Embedded Convex Optimization

Jacob Mattingley

joint work with Stephen Boyd

Electrical Engineering Department, Stanford University

March 2010

Overview

- convex optimization
- embedded convex optimization
- tools
- cvxmod code generation
- active suspension example
- a few implementation details

Optimization problems

- useful in many fields, especially engineering and data analysis
- often hard to solve
- standard formulation:

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & x \in \mathcal{C} \end{array}$$

  x is the variable, f is the objective, $\mathcal{C}$ is the constraint set

- **important subset:** convex optimization problems

Convex optimization problems

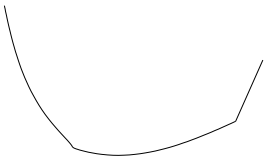
$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & x \in \mathcal{C} \end{array}$$

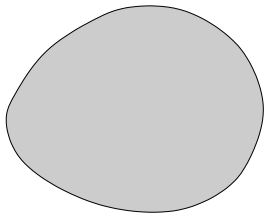
- f is convex (graph of f curves upward):

$$\theta \in [0,1] \implies f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta) f(y)$$

- $\mathcal{C}$ is convex (closed under averaging):

$$x, y \in \mathcal{C},\ \theta \in [0,1] \implies \theta x + (1-\theta)y \in C$$

- f 

$\mathcal{C}$ 

Properties of convex optimization problems

- usually no analytical solution (except least squares, etc)
- can solve **extremely well** (both theoretically and in practice)
- can get globally optimal solution and a certificate
- can easily solve problems with $\leq 10^5$ variables and constraints
- fairly complete theory

Applications of convex optimization

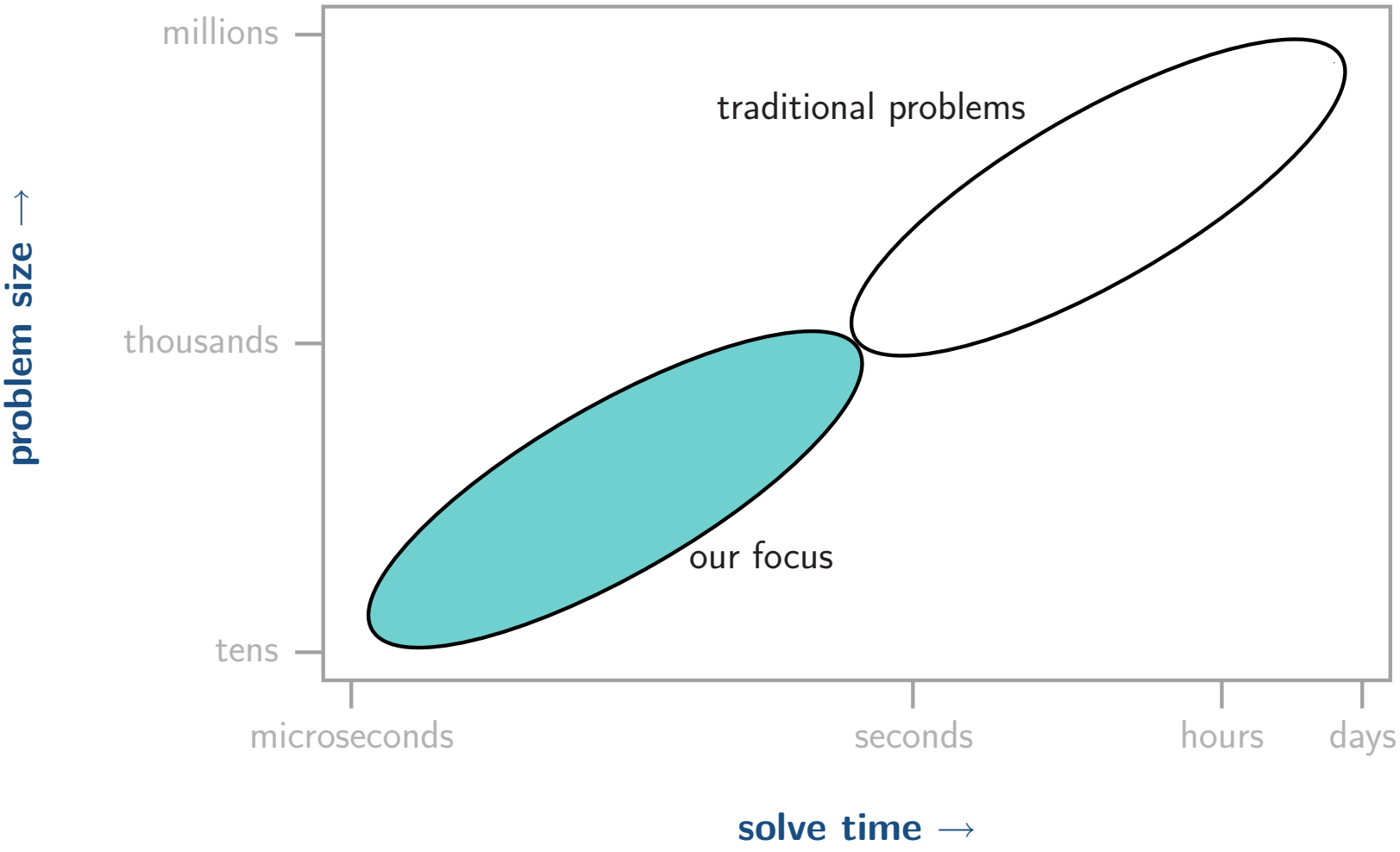
- come up more often than you might think
- convexity is not always obvious
- applications include
  - control
  - signal processing
  - image processing
  - communications, networking
  - analog and digital circuit design
  - statistics, machine learning
  - finance
  - combinatorial optimization

Overview

- convex optimization
- **embedded convex optimization**
- tools
- cvxmod code generation
- active suspension example
- a few implementation details

Embedded convex optimization

- small, fixed problems
- **millisecond** or even **microsecond** time scales
- online applications, possibly with real-time deadlines
- automatic code generation can make it easier


millions
thousands
tens
problem size →
traditional problems
our focus
microseconds
seconds
hours
days
solve time →

Embedded optimization applications

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

Overview

- convex optimization
- embedded convex optimization
- **tools**
- cvxmod code generation
- active suspension example
- a few implementation details

Tools for convex optimization

- traditional domain: parser/solver
  - simplifies transformation to standard form
  - checks convexity, data validity etc
  - calls solver
  - transforms solution back to original form

Parser/solvers: problem transformation example

- ℓ_1-regularized least-squares:

$$\text{minimize} \quad \|Ax - b\|_2^2 + \lambda \|x\|_1$$

- convex objective; non-standard form
- introduce new variable t, same size as x; standard form is

$$\begin{aligned} \text{minimize} \quad & \begin{bmatrix} x \\ t \end{bmatrix}^T \begin{bmatrix} A^T A & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ t \end{bmatrix} + \begin{bmatrix} -2A^T b \\ \lambda \mathbf{1} \end{bmatrix}^T \begin{bmatrix} x \\ t \end{bmatrix} \\ \text{subject to} \quad & \begin{bmatrix} I & -I \\ -I & -I \end{bmatrix} \begin{bmatrix} x \\ t \end{bmatrix} \leq 0 \end{aligned}$$

- equivalent problem, with $2n$ variables and $2n$ constraints

Parser/solvers: problem transformations

- many such transformations; sometimes obvious, often not
- rules correspond to basic rules of convex analysis, eg,
  - sum, max of convex functions is convex
  - composition: $h \circ g$ is convex if h convex and increasing, g convex
  - partial minimization: $g(x) = \inf_{y \in \mathcal{C}} f(x, y)$ is convex if f, $\mathcal{C}$ convex
  - many others . . .

General versus embedded solvers

- if a parser/solver works, use it
- otherwise, develop custom code
  - by hand: painful, difficult to get exactly right
  - automatically via code generation

Tools for convex optimization

- traditional domain: parser/solver
  - simplifies transformation to standard form
  - checks convexity, data validity etc
  - calls solver
  - transforms solution back to original form
- embedded optimization: code generator
  - simplifies transformation to standard form
  - checks convexity, data validity etc
  - **creates custom solver**
  - generates code for specific transformations (only)

General versus embedded solvers

- parser/solvers
  - many problem instances, with any structure or size
  - typically optimized for larger problems
  - must deliver high accuracy
  - variable execution time; stop after achieving tolerance
- embedded convex optimization solvers
  - every problem instance from the same family
  - smaller problems
  - reduced accuracy often ok
  - often, hard real-time deadlines

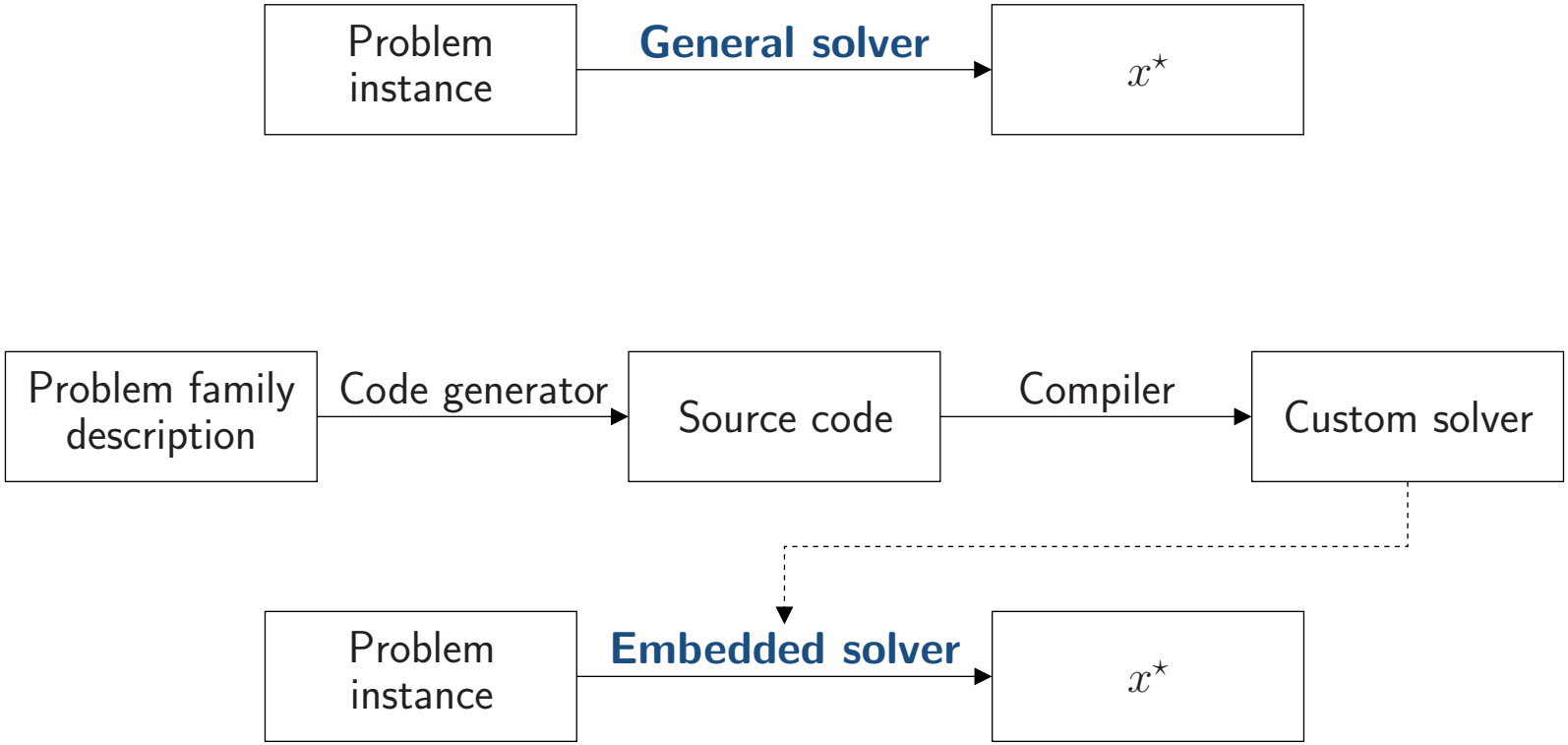
Overview

- convex optimization
- embedded convex optimization
- tools
- **cvxmod code generation**
- active suspension example
- a few implementation details

Code generation

- basic idea: exploit properties known at code development time
  - problem structure/sparsity
  - data ranges
  - required tolerance
- **spend time now, save time later**
- we've had good results with interior-point methods;
  (other methods, eg, active set, first order might work well too)
- typical speed-up over parser/solver: 100–10,000$\times$ (!)
- code generators aren't a new idea: many examples (ptolemy)

Parser/solvers versus code generation



Tools: examples

- parser/solvers
  - AMPL, GAMS
  - YALMIP, CVX
  - many others . . .
- code generators for convex optimization
  - today's talk: **cvxmod**

cvxmod

- Mattingley, Boyd
- written in Python
- started life as a parser/solver, targeting CVXOPT (Vandenberghe, Dahl)
- now has experimental code generation capability, targeting C
- `http://cvxmod.net/`, although code generation not yet available

cvxmod code generation

- currently targets small QPs only:

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

- plain, library-free C code (will come back to this)
- written in Python, and with combined Python/C templates
- sometimes works surprisingly well: up to 10,000$\times$ faster than CVX+SDPT3

Problem specification

- simple example, with variable $x \in \mathbf{R}^5$:

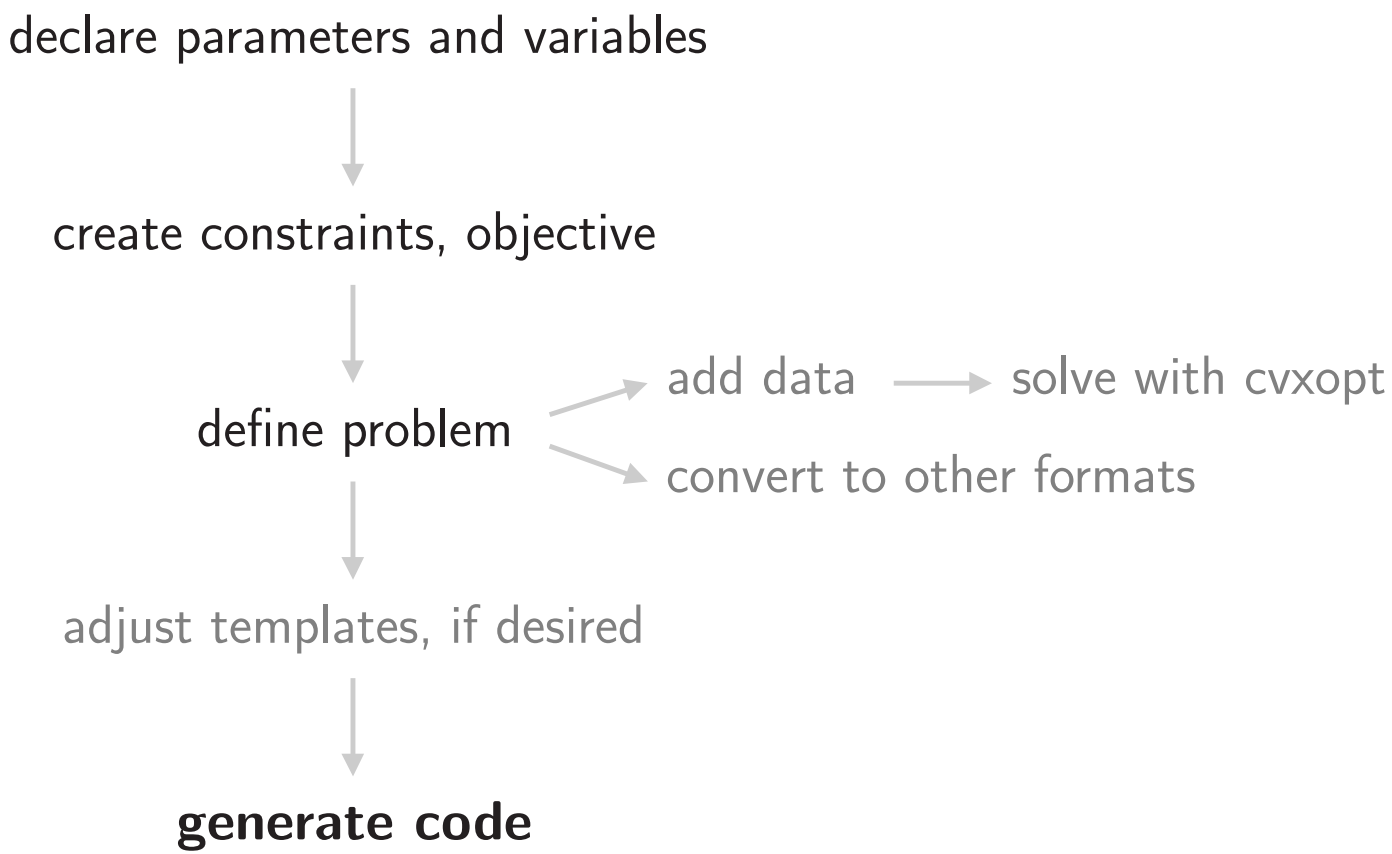
$$\begin{array}{ll} \text{minimize} & x^T Q x \\ \text{subject to} & |x| \leq 1, \quad \sum_i x_i = 10, \quad Ax \geq b \end{array}$$

- cvxmod (Python) specification:

```
A = matrix(...)  # 3x5
b = param('b', 3)
Q = param('Q', 5, 5, psd=True)
x = optvar('x', 5)

qpfam = problem(
  minimize(quadform(x, Q)),
  [abs(x) <= 1, sum(x) == 10, A*x >= b]
)
```

Using cvxmod



Code generation

- in Python, generate solver for problem family `qpfam` with

```
qpfam.gen()
```

- **generates:**

`solver.c`	provides `solve(params, vars, work)`
`solver.h`	includes and defintions
`testsolver.c`	simple driver using Python-specified default values
`prob.tex`	latex problem and transformation description
`csolve.c`	matlab mex interface
various others	example files, data files, partitioned implementation files

Using generated C code

```c
#include "solver.h"
int main(int argc, char **argv) {
  Params params = initparams();
  Vars vars = initvars();
  Workspace work = initwork(vars);

  for (;;) { // Real-time loop.
    // Get new parameter values...
    status = solve(params, vars, work);
    // Test status, export variables...
  }
}
```

Sample solve times

problem family	vars	constrs	CVX	cvxmod	speedup
`control1`	140	190	250	0.4	600 ×
`control2`	360	1080	1400	2.0	700 ×
`control3`	1110	3180	3400	13.2	250 ×
`order_exec`	20	41	490	0.05	10,000 ×
`net_utility`	50	150	130	0.23	600 ×
`actuator`	50	106	300	0.17	2,000 ×

- all times in ms
- CVX is using SDPT3; cvxmod is using generated C code
- speedup is best with very small problems
- additional manual tuning could improve, say, `control3`

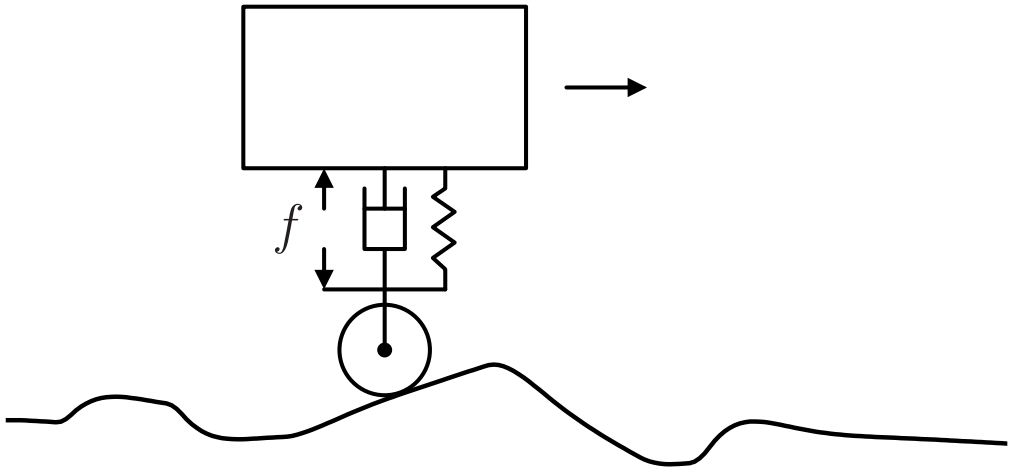
Why is cvxmod-generated code fast?

- efficient algorithm (primal-dual Mehrotra predictor-corrector)
- custom ordering method
- explicit, micro-managed code
- simple data structures, known memory requirements
- compiler optimizations

Overview

- convex optimization
- embedded convex optimization
- tools
- cvxmod code generation
- **active suspension example**
- a few implementation details

Active suspension control



- simple mass-spring-damper model of vehicle suspension
- additional actuator control, f
- minimize ride roughness, subject to various constraints
- solve as horizon-T MPC problem
- **use embedded convex optimization**

Suspension: discretized problem

$$
\begin{array}{ll}
\text{minimize} & \sum_{\tau=t}^{t+T} \|y''_\tau\|_2^2 \\
\text{subject to} & y''_\tau = -(K/M)(x_\tau)_1 - (D/M)(x_\tau)_2 + u_\tau, \\
& x_{\tau+1} = Ax_\tau + Bu_\tau \\
& u_\tau = (1/M)(f_\tau + Dh'_\tau + Kh_\tau), \\
& E^{\min} \leq (x_\tau)_1 - h_\tau \leq E^{\max}, \\
& F^{\min} \leq f_\tau \leq F^{\max}, \quad \tau = t, \ldots t+T \\
& x_{t+T} = 0
\end{array}
$$

- h terrain height; M, K, D, physical constants
- variables: y vehicle height, f actuator force, state $x = (y_\tau, y'_\tau)$
- minimize mean-square acceleration
- constraints on suspension length, actuator force

Suspension: Python declarations

```
# Parameters, which are placeholders for problem data.
x[0] = param('x0', 2)
A = param('A', 2, 2)
B = param('B', 2, 1)
h = param('h', T, 1)
D = param('D', pos=True)
Minv = param('1/M', pos=True)
# ...

# Optimization variables.
f = optvar('f', T)
for i in range(T):
  x[i+1] = optvar('x%d' % (i+1), 2)
  u[i] = optvar('u%d' % i)
```

Python problem description

```
# Build constraints.
constr = [f <= Fmax, f >= Fmin]
for i in range(T):
  constr += [
    x[i+1] == A*x[i] + B*u[i],
    u[i] == Minv*(f[i] + D*hdot[i] + K*h[i]),
    x[i][0] - h[i] <= Emax,
    x[i][0] - h[i] >= Emin
  ]

# Build the objective.
objv = 0
for i in range(T):
  ydd = -KMinv*x[i][0] - DMinv*x[i][1] + u[i]
  objv = objv + quadform(ydd)
```

Code generation

```
# Create the problem and generate code.
problem(minimize(objv), constr).gen()
```

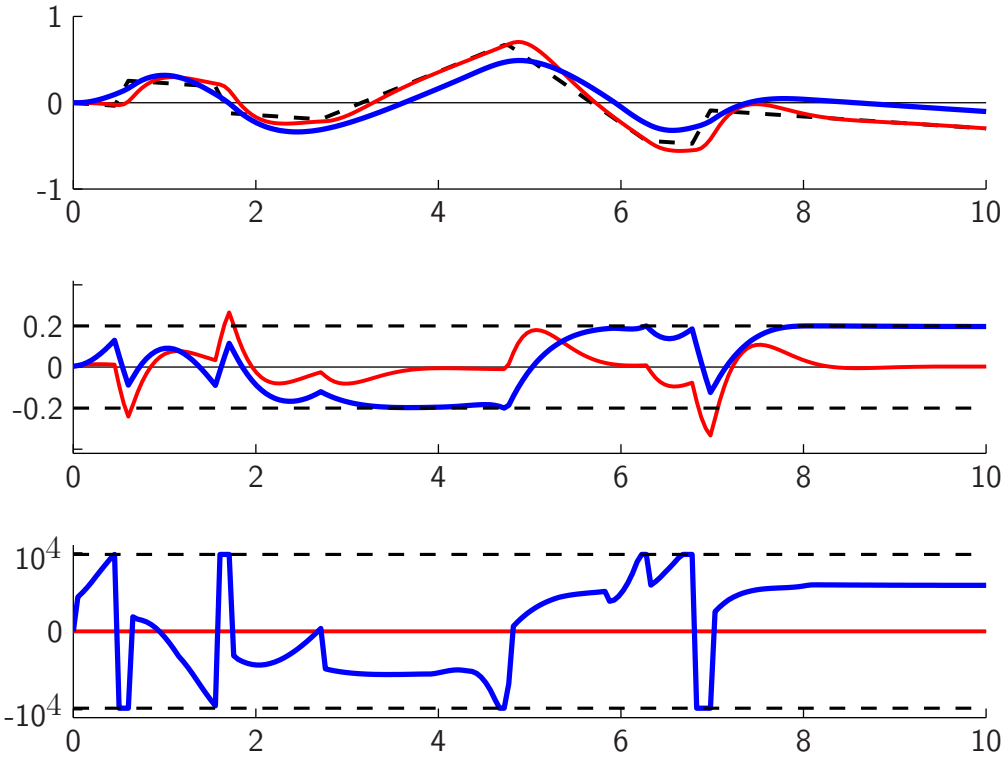
Numerical example

- $M = 2000$ kg, $D = 10{,}000$ Ns/m, $K = 30{,}000$ N/m
- extension limits $E^{\min} = E^{\max} = 0.2$ m
- force limits $F^{\min} = F^{\max} = 10{,}000$ N
- natural frequency 0.6 Hz, damping coefficient 0.65
- 20 Hz sampling, horizon $T = 20$ $\Rightarrow$ 1 second foresight

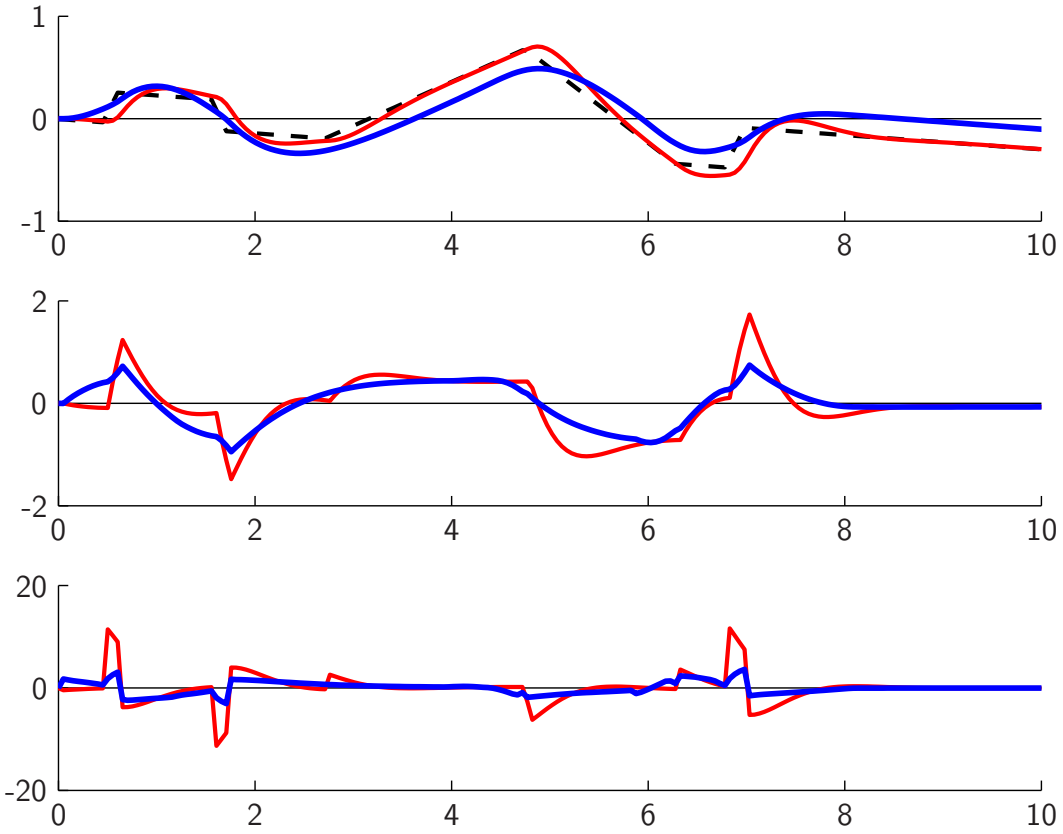
on a desktop (Intel Core 2 Duo, 2.33 GHz)

- CVX + SDPT3: 1.2 s
- cvxmod: 1.4ms

- from top: vehicle position, suspension extension, actuator force
- uncontrolled: thin red; controlled: thick blue
- constraints: dotted; terrain: dashed



- from top: vehicle position, velocity, acceleration
- uncontrolled: thin red; controlled: thick blue



Overview

- convex optimization
- embedded convex optimization
- tools
- cvxmod code generation
- active suspension example
- **a few implementation details**

Templates

- freely mix Python and C by writing templates:

```
sigma = 0;
for (i = 0; i < @pyval{p}@; i++)
  sigma += (work->s[i] + alpha*ds_aff[i])*(work->z[i] + alpha*dz_aff[i]);
sigma /= mu;
sigma = sigma*sigma*sigma;

// Finish calculating mu now.
@pyif{ (p != 0)
  mu *= @pyval{1.0/p}@;
}@
smu = sigma*mu;
```

- can also include Python blocks, with `print` statements

Code style

- writing by hand: use libraries to simplify
- code generation: whatever runs fastest
- with small problems, we found explicit code surprisingly effective

```
v[30] = ((((((((((((((((((((((((((-L[85]*v[0] - L[86]*v[1])
       - L[87]*v[2]) - L[88]*v[3]) - L[89]*v[4]) - L[90]*v[5])
       - L[91]*v[6]) - L[92]*v[7]) - L[93]*v[8]) - L[94]*v[9])
       - L[95]*v[10]) - L[96]*v[11]) - L[97]*v[12]) - L[98]*v[13])
       - L[99]*v[14]) - L[100]*v[15]) - L[101]*v[16]) - L[102]*v[17])
       - L[103]*v[19]) - L[104]*v[20]) - L[105]*v[21])
       - L[106]*v[22]) - L[107]*v[23]) - L[108]*v[24])
       - L[109]*v[26]) - L[110]*v[28]) - L[111]*v[29]);
```

- slow generation and compilation, but very fast runtimes
- does not scale to large problems, may not work so well on small processors

Conclusions

- convex optimization problems come up in many application areas
- limited use in real-time and embedded applications so far
- code generation makes solvers easier to write, faster