Real-Time Embedded Convex Optimization

Stephen Boyd

joint work with Michael Grant, Jacob Mattingley, Yang Wang

Electrical Engineering Department, Stanford University

Outline

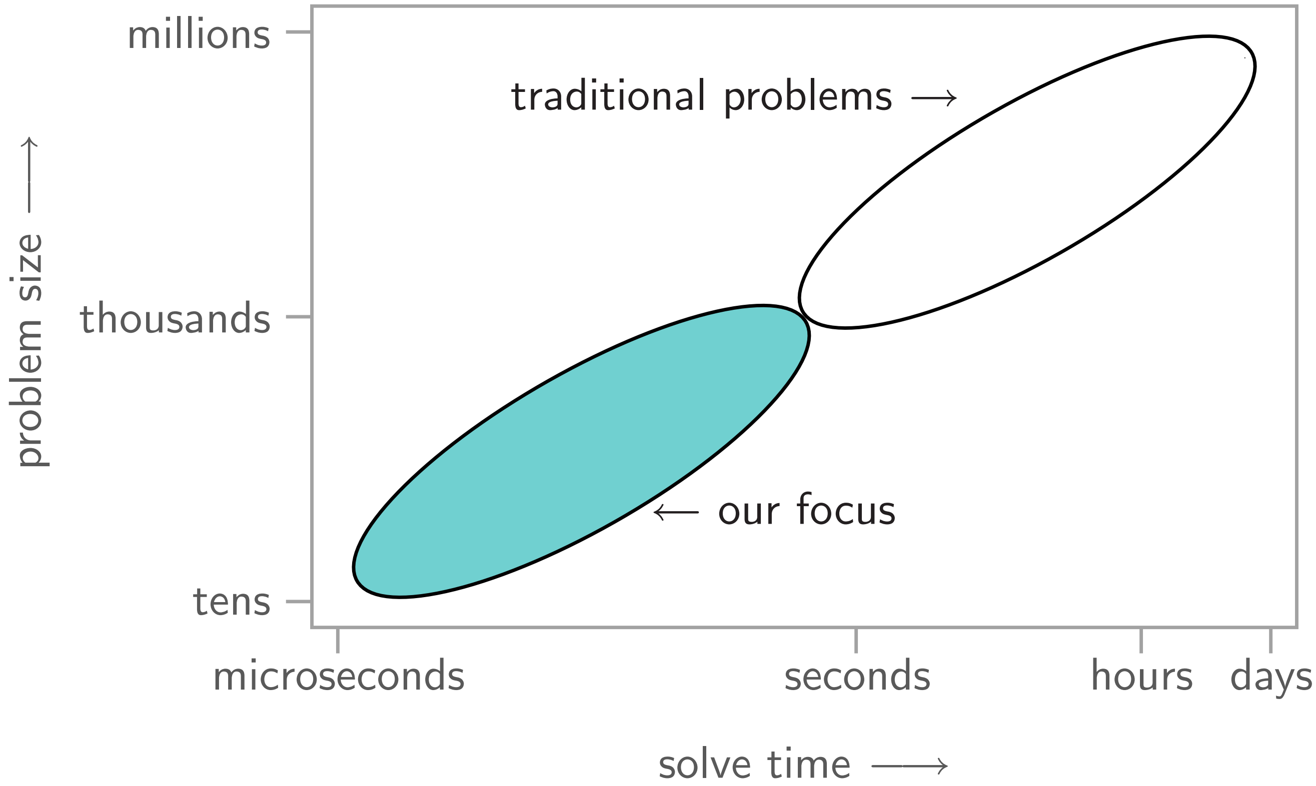
- Real-time embedded convex optimization
- Examples
- Parser/solvers for convex optimization
- Code generation for real-time embedded convex optimization

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

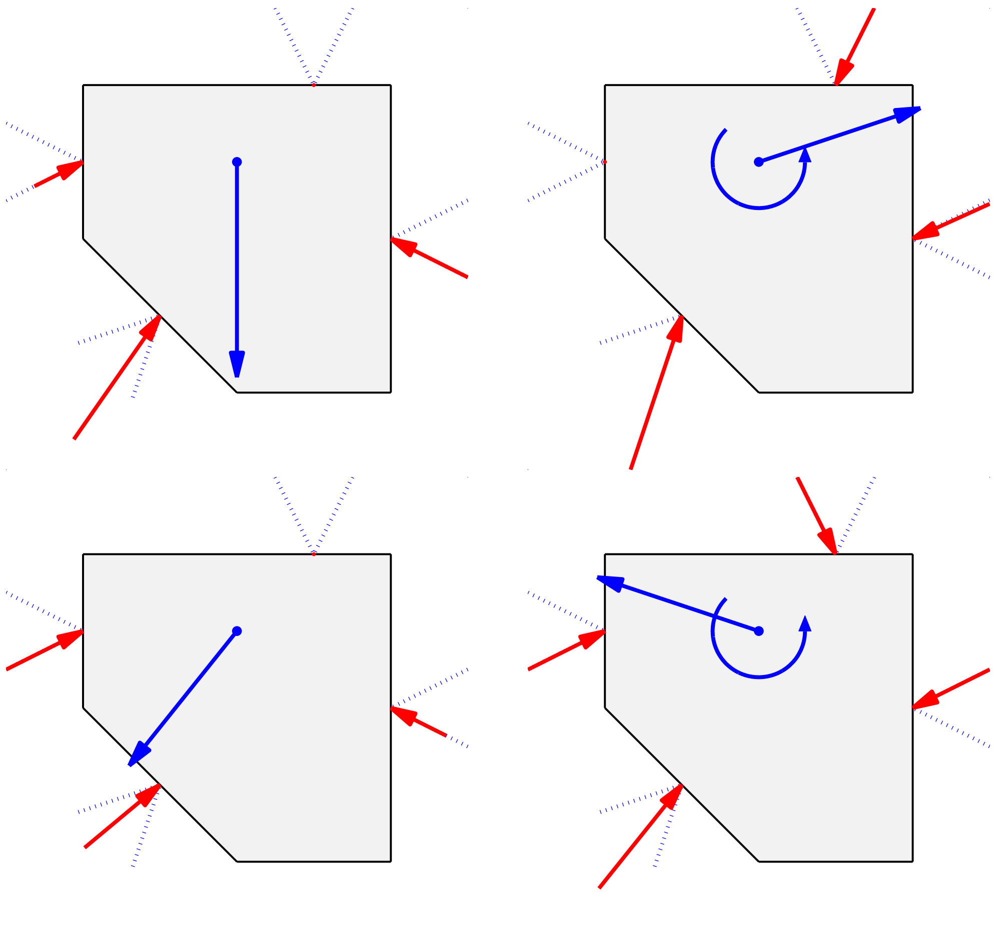
Grasp force optimization

- 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):

minimize	$\max_i \|f^{(i)}\|_2$	*max contact force*
subject to	$\sum_i Q^{(i)} f^{(i)} = f^{\mathrm{ext}}$	*force equillibrium*
	$\sum_i p^{(i)} \times (Q^{(i)} f^{(i)}) = \tau^{\mathrm{ext}}$	*torque equillibrium*
	$\mu_i f_z^{(i)} \geq \left(f_x^{(i)2} + f_y^{(i)2}\right)^{1/2}$	*friction cone constraints*

variables $f^{(i)} \in \mathbf{R}^3$, $i = 1, \ldots, K$ (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: 50μs (SDPT3: 100ms)

Robust Kalman filtering

- estimate state of a linear dynamical system driven by IID noise
- sensor measurements have occasional outliers (failures, jamming, . . .)
- model: $x_{t+1} = Ax_t + w_t, \quad y_t = Cx_t + v_t + z_t$
  - $w_t \sim \mathcal{N}(0, W)$, $v_t \sim \mathcal{N}(0, V)$
  - 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

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

  with variables x, v (simple analytic solution)

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

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

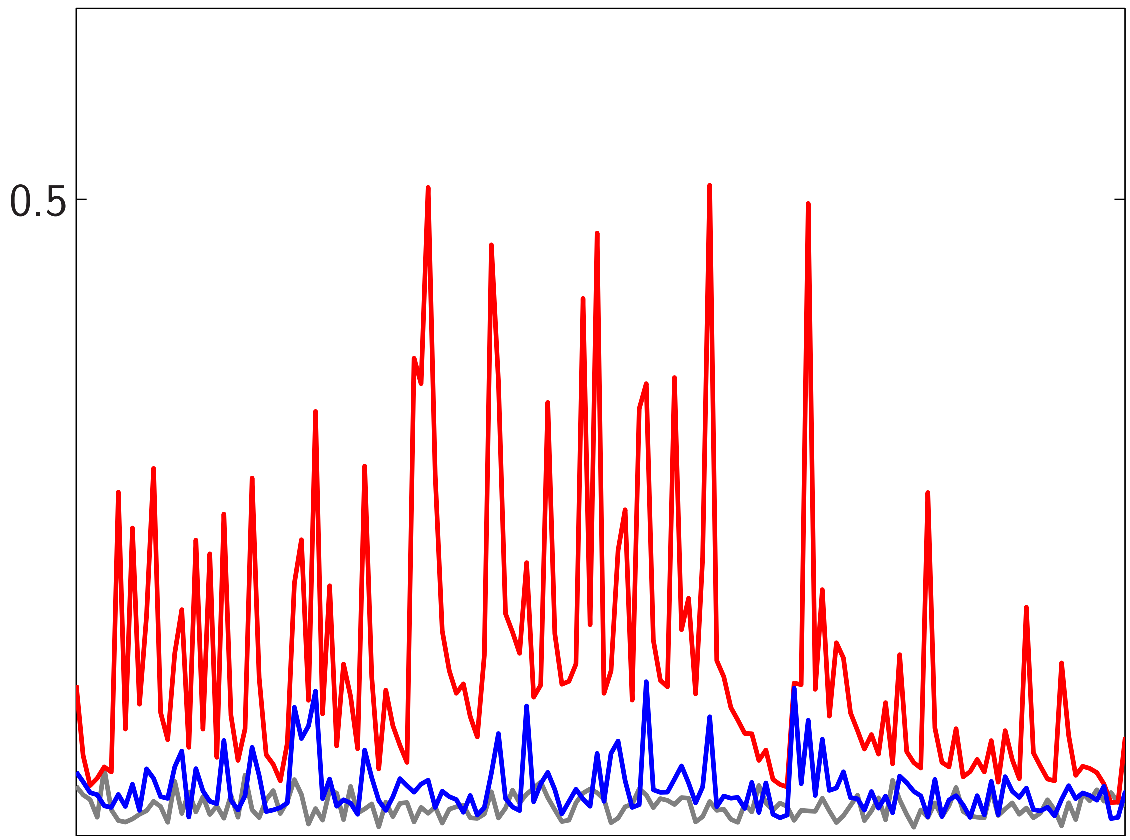
with variables x, v, z (requires solving a QP)

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)

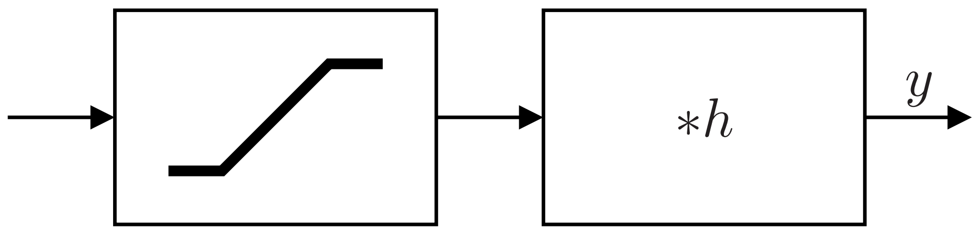


Robust Kalman filter solve time

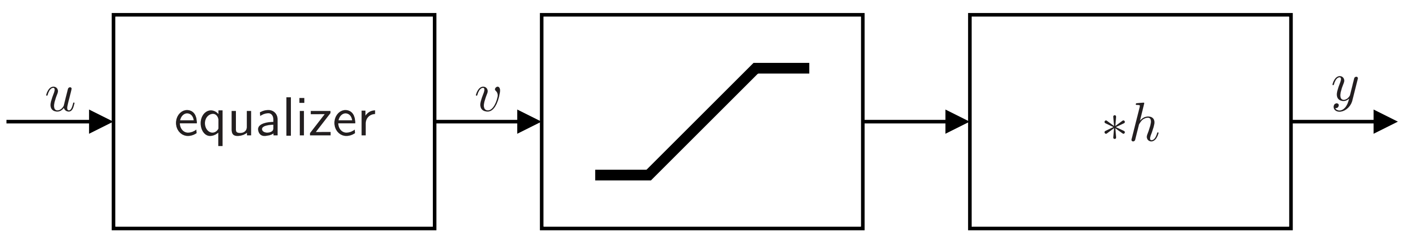
- robust KF requires solution of QP with 95 vars, 15 eqs, 30 ineqs
- automatically generated code solves QP in 120 μs (SDPT3: 120 ms)
- standard Kalman filter update requires 10 μs

Linearizing pre-equalizer

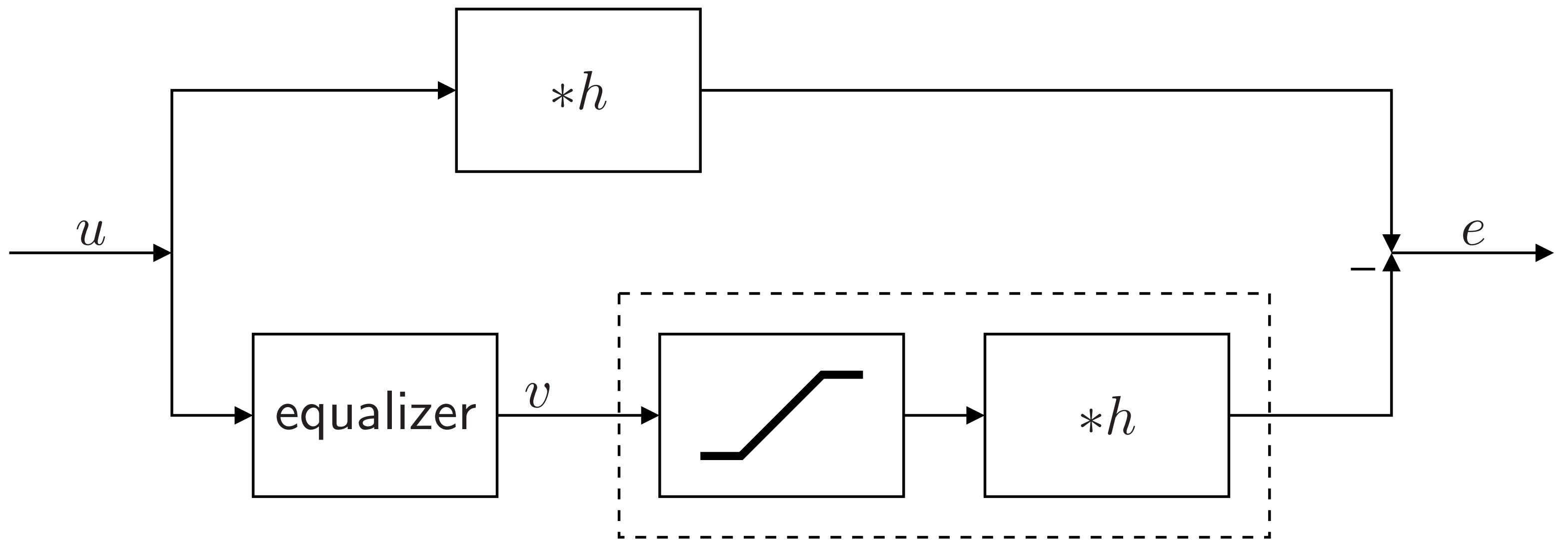
- linear dynamical system with input saturation



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

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

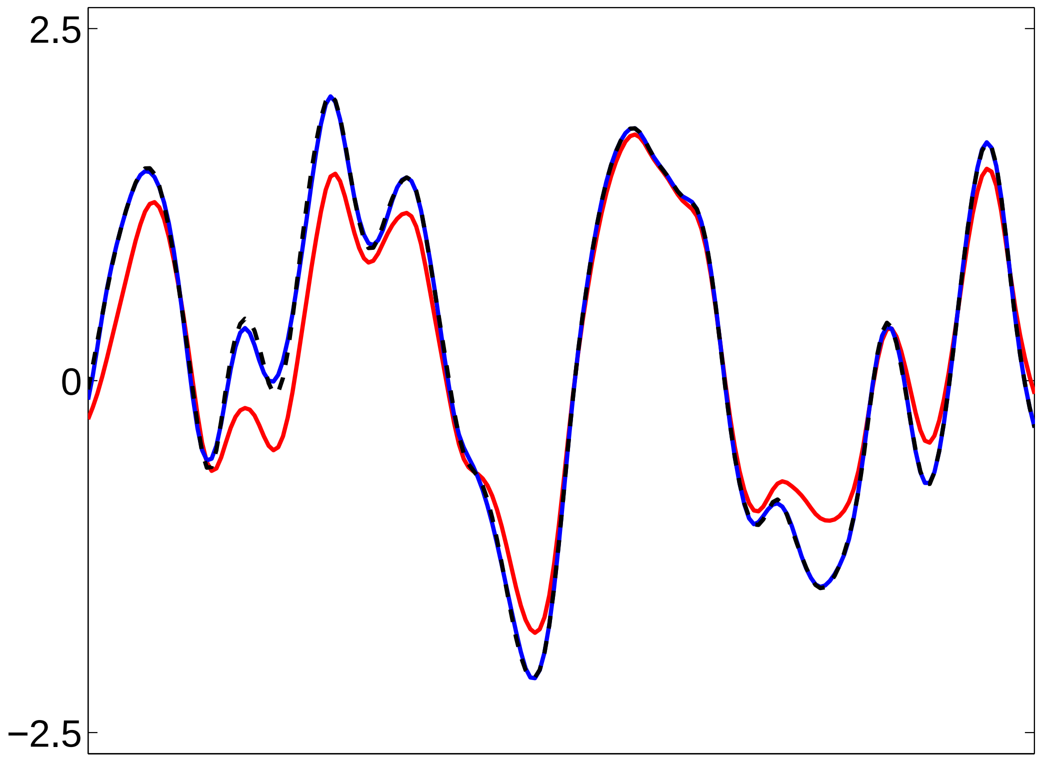
P gives final cost; obvious choice is output Grammian

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

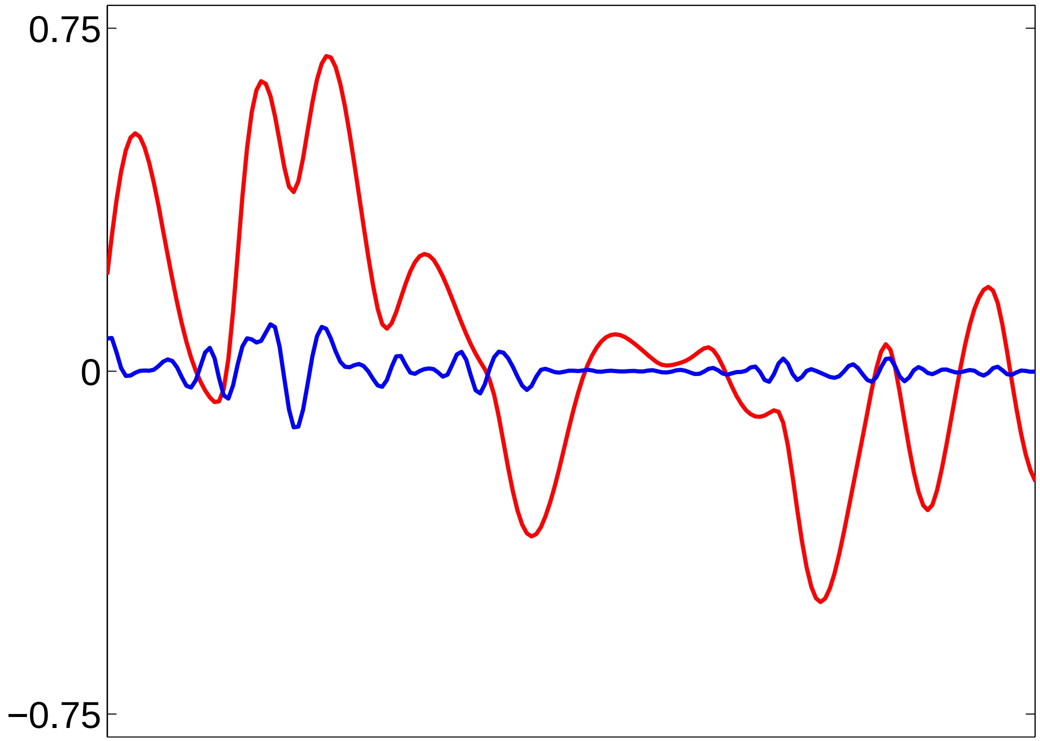
Outputs

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



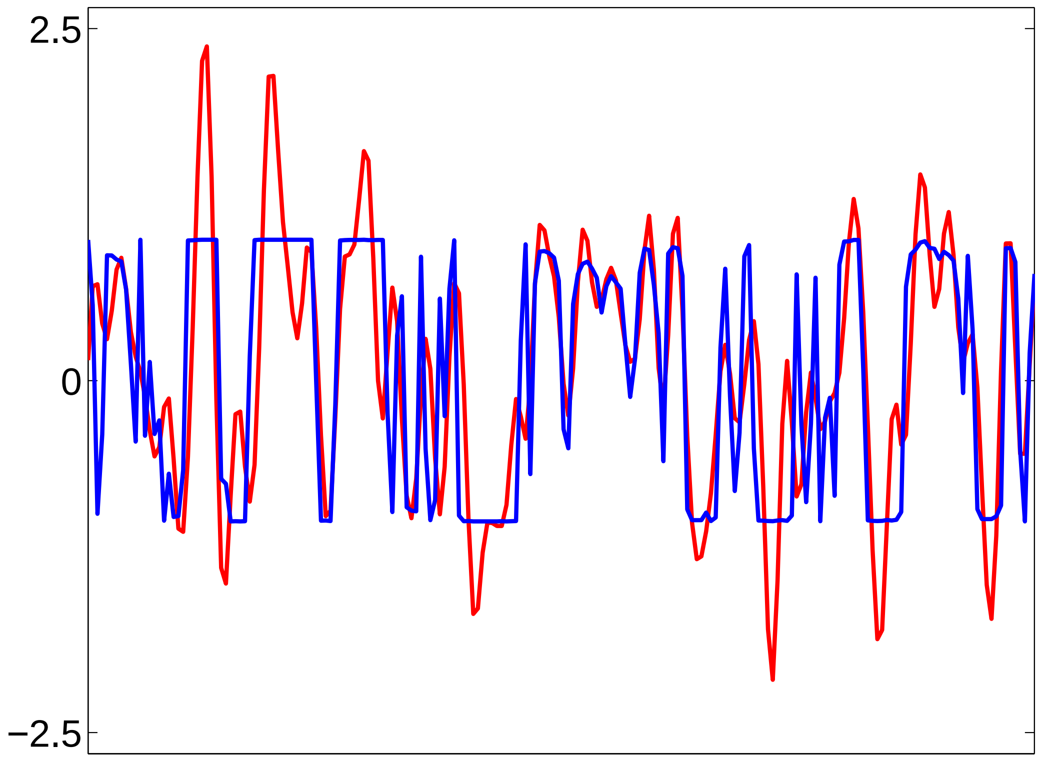
Errors

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 600μs (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 $\phi : \mathbf{R}^n \to \mathcal{U}$ is (state feedback) policy

- objective: minimize average expected stage cost ($Q \geq 0$, $R > 0$)

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

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

Constrained linear quadratic stochastic control

- optimal policy has form

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

  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

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

  where $V_{\rm clf} : \mathbf{R}^n \rightarrow \mathbf{R}$ is the control-Lyapunov function
- evaluating $u_t = \phi_{\rm clf}(x_t)$ requires solving an optimization problem at **each step**
- many tractable methods can be used to find a good $V_{\rm clf}$
- 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_{clf} (μs)	t_{lin} (μs)	t_{kf} (μs)
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

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

- 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

The same example, transformed by 'hand'

transform problem to SOCP, call SeDuMi, reconstruct solution:

```
% Set up big matrices.
[m,n] = size(A); [p,n] = size(F);
AA = [speye(n), -speye(n), speye(n), sparse(n,p+m+1); ...
      F, sparse(p,2*n), speye(p), sparse(p,m+1); ...
      A, sparse(m,2*n+p), speye(m), sparse(m,1)];
bb = [zeros(n,1); g; b];
cc = [zeros(n,1); gamma*ones(2*n,1); zeros(m+p,1); 1];
K.f = m; K.l = 2*n+p; K.q = m + 1;     % specify cone
xx = sedumi(AA, bb, cc, K);            % solve SOCP
x = x(1:n);                            % extract solution
```

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

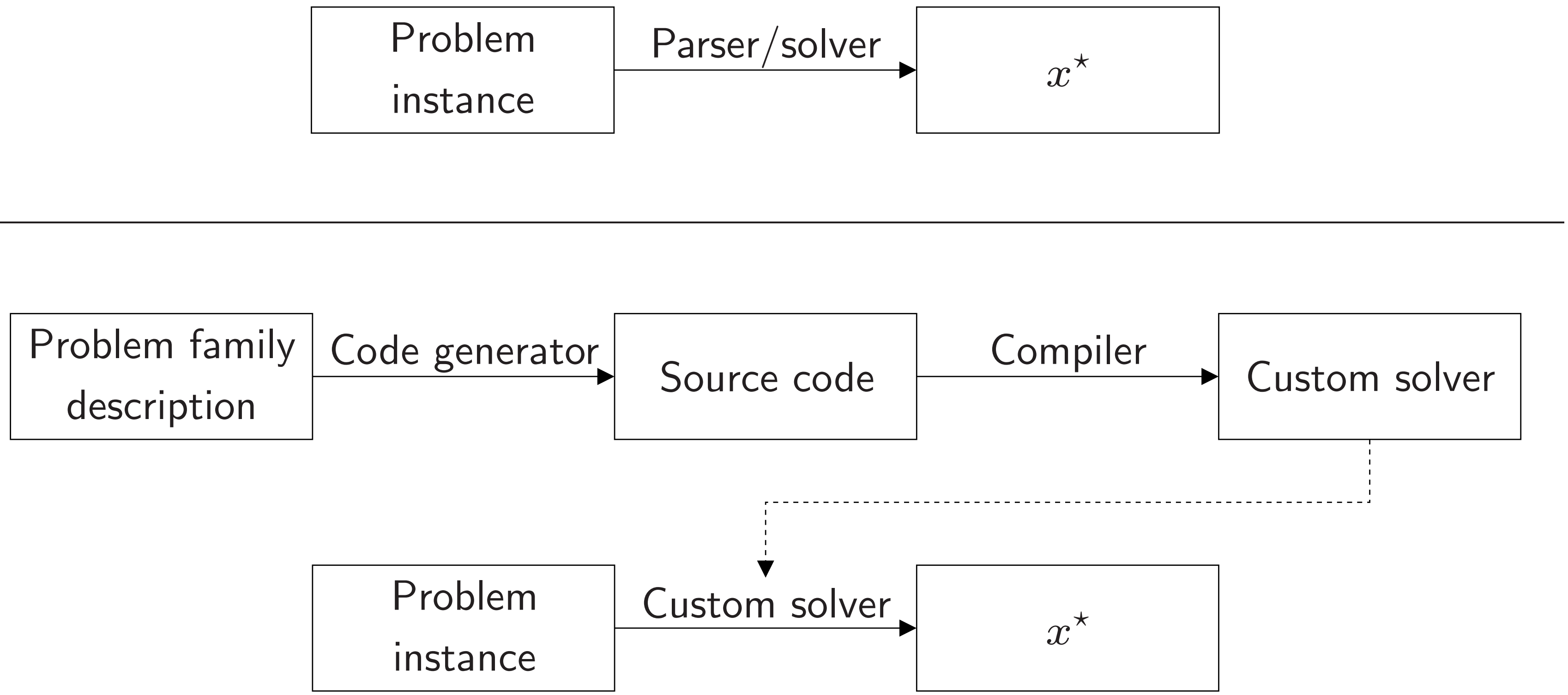
Embedded solvers

- (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**$\times$

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 generation



Example: `cvxmod`

- written in Python
- QP family, with variable $x \in \mathbf{R}^n$, parameters P, q, g, h

$$\begin{array}{ll} \text{minimize} & x^T P x + q^T x \\ \text{subject to} & Gx \leq h, \quad Ax = b \end{array}$$

- `cvxmod` specification:

```
A = matrix(...); b = matrix(...)
P = param('P', n, n, psd=True); q = param('q', n)
G = param('G', m, n); h = param('h', m)
x = optvar('x', n)
qpfam = problem(minimize(quadform(x, P) + tp(q)*x),
                [G*x <= h, A*x == b])
```

`cvxmod` code generation

- generate solver for problem family `qpfam` with

```
qpfam.codegen()
```

- output includes `qpfam/solver.c` and ancillary files
- solve instance with (C function call)

```
status = solve(params, vars, work);
```

Using cvxmod generated code

```
#include "solver.h"
int main(int argc, char **argv) {
    // Initialize structures at application start-up.
    Params params = init_params();
    Vars vars = init_vars();
    Workspace work = init_work(vars);
    // Enter real-time loop.
    for (;;) {
        update_params(params);
        status = solve(params, vars, work);
        export_vars(vars);
}}
```

`cvxmod` code generator

(preliminary implementation)

- handles problems transformable to QP
- primal-dual interior-point method with iteration limit
- direct LDL^T factorization of KKT matrix
- (slow) method to determine variable ordering (at code generation time)
- explicit factorization code generated

Sample solve times for `cvxmod` generated code

problem family	vars	constrs	SDPT3 (ms)	`cvxmod` (ms)
`control1`	140	190	250	0.4
`control2`	360	1080	1400	2.0
`control3`	1110	3180	3400	13.2
`order_exec`	20	41	490	0.05
`net_utility`	50	150	130	0.23
`actuator`	50	106	300	0.17
`robust_kalman`	95	45	120	0.12

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)
- *Fast Evaluation of Quadratic Control-Lyapunov Policy* (Wang, Boyd)
- *Fast Model Predictive Control Using Online Optimization* (Wang, Boyd)
- `cvx` (Grant, Boyd, Ye)
- `cvxmod` (Mattingley, Boyd)

all available on-line, but `cvxmod` code gen not yet ready for prime-time