahead capability

- system: $x_{t+1} = Ax_t + B \text{sat}(v_t), \quad y_t = Cx_t$
- (linear) reference system: $x_{t+1}^{\text{ref}} = Ax_t^{\text{ref}} + Bu_t, \quad y_t^{\text{ref}} = Cx_t^{\text{ref}}$
- $e_t = Cx_t^{\text{ref}} - Cx_t$
- state error $\tilde{x}_t = x_t^{\text{ref}} - x_t$ satisfies

$$\tilde{x}_{t+1} = A\tilde{x}_t + B(u_t - v_t), \quad e_t = C\tilde{x}_t$$

- to choose v_t , solve QP

$$\begin{aligned} &\text{minimize} && \sum_{\tau=t}^{t+T} e_{\tau}^2 + \tilde{x}_{t+T+1}^T P \tilde{x}_{t+T+1} \\ &\text{subject to} && \tilde{x}_{\tau+1} = A\tilde{x}_{\tau} + B(u_{\tau} - v_{\tau}), \quad e_{\tau} = C\tilde{x}_{\tau}, \quad \tau = t, \dots, t+T \\ &&& |v_{\tau}| \leq 1, \quad \tau = t, \dots, t+T \end{aligned}$$

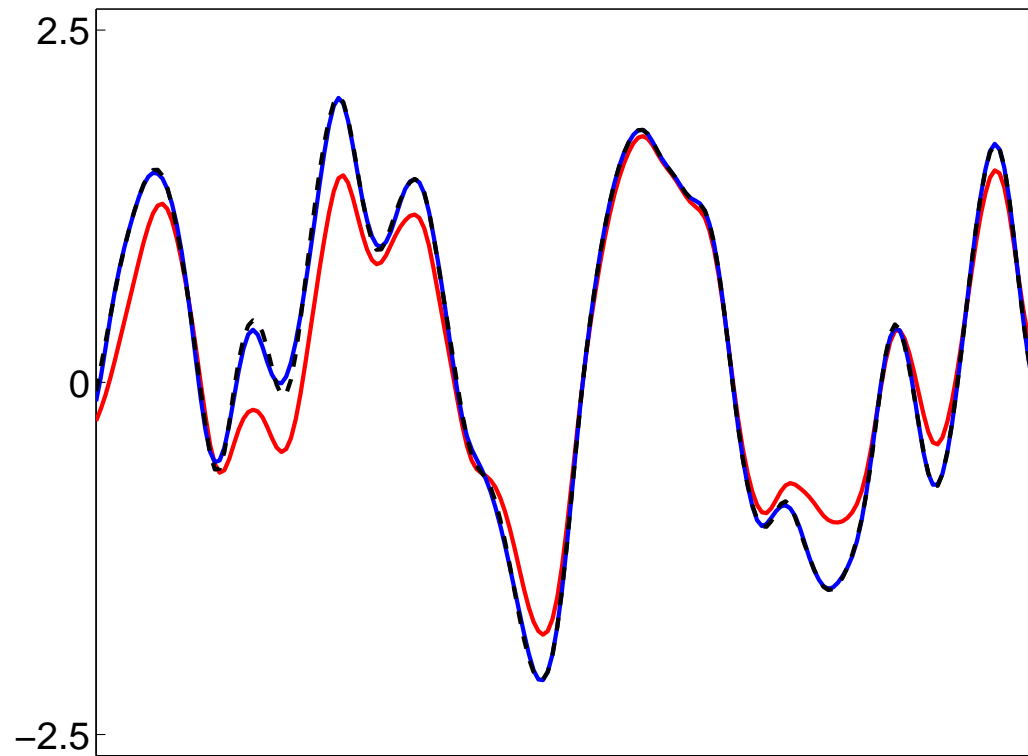
P gives final cost; obvious choice is output Grammian

Example

- state dimension $n = 3$; h decays in around 35 samples
- pre-equalizer look-ahead $T = 15$ samples
- input u random, saturates ($|u_t| > 1$) 20% of time

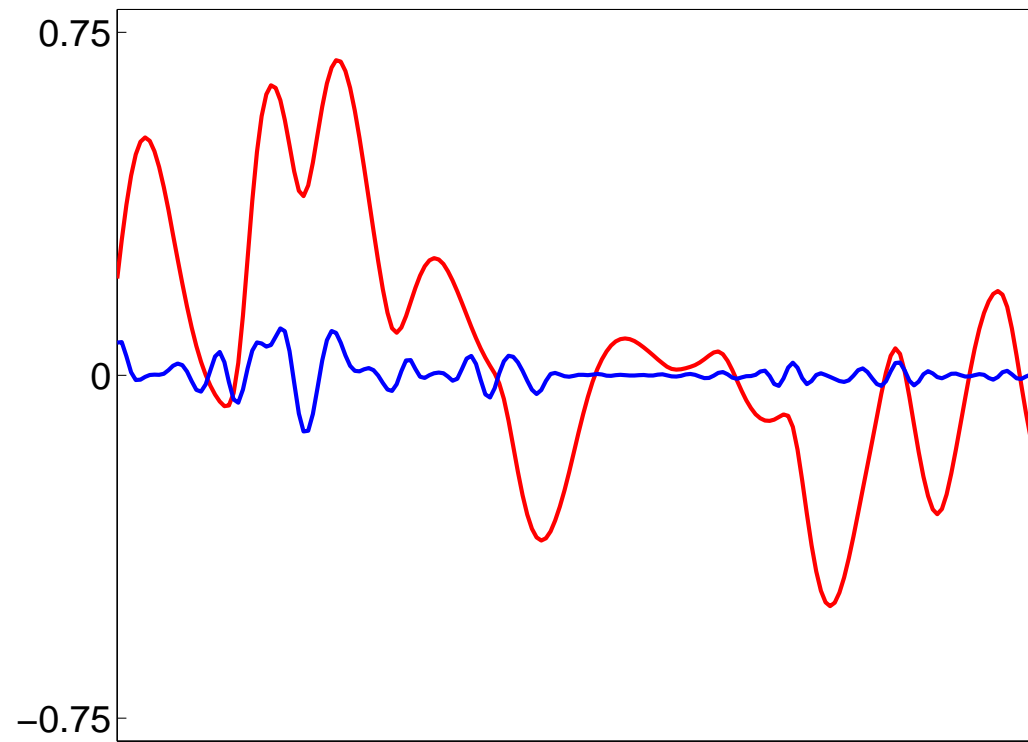
Outputs

desired (black), no compensation (red), equalized (blue)



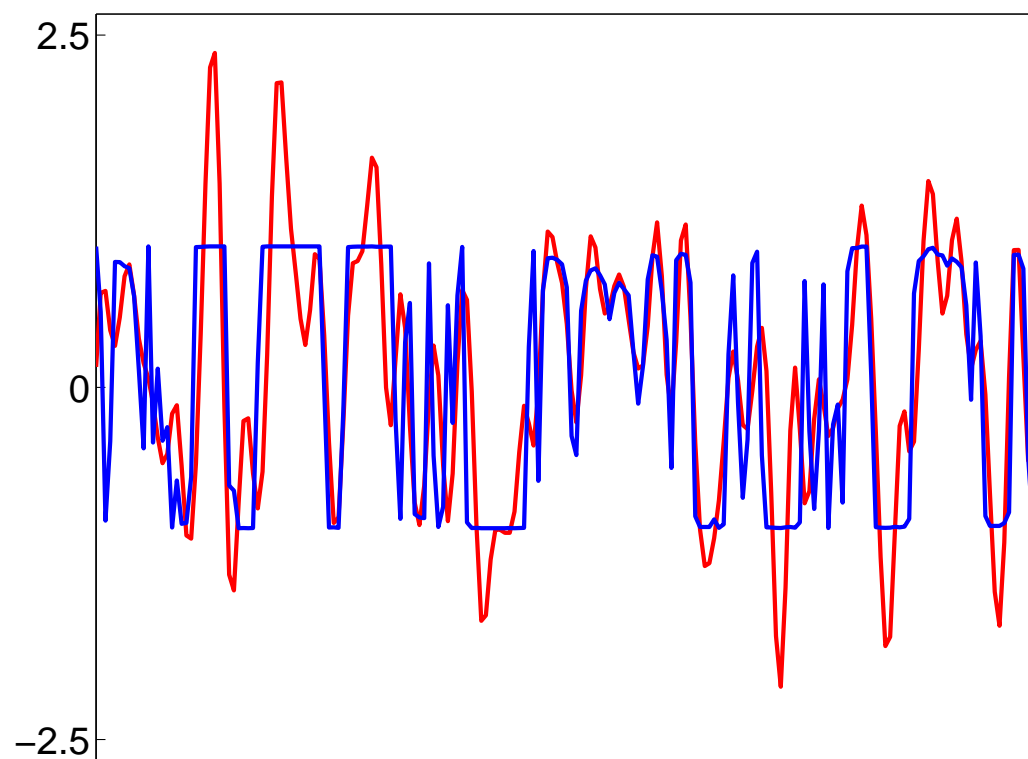
Errors

no compensation (red), with equalization (blue)



Inputs

no compensation (red), with equalization (blue)



Linearizing pre-equalizer solve time

- pre-equalizer problem reduces to QP with 96 vars, 63 eqs, 48 ineqs
- automatically generated code solves QP in $600\mu\text{s}$ (SDPT3: 310ms)

Constrained linear quadratic stochastic control

- linear dynamical system: $x_{t+1} = Ax_t + Bu_t + w_t$
 - $x_t \in \mathbf{R}^n$ is state; $u_t \in \mathcal{U} \subset \mathbf{R}^m$ is control input
 - w_t is IID zero mean disturbance
- $u_t = \phi(x_t)$, where $\phi : \mathbf{R}^n \rightarrow \mathcal{U}$ is (state feedback) policy
- objective: minimize average expected stage cost ($Q \geq 0, R > 0$)

$$J = \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbf{E} (x_t^T Q x_t + u_t^T R u_t)$$

- constrained LQ stochastic control problem: choose ϕ to minimize J

Constrained linear quadratic stochastic control

- optimal policy has form

$$\phi(z) = \operatorname{argmin}_{v \in \mathcal{U}} \{v^T R v + \mathbf{E} V(Az + Bv + w_t)\}$$

where V is Bellman function

- but V is hard to find/describe except when $\mathcal{U} = \mathbf{R}^m$ (in which case V is quadratic)
- many heuristic methods give suboptimal policies, *e.g.*
 - projected linear control
 - control-Lyapunov policy
 - model predictive control, certainty-equivalent planning

Control-Lyapunov policy

- also called approximate dynamic programming, horizon-1 model predictive control
- CLF policy is

$$\phi_{\text{clf}}(z) = \underset{v \in \mathcal{U}}{\operatorname{argmin}} \{v^T R v + \mathbf{E} V_{\text{clf}}(Az + Bv + w_t)\}$$

where $V_{\text{clf}} : \mathbf{R}^n \rightarrow \mathbf{R}$ is the control-Lyapunov function

- evaluating $u_t = \phi_{\text{clf}}(x_t)$ requires solving an optimization problem at **each step**
- many tractable methods can be used to find a good V_{clf}
- often works really well

Quadratic control-Lyapunov policy

- assume
 - polyhedral constraint set: $\mathcal{U} = \{v \mid Fv \leq g\}$, $g \in \mathbf{R}^k$
 - quadratic control-Lyapunov function: $V_{\text{clf}}(z) = z^T P z$
- evaluating $u_t = \phi_{\text{clf}}(x_t)$ reduces to solving QP

$$\begin{array}{ll} \text{minimize} & v^T R v + (Az + Bv)^T P (Az + Bv) \\ \text{subject to} & Fv \leq g \end{array}$$

Control-Lyapunov policy evaluation times

- t_{clf} : time to evaluate $\phi_{\text{clf}}(z)$
- t_{lin} : linear policy $\phi_{\text{lin}}(z) = Kz$
- t_{kf} : Kalman filter update
- (SDPT3 times around $1000\times$ larger)

n	m	k	$t_{\text{clf}} (\mu s)$	$t_{\text{lin}} (\mu s)$	$t_{\text{kf}} (\mu s)$
15	5	10	35	1	1
50	15	30	85	3	9
100	10	20	67	4	40
1000	30	60	298	130	8300

Outline

- Real-time embedded convex optimization
- Examples
- Parser/solvers for convex optimization
- Code generation for real-time embedded convex optimization

Parser/solvers for convex optimization

- specify convex problem in natural form
 - declare optimization variables
 - form convex objective and constraints using a specific set of atoms and calculus rules (**disciplined convex programming**)
- problem is convex-by-construction
- easy to parse, automatically transform to standard form, solve, and transform back
- implemented using object-oriented methods and/or compiler-compilers
- huge gain in productivity (rapid prototyping, teaching, research ideas)

Example: cvx

- parser/solver written in Matlab
- convex problem, with variable $x \in \mathbf{R}^n$; A, b, λ, F, g constants

$$\begin{aligned} & \text{minimize} && \|Ax - b\|_2 + \lambda\|x\|_1 \\ & \text{subject to} && Fx \leq g \end{aligned}$$

- cvx specification:

```
cvx_begin
    variable x(n)           % declare vector variable
    minimize (norm(A*x-b,2) + lambda*norm(x,1))
    subject to F*x <= g
cvx_end
```

when `cvx` processes this specification, it

- verifies convexity of problem
- generates equivalent cone problem (here, an SOCP)
- solves it using SDPT3 or SeDuMi
- transforms solution back to original problem

the `cvx` code is easy to read, understand, modify

Outline

- Real-time embedded convex optimization
- Examples
- Parser/solvers for convex optimization
- Code generation for real-time embedded convex optimization

General vs. embedded solvers

- general solver (say, for QP)
 - handles single problem instances with any dimensions, sparsity pattern
 - typically optimized for large problems
 - must deliver high accuracy
 - variable execution time: stops when tolerance achieved
- embedded solver
 - solves many instances of the same problem family (dimensions, sparsity pattern) with different data
 - solves small or smallish problems
 - can deliver lower (application dependent) accuracy
 - often must satisfy hard real-time deadline

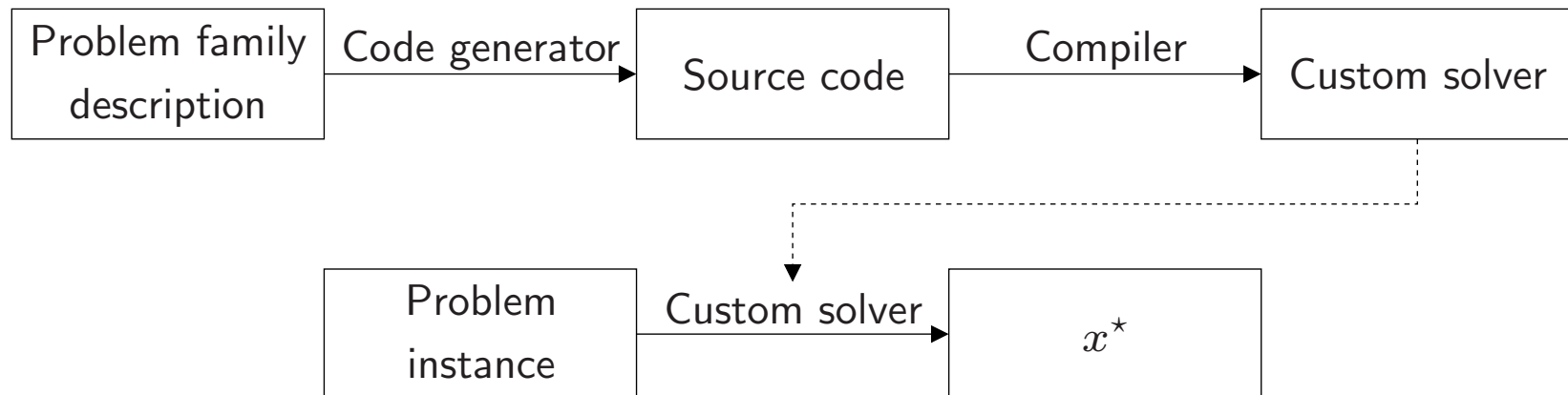
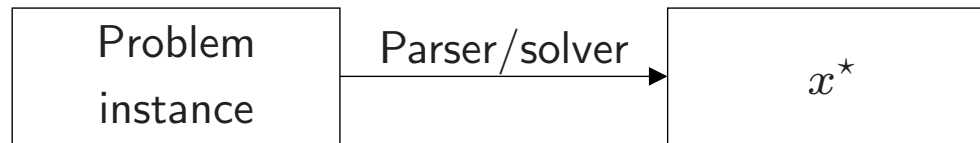
Embedded solvers

- (if a general solver works, use it)
- otherwise, develop custom code
 - by hand
 - automatically via code generation
- can exploit known sparsity pattern, data ranges, required tolerance at solver code development time
- we've had good results with interior-point methods; other methods (*e.g.*, active set, first order) might work well too
- typical speed-up over (efficient) general solver: **100–10000**×

Convex optimization solver generation

- specify convex problem **family** in natural form, via disciplined convex programming
 - declare optimization variables, parameters
 - form convex objective and constraints using a specific set of atoms and calculus rules
- code generator
 - analyzes problem structure (dimensions, sparsity, . . .)
 - chooses elimination ordering
 - generates solver code for specific problem family
- idea:
 - spend (perhaps much) time generating code
 - save (hopefully much) time solving problem instances

Parser/solver vs. code generator



cvxgen code generator

- handles problems transformable to QP
- accepts high-level problem family description
- generates flat C source, auxiliary files, documentation
 - no libraries, almost branch-free
- primal-dual interior-point method with iteration limit
 - direct LDL^T factorization of KKT matrix
 - regularization and iterative refinement for extreme reliability
 - (slow) method to determine static variable ordering
 - explicit factorization code generated

cvxgen example specification

dimensions

```
n = 30 # assets.
```

parameters

```
alpha (n) # mean returns.
```

```
lambda nonnegative # risk aversion.
```

```
Sigma (n,n) psd # covariance.
```

```
eta nonnegative # limit on total short position.
```

variables

```
w (n) # asset weights.
```

maximize

```
alpha'*w - lambda*quad(w,Sigma) # risk adjusted mean return.
```

subject to

```
sum(w) == 1 # budget constraint.
```

```
sum(neg(w)) <= eta # limit on total short position.
```

Sample solve times for cvxgen generated code

	<i>Scheduling</i>	<i>Battery</i>	<i>Suspension</i>
Variables	279	153	104
Constraints	465	357	165
CVX, Intel i7	4.2 s	1.3 s	2.6 s
CVXGEN, Intel i7	850 μ s	360 μ s	110 μ s
CVXGEN, Atom	7.7 ms	4.0 ms	1.0 ms

Conclusions

- can solve convex problems on **millisecond, microsecond** time scales
 - (using existing algorithms, but not using existing codes)
 - there should be many applications
- parser/solvers make rapid prototyping easy
- new code generation methods yield solvers that
 - are extremely fast, even competitive with ‘analytical methods’
 - can be embedded in real-time applications

References

- *Automatic Code Generation for Real-Time Convex Optimization* (Mattingley, Boyd)
- *Real-Time Convex Optimization in Signal Processing* (Mattingley, Boyd)
- *Code Generation for Receding Horizon Control* (Mattingley, Boyd)
- *Fast Evaluation of Quadratic Control-Lyapunov Policy* (Wang, Boyd)
- *Fast Model Predictive Control Using Online Optimization* (Wang, Boyd)
- `cvx` (Grant, Boyd, Ye)
- `cvxgen` (Mattingley, Boyd)

But my problem isn't convex

- local convexification (*e.g.*, sequential QP)
 - can even update problem data during interior-point iterations
- combinatorial problems: relax & round
- global guarantees go away; robustness harder to guarantee