

Embedded Convex Optimization with CVXPY

Nicholas Moehle, Jacob Mattingley, Stephen Boyd

Stanford University

November 2017

Outline

Convex optimization

Embedded convex optimization

DSLs for embedded convex optimization

Examples

Optimization

optimization problem:

$$\begin{array}{ll} \text{minimize} & f_0(x; \theta) \\ \text{subject to} & f_i(x; \theta) \leq 0, \quad i = 1, \dots, m \end{array}$$

- ▶ decision variable x (a vector)
- ▶ objective function f_0
- ▶ constraint functions f_i , for $i = 1, \dots, m$
- ▶ parameter(s) θ

the solution x^* minimizes the objective over all vectors satisfying the constraints.

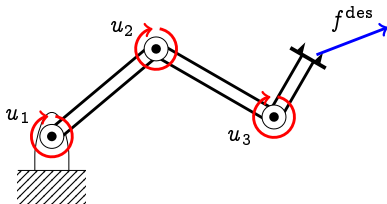
Convex optimization

- ▶ all f_i are convex (have nonnegative curvature)
- ▶ includes least-squares, linear/quadratic programming, ...
- ▶ can obtain global solution quickly and reliably
- ▶ mature software available (open source and commercial)

Domain-specific languages for convex optimization

- ▶ used to specify (and solve) convex problems
- ▶ problem constructed out of:
 - variables
 - constants
 - parameters (value fixed at solve time, may change between solves)
 - functions from a library
- ▶ specified problem mapped to a solver format (e.g., conic form)
- ▶ makes prototyping easier
- ▶ DSLs include CVXPY, CVX, Convex.jl, YALMIP, . . .

Example: actuator allocation



- ▶ $u \in \mathbf{R}^m$ are actuator values
- ▶ generate force $f^{\text{des}} \in \mathbf{R}^n$ according to $f^{\text{des}} = Au$
- ▶ A depends on system configuration (e.g., joint angles)
- ▶ want u small, near previous value u^{prev}
- ▶ actuator limits $u^{\text{min}} \leq u \leq u^{\text{max}}$

Actuator allocation problem

$$\begin{aligned} & \text{minimize} && \|u\|_1 + \lambda \|u - u^{\text{prev}}\|_2^2 \\ & \text{subject to} && Au = f^{\text{des}} \\ & && u^{\text{min}} \leq u \leq u^{\text{max}} \end{aligned}$$

- ▶ variable is u
- ▶ constants are u^{min} , u^{max} , $\lambda > 0$
- ▶ parameters are A , f^{des} , u^{prev}

Actuator allocation in CVXPY

CVXPY code:

```
u = Variable(10)
A = Parameter((6, 10), value=A_val)
f_des = Parameter(6, value=f_val)
u_prev = Parameter(10, value=u_val)

prob = Problem(Minimize(norm(u, 1)
                    + lambda*sum_square(u - u_des),
                    [A*u == f_des, u_min <= u, u <= u_max]))
prob.solve()
```


Under the hood: canonicalization

CVXPY transforms original problem

$$\begin{aligned} & \text{minimize} && \|u\|_1 + \lambda \|u - u^{\text{prev}}\|_2^2 \\ & \text{subject to} && Au = f^{\text{des}} \\ & && u^{\text{min}} \leq u \leq u^{\text{max}} \end{aligned}$$

into equivalent standard-form QP:

$$\begin{aligned} & \text{minimize} && \begin{bmatrix} u \\ t \end{bmatrix}^T \begin{bmatrix} \lambda I & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} u \\ t \end{bmatrix} + \begin{bmatrix} -2\lambda u^{\text{prev}} \\ \mathbf{1} \end{bmatrix}^T \begin{bmatrix} u \\ t \end{bmatrix} \\ & \text{subject to} && \begin{bmatrix} A & 0 \end{bmatrix} \begin{bmatrix} u \\ t \end{bmatrix} = f^{\text{des}} \\ & && \begin{bmatrix} I & 0 \\ -I & 0 \\ I & -I \\ -I & -I \end{bmatrix} \begin{bmatrix} u \\ t \end{bmatrix} \leq \begin{bmatrix} u^{\text{max}} \\ -u^{\text{min}} \\ 0 \\ 0 \end{bmatrix} \end{aligned}$$

with variable $(u, t) \in \mathbf{R}^{2m}$

Outline

Convex optimization

Embedded convex optimization

DSLs for embedded convex optimization

Examples

Embedded convex optimization

- ▶ solve same problem many times, using different parameter values
- ▶ real-time deadlines (milliseconds, microseconds)
- ▶ small software footprint
- ▶ extreme reliability
- ▶ no babysitting

Embedded optimization applications

- ▶ automatic control
 - actuator allocation
 - model predictive control
 - trajectory generation
- ▶ signal processing
 - moving-horizon estimation
- ▶ energy
 - battery management
 - hybrid vehicle control
 - HVAC control
- ▶ finance
 - quantitative trading

Embedded convex optimization solvers

- ▶ a solver maps parameters to solution (for a specific problem family)
- ▶ some (open-source) examples:
 - ECOS (2013)
 - qpOASES (2014)
 - OSQP (2016)
- ▶ typically written in C or C++
- ▶ special attention to memory allocation, division, . . .
- ▶ solve one problem repeatedly with different parameters
 - symbolic step, followed by numerical step (ECOS, OSQP)
 - factorization caching (OSQP)

Outline

Convex optimization

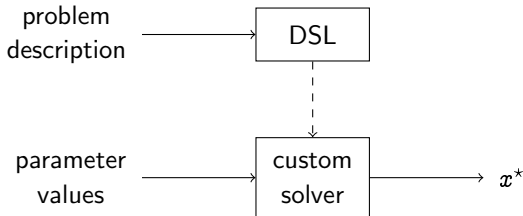
Embedded convex optimization

DSLs for embedded convex optimization

Examples

DSLs for embedded convex optimization

- ▶ parse a parametrized problem, generate a custom solver



- ▶ can re-solve problem with new parameters
- ▶ problem structure fixed (including parameter size)
- ▶ solver code optimized during code generation

DSLs for embedded convex optimization

what can we pre-compute?

- ▶ reduction to standard form (canonicalization)
- ▶ sparsity patterns of problem data
- ▶ efficient permutation for sparse matrix factorization
- ▶ factorization fill-in
- ▶ in some cases, can cache matrix factorizations

for small–medium problems, saved overhead is (very) significant

CVXGEN

- ▶ Mattingley, Boyd (2012)
- ▶ code generation for quadratic programs
- ▶ generates library-free C source
- ▶ built-in backend solver (interior point method)
- ▶ explicit coding style
 - very fast for small problems
 - code size scales poorly past a few thousand scalar parameters
- ▶ used in industry, e.g., SpaceX

CVXPY-codegen

- ▶ Moehle, Boyd
- ▶ Python-based (an extension of CVXPY)
- ▶ generates library-free, embedded C source
- ▶ interchangeable backend solvers:
 - ECOS (interior point)
 - OSQP (ADMM), soon
- ▶ code size / runtime scale gracefully with problem description size
 - (but slower than CVXGEN for very small problems)
- ▶ open source
- ▶ makes Python interface for generated solver

Canonicalization

parametrized problem

$$\begin{aligned} & \text{minimize} && f_0(x; \theta) \\ & \text{subject to} && f_i(x; \theta) \leq 0, \quad i = 1, \dots, m \end{aligned}$$

converted to a QP:

$$\begin{aligned} & \text{minimize} && z^T P(\theta) z + q(\theta)^T z \\ & \text{subject to} && A(\theta) z + b(\theta) \geq 0 \end{aligned}$$

- ▶ z is (augmented) decision variable
- ▶ P , q , A , and b depend on parameters
- ▶ solution x^* recovered from z^*
- ▶ canonicalization step during code generation
- ▶ (conversion to conic problems is similar)

Storing P , q , A , and b in CVXGEN

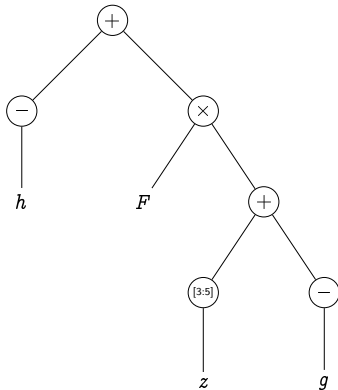
- ▶ $P(\theta)$, $q(\theta)$, $A(\theta)$, and $b(\theta)$ must be updated before each solve
- ▶ CVXGEN uses an explicit style:

```
b[13] = -(G[300]*h[0] + G[301]*h[1] + G[302]*h[2] + G[303]*h[3]
        + G[304]*h[4] + G[305]*h[5] + G[306]*h[6] + G[307]*h[7]
        + G[308]*h[8] + G[309]*h[9] + G[310]*h[10] + G[311]*h[11]
        + G[312]*h[12] + G[313]*h[13] + G[314]*h[14] + G[315]*h[15]
        + G[316]*h[16] + G[317]*h[17] + G[318]*h[18] + G[319]*h[19]
        + G[320]*h[20] + G[321]*h[21] + G[322]*h[22] + G[323]*h[23]
        + G[324]*h[24] + G[325]*h[25] + G[326]*h[26] + G[327]*h[27]
        + G[328]*h[28] + G[329]*h[29] + G[330]*h[30] + G[331]*h[31]
        + G[347]*h[47] + G[348]*h[48] + G[349]*h[49]);
```

- ▶ fast, but limits CVXGEN to small problems

Affine parse tree

- ▶ or, store $A(\theta)z + b(\theta)$ as an affine parse tree:



for $F(z_{[3:5]} - g) - h$

- ▶ $P(\theta)$ and $q(\theta)^T$ represented similarly

Affine parse tree in CVXPY-codegen

- ▶ each subtree is an affine function
 - recursively walk the tree to build $A(\theta)$ and $b(\theta)$
 - at each node, carry out operation directly on subtree coefficients
- ▶ recursion is unwrapped in generated code:

```
neg(param_h, node1_offset);  
index_coeff(var_z, node1_var_z);  
neg(param_g, node2_offset);  
matmul(param_F, node1_var_z, node3_var_z);  
matmul(param_F, node2_offset, node3_offset);  
sum(node1_offset, node3_offset);
```

- ▶ sparsity patterns fixed during code generation
- ▶ better scalability than explicit methods; still fast for small problems

C code structure

structure of generated code:

1. `init()`: initializes backend solver, allocate solver memory (if needed)
2. `solve()`: takes in parameters, solves problem
3. `cleanup()`: frees solver memory

practical usage: `init` once, then `solve` many times in a loop

Outline

Convex optimization

Embedded convex optimization

DSLs for embedded convex optimization

Examples

Actuator allocation: code generation

$$\begin{aligned} & \text{minimize} && \|u\|_1 + \lambda \|u - u^{\text{prev}}\|_2^2 \\ & \text{subject to} && Au = f^{\text{des}} \\ & && u^{\text{min}} \leq u \leq u^{\text{max}} \end{aligned}$$

Python code:

```
u = Variable(10, name='u')
A = Parameter((6, 10), name='A')
f_des = Parameter(6, name='f_des')
u_prev = Parameter(10, name='u_prev')

prob = Problem(Minimize(norm(u, 1)
                    + lambda*sum_square(u - u_des),
                    [A*u == f_des, u_min <= u, u <= u_max]))
codegen(prob, 'target_directory')
```

Actuator allocation: generated code

```
typedef struct params_struct{
    double A[6][10];
    double f_des[6];
    double u_prev[10];
} Params;

typedef struct vars_struct{
    double u[10];
} Vars;

typedef struct work_struct{
    ...
} Work;

void cg_init(Work *work);
int cg_solve(Params *params, Work *work, Vars *vars);
void cg_cleanup(Work *work);
```

Actuator allocation: usage example

usage example:

```
int main(){
    Params params;
    Vars vars;
    Work work;
    cg_init(&work); // Initialize solver.

    while(1){
        update_params(&params); // Get new data, update parameters.
        cg_solve(&params, &work, &vars); // Solve problem.
        implement_vars(&vars); // Implement the solution.
    }
}
```

Actuator allocation: results

CVXGEN:

- ▶ solve time: 60 μ s (1500 \times speedup over CVXPY)
- ▶ memory usage: 11 kB
- ▶ code size: 91 kB

CVXPY-codegen (with ECOS):

- ▶ solve time: 300 μ s
- ▶ memory usage: 23 kB
- ▶ code size: 96 kB

Model predictive control

- ▶ control the linear dynamical system

$$x_{t+1} = Ax_t + Bu_t$$

over T time periods

- ▶ input constraints $\|u_t\|_\infty \leq u^{\max}$
- ▶ problem is:

$$\begin{aligned} & \text{minimize} && \sum_{t=0}^{T-1} \|x_t\|_2^2 + \|u_t\|_2^2 \\ & \text{subject to} && x_{t+1} = Ax_t + Bu_t, \quad t = 0, \dots, T-1 \\ & && \|u_t\|_\infty \leq u^{\max}, \quad t = 0, \dots, T-1 \\ & && x_0 = x_{\text{init}} \\ & && x_T = 0 \end{aligned}$$

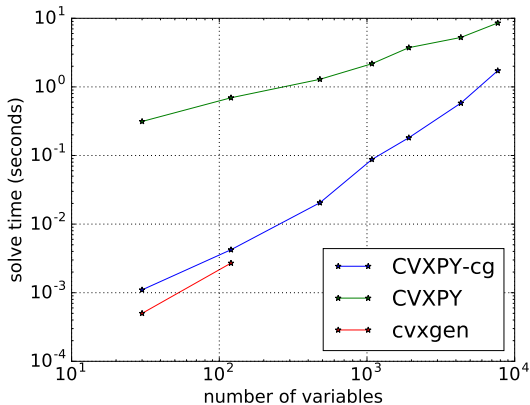
Model predictive control: Python code

```
A = Parameter((n, n), name='A')
B = Parameter((n, m), name='B')
x0 = Parameter(n, name='x0')
u_max = Parameter(name='u_max')
x = Variable((n, T+1), name='x')
u = Variable((m, T), name='u')

obj = 0
constr = [x[:,0] == x0, x[:, -1] == 0]
for t in range(T):
    constr += [x[:, t+1] == A*x[:, t] + B*u[:, t],
               norm(u[:, t], 'inf') <= u_max]
    obj += sum_squares(x[:, t]) + sum_squares(u[:, t])

prob = Problem(Minimize(obj), constr)
codegen(prob, 'target_directory')
```

Model predictive control: solve times



- ▶ inputs, states, and horizon in ratio 1 : 2 : 5
- ▶ backend solver for CVXPY-codegen was ECOS

Other problems

Solve times for other problems (in milliseconds)

problem	CVXGEN	CVXPY-cg	CVXPY
battery1	.303	1.46	509
battery2	1.52	4.50	3112
battery3	—	61.5	55591
portfolio1	0.342	1.099	61.6
portfolio2	1.127	2.684	62.9
portfolio3	—	79.8	152.9
lasso1	0.136	1.40	44.4
lasso2	—	10.51	73.3
lasso3	—	52.37	185.4

Conclusion

- ▶ DSLs make (embedded) convex optimization easy to use
- ▶ convex optimization for real-time applications
- ▶ automatic code generation makes deployment easy