

Code Generation for Embedded Convex Optimization

Jacob Mattingley

Stanford University

October 2010

Convex optimization

- ▶ Problems solvable reliably and efficiently
- ▶ Widely used in scheduling, finance, engineering design
- ▶ Solve every few minutes or seconds

Code generation for embedded convex optimization

Replace 'minutes' with 'milliseconds' and eliminate failure

Agenda

- I. **Introduction** to embedded convex optimization and CVXGEN
- II. **Demonstration** of CVXGEN
- III. **Techniques** for constructing fast, robust solvers
- IV. **Verification** of technical choices
- V. **Final notes** and conclusions

Part I: Introduction

1. Embedded convex optimization
2. Embedded solvers
3. CVXGEN

Embedded convex optimization: **Requirements**

Embedded solvers must have:

- ▶ **Time limit**, sometimes strict, in milliseconds or microseconds
- ▶ **Simple footprint** for portability and verification
- ▶ **No failures**, even with somewhat poor data

Embedded convex optimization: **Exploitable features**

Embedded solvers can exploit:

- ▶ Modest accuracy requirements
- ▶ Fixed dimensions, sparsity, structure
- ▶ Repeated use
- ▶ **Custom design** in pre-solve phase

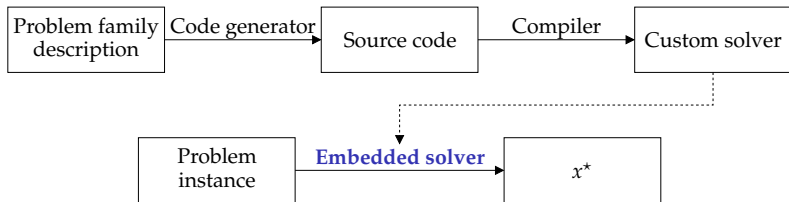
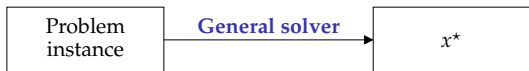
Embedded convex optimization: **Applications**

- ▶ Signal processing, model predictive control
- ▶ Fast simulations, Monte Carlo
- ▶ Low power devices
- ▶ Sequential QP, branch-and-bound

Embedded convex optimization: **Pre-solve phase**



Embedded convex optimization: **Pre-solve phase**



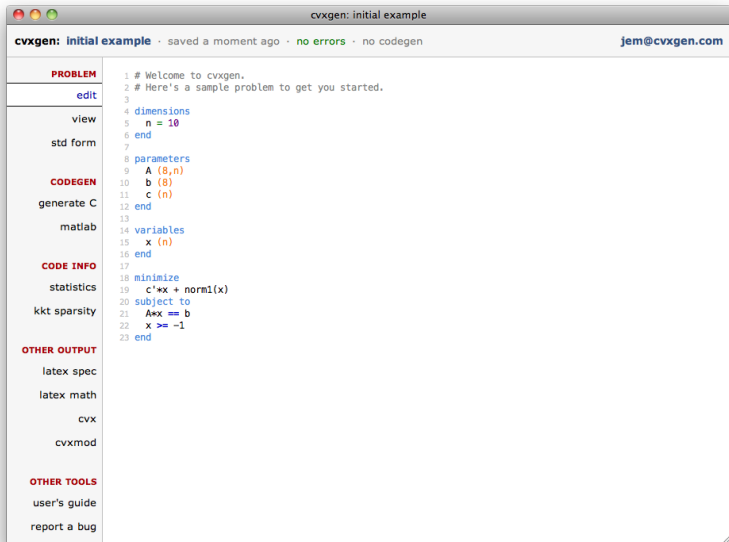
CVXGEN

- ▶ Code generator for embedded convex optimization
- ▶ Mattingley, Boyd
- ▶ Disciplined convex programming input
- ▶ Targets small QPs in flat, library-free C

Part II: **Demonstration**

1. Manipulating optimization problems with CVXGEN
2. Generating and using solvers
3. Important hidden details

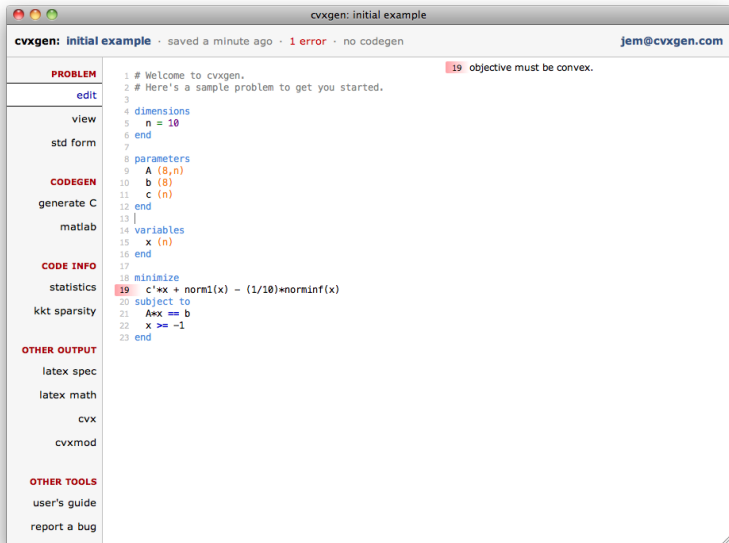
CVXGEN: Problem specification



The screenshot shows a web browser window titled "cvxgen: initial example". The address bar shows "cvxgen: initial example" and the user is logged in as "jem@cvxgen.com". The page content is organized into a sidebar on the left and a main code area on the right. The sidebar contains several sections: "PROBLEM" (with sub-links for "edit", "view", "std form"), "CODEGEN" (with sub-links for "generate C", "matlab"), "CODE INFO" (with sub-links for "statistics", "kkt sparsity"), "OTHER OUTPUT" (with sub-links for "latex spec", "latex math", "cvx", "cvxmod"), and "OTHER TOOLS" (with sub-links for "user's guide", "report a bug"). The main code area displays the following text:

```
1 # Welcome to cvxgen.
2 # Here's a sample problem to get you started.
3
4 dimensions
5   n = 10
6 end
7
8 parameters
9   A (8,n)
10  b (8)
11  c (n)
12 end
13
14 variables
15   x (n)
16 end
17
18 minimize
19   c'*x + norm1(x)
20 subject to
21   A*x == b
22   x >= -1
23 end
```

CVXGEN: Automatic checking



The screenshot shows the CVXGEN web interface for a file named "cvxgen: initial example". The interface includes a sidebar with navigation options and a main content area displaying the problem code. An error message is highlighted in red at the top right of the code area.

cvxgen: initial example · saved a minute ago · 1 error · no codegen jem@cvxgen.com

PROBLEM

- edit
- view
- std form

CODEGEN

- generate C
- matlab

CODE INFO

- statistics
- kkt sparsity

OTHER OUTPUT

- latex spec
- latex math
- cvx
- cvxmod

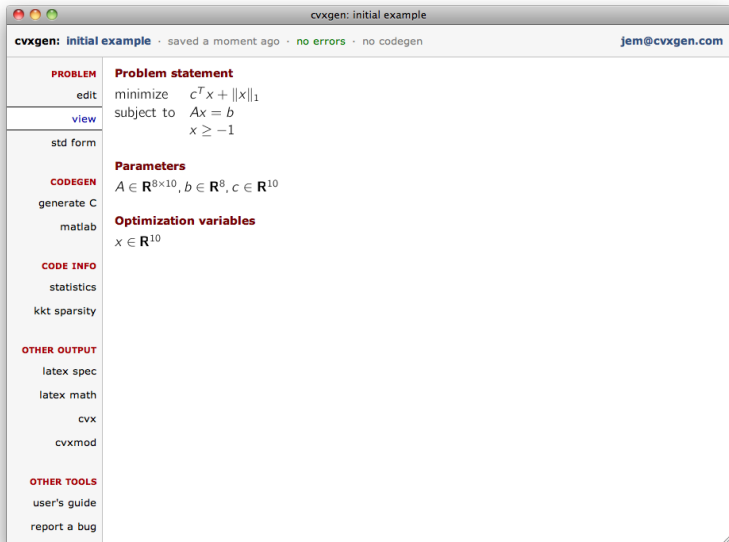
OTHER TOOLS

- user's guide
- report a bug

```
1 # Welcome to cvxgen.
2 # Here's a sample problem to get you started.
3
4 dimensions
5   n = 10
6 end
7
8 parameters
9   A (8,n)
10  b (8)
11  c (n)
12 end
13 |
14 variables
15   x (n)
16 end
17
18 minimize
19   c'*x + norm1(x) - (1/10)*norminf(x)
20 subject to
21   A*x == b
22   x >= -1
23 end
```

19 objective must be convex.

CVXGEN: Formatted problem statement



The screenshot shows a web browser window titled "cvxgen: initial example". The address bar shows "cvxgen: initial example" and the user is logged in as "jem@cvxgen.com". The page content is organized into several sections:

- PROBLEM**
 - edit
 - view
 - std form
- CODEGEN**
 - generate C
 - matlab
- CODE INFO**
 - statistics
 - kkt sparsity
- OTHER OUTPUT**
 - latex spec
 - latex math
 - cvx
 - cvxmod
- OTHER TOOLS**
 - user's guide
 - report a bug

The main content area displays the following problem statement:

Problem statement

$$\begin{aligned} \text{minimize} \quad & c^T x + \|x\|_1 \\ \text{subject to} \quad & Ax = b \\ & x \geq -1 \end{aligned}$$

Parameters

$$A \in \mathbf{R}^{8 \times 10}, b \in \mathbf{R}^8, c \in \mathbf{R}^{10}$$

Optimization variables

$$x \in \mathbf{R}^{10}$$

CVXGEN: Single-button code generation

The screenshot shows a web browser window titled "cvxgen: initial example". The browser's address bar shows "cvxgen: initial example" and the user's email "jem@cvxgen.com". The page content is organized into sections:

- PROBLEM**: Includes links for "edit", "view", and "std form".
- CODEGEN**: Includes a link for "generate C" and a "matlab" link.
- CODE INFO**: Includes links for "statistics" and "kkt sparsity".
- OTHER OUTPUT**: Includes links for "latex spec", "latex math", "cvx", and "cvxmod".
- OTHER TOOLS**: Includes links for "user's guide" and "report a bug".

The main content area displays the following information:

- Problem size**: "Your problem has 306 non-zero KKT matrix entries, which is relatively few. Code generation should be relatively fast. (cvxgen is best for optimization problems with up to around 2000 entries.)"
- Code generation status**: "You have not generated code for this problem."
- A "Generate code" button is located below the status message.

CVXGEN: Completed code generation

cvxgen: initial example

cvxgen: initial example · saved a minute ago · no errors · up-to-date codegen jem@cvxgen.com

PROBLEM **Problem size**

edit Your problem has 306 non-zero KKT matrix entries, which is relatively few. Code generation should be relatively fast.
view (cvxgen is best for optimization problems with up to around 2000 entries.)
std form

CODEGEN You generated code a moment ago. The code matches the problem statement.

generate C

matlab

CODE INFO

statistics

kkt sparsity

OTHER OUTPUT

latex spec

latex math

cvx

cvxmod

OTHER TOOLS

user's guide

report a bug

Generated files

cvxgen.zip	complete	download zip	26 k
cvxgen.tar.gz	complete	download tar	23 k
Makefile	complete	preview · download	1 k
csolve.c	complete	preview · download	6 k
csolve.m	complete	preview · download	1 k
cvxsolve.m	complete	preview · download	1 k
ldl.c	complete	preview · download	89 k
make_csolve.m	complete	preview · download	1 k
matrix_support.c	complete	preview · download	8 k
solver.c	complete	preview · download	8 k
solver.h	complete	preview · download	4 k
testsolver.c	complete	preview · download	5 k
util.c	complete	preview · download	3 k

CVXGEN: Fast, problem-specific code



The screenshot shows a web browser window titled "cvxgen: initial example". The page displays the generated C code for a solver, titled "preview of solver.c". The code is as follows:

```
// Produced by cvxgen, 2010-08-16 11:27:13 -0700.
// cvxgen is Copyright (C) 2006-2010 Jacob Mattingley, jem@cvxgen.com.

// Filename: solver.c.
// Description: Main solver file.

#include "solver.h"

void set_defaults(void) {
    settings.resid_tol = 1e-6;
    settings.eps = 1e-4;
    settings.max_iters = 25;
    settings.refine_steps = 1;

    settings.s_init = 1;
    settings.z_init = 1;
    settings.debug = 0;
    settings.verbose = 1;
    settings.verbose_refinement = 0;

    settings.kkt_reg = 1e-7;
}

double eval_gap(void) {
    int i;
    double gap;

    gap = 0;
    for (i = 0; i < 30; i++)
```

On the right side of the code preview, there is a "close" button and a vertical scrollbar. Below the code, there is a table of links for downloading or previewing various files:

complete	download zip	26 k
complete	download tar	23 k
complete	preview · download	1 k
complete	preview · download	6 k
complete	preview · download	1 k
complete	preview · download	1 k
complete	preview · download	89 k
complete	preview · download	1 k
complete	preview · download	8 k
complete	preview · download	8 k
complete	preview · download	4 k
complete	preview · download	3 k

CVXGEN: Automatic problem transformations

The screenshot shows the CVXGEN web interface for an "initial example". The page is organized into several sections:

- PROBLEM**:
 - KKT matrix**: 88x88; 306 non-zeros. 10 variables, transformed to 20 in the solver.
 - Buttons: edit, view, std form
- CODEGEN**:
 - Buttons: generate C, matlab
- CODE INFO**:
 - Buttons: statistics, kkt sparsity
- OTHER OUTPUT**:
 - Buttons: latex spec, latex math, cvx, cvxmod
- OTHER TOOLS**:
 - Buttons: user's guide, report a bug

The **KKT matrix** section contains a sparsity diagram showing a block-tridiagonal structure with several diagonal blocks and a dense block in the top right corner.

The **Minimization objective** is given by the equation:

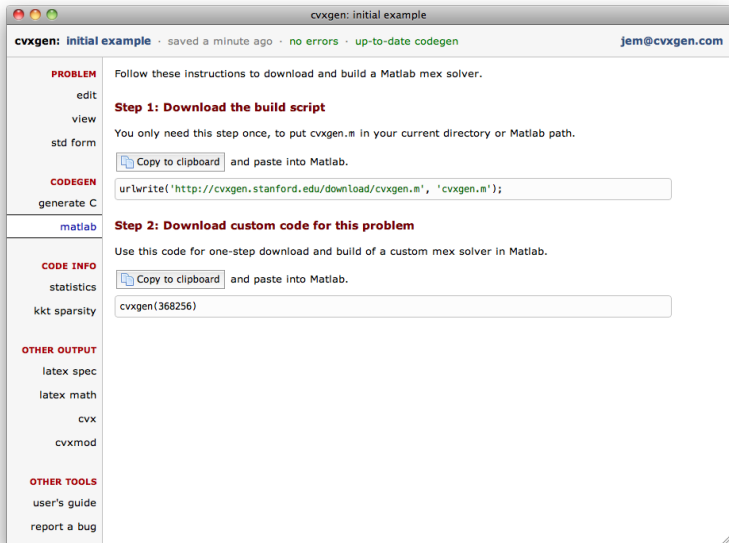
$$\frac{1}{2} \begin{bmatrix} t_{01} \\ x \end{bmatrix}^T \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} t_{01} \\ x \end{bmatrix} + \begin{bmatrix} \mathbf{1} \\ c \end{bmatrix}^T \begin{bmatrix} t_{01} \\ x \end{bmatrix}$$

The **Equality constraint** is given by the equation:

$$\begin{bmatrix} 0 & A \end{bmatrix} \begin{bmatrix} t_{01} \\ x \end{bmatrix} = \begin{bmatrix} b \end{bmatrix}$$

The **OTHER TOOLS** section also includes the text: "8x20; 80 nonzeros. Sparsity diagram follows." and a small sparsity diagram with a white square and a black square.

CVXGEN: Automatically generated Matlab interface



The screenshot shows a web browser window titled "cvxgen: initial example". The page header includes "cvxgen: initial example", "saved a minute ago", "no errors", "up-to-date codegen", and the email "jem@cvxgen.com".


PROBLEM

Follow these instructions to download and build a Matlab mex solver.

edit
view
std form

Step 1: Download the build script

You only need this step once, to put `cvxgen.m` in your current directory or Matlab path.

 Copy to clipboard and paste into Matlab.

CODEGEN


generate C

```
urlwrite('http://cvxgen.stanford.edu/download/cvxgen.m', 'cvxgen.m');
```

matlab

Step 2: Download custom code for this problem

Use this code for one-step download and build of a custom mex solver in Matlab.

 Copy to clipboard and paste into Matlab.

```
cvxgen(368256)
```

CODE INFO

statistics
kkt sparsity

OTHER OUTPUT

latex spec
latex math
cvx
cvxmod

OTHER TOOLS

user's guide
report a bug

Important hidden details

Important details not seen in demonstration:

- ▶ Extremely high speeds
- ▶ Bounded computation time
- ▶ Algorithm robustness

Part III: Techniques

1. Transformation to canonical form
2. Interior-point algorithm
3. Solving the KKT system
 - ▶ Permutation
 - ▶ Regularization
 - ▶ Factorization
 - ▶ Iterative refinement
 - ▶ Eliminating failure
4. Code generation

Transformation to canonical form

- ▶ Problem description uses high-level language
- ▶ Solve problems in canonical form: with variable $x \in \mathbf{R}^n$,

$$\begin{array}{ll} \text{minimize} & (1/2)x^T Qx + q^T x \\ \text{subject to} & Gx \leq h, \quad Ax = b \end{array}$$

- ▶ Transform high-level description to canonical form automatically:
 1. Expand convex functions via epigraphs.
 2. Collect optimization variables into single vector variable.
 3. Shape parameters into coefficient matrices and constants.
 4. Replace certain products with more efficient pre-computations.
- ▶ Generate code for forwards, backwards transformations

Transformation to canonical form: Example

- ▶ Example problem in original form with variables x, y :

$$\begin{aligned} & \text{minimize} && x^T Q x + c^T x + \alpha \|y\|_1 \\ & \text{subject to} && A(x - b) \leq 2y \end{aligned}$$

- ▶ After epigraphical expansion, with new variable t :

$$\begin{aligned} & \text{minimize} && x^T Q x + c^T x + \alpha \mathbf{1}^T t \\ & \text{subject to} && A(x - b) \leq 2y, \quad -t \leq y \leq t \end{aligned}$$

- ▶ After reshaping variables and parameters into standard form:

$$\begin{aligned} & \text{minimize} && \begin{bmatrix} x \\ y \\ t \end{bmatrix}^T \begin{bmatrix} Q & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ t \end{bmatrix} + \begin{bmatrix} c \\ \alpha \mathbf{1} \\ 0 \end{bmatrix}^T \begin{bmatrix} x \\ y \\ t \end{bmatrix} \\ & \text{subject to} && \begin{bmatrix} A & -2I & 0 \\ 0 & -I & -I \\ 0 & I & I \end{bmatrix} \begin{bmatrix} x \\ y \\ t \end{bmatrix} \leq \begin{bmatrix} Ab \\ 0 \\ 0 \end{bmatrix} \end{aligned}$$

Solving the standard-form QP

- ▶ Standard primal-dual interior-point method with Mehrotra correction
- ▶ Reliably solve to high accuracy in 5–25 iterations
- ▶ Mehrotra '89, Wright '97, Vandenberghe '09

Algorithm

Initialize via least-squares. Then, repeat:

1. Stop if the residuals and duality gap are sufficiently small.
2. Compute affine scaling direction by solving

$$\begin{bmatrix} Q & 0 & G^T & A^T \\ 0 & Z & S & 0 \\ G & I & 0 & 0 \\ A & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta x^{\text{aff}} \\ \Delta s^{\text{aff}} \\ \Delta z^{\text{aff}} \\ \Delta y^{\text{aff}} \end{bmatrix} = \begin{bmatrix} -(A^T y + G^T z + P x + q) \\ -S z \\ -(G x + s - h) \\ -(A x - b) \end{bmatrix}.$$

3. Compute centering-plus-corrector direction by solving

$$\begin{bmatrix} Q & 0 & G^T & A^T \\ 0 & Z & S & 0 \\ G & I & 0 & 0 \\ A & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta x^{\text{cc}} \\ \Delta s^{\text{cc}} \\ \Delta z^{\text{cc}} \\ \Delta y^{\text{cc}} \end{bmatrix} = \begin{bmatrix} 0 \\ \sigma \mu \mathbf{1} - \mathbf{diag}(\Delta s^{\text{aff}}) \Delta z^{\text{aff}} \\ 0 \\ 0 \end{bmatrix},$$

with

$$\begin{aligned} \mu &= s^T z / p & \sigma &= ((s + \alpha \Delta s^{\text{aff}})^T (z + \alpha \Delta z^{\text{aff}}) / (s^T z))^3 \\ \alpha &= \sup\{\alpha \in [0, 1] \mid s + \alpha \Delta s^{\text{aff}} \geq 0, z + \alpha \Delta z^{\text{aff}} \geq 0\} \end{aligned}.$$

Algorithm (continued)

4. Combine the updates with

$$\begin{aligned}\Delta x &= \Delta x^{\text{aff}} + \Delta x^{\text{cc}} & \Delta s &= \Delta s^{\text{aff}} + \Delta s^{\text{cc}} \\ \Delta y &= \Delta y^{\text{aff}} + \Delta y^{\text{cc}} & \Delta z &= \Delta z^{\text{aff}} + \Delta z^{\text{cc}} .\end{aligned}$$

5. Find

$$\alpha = \min\{1, 0.99 \sup\{\alpha \geq 0 \mid s + \alpha\Delta s \geq 0, z + \alpha\Delta z \geq 0\}\},$$

and update

$$\begin{aligned}x &:= x + \alpha\Delta x & s &:= s + \alpha\Delta s \\ y &:= y + \alpha\Delta y & z &:= z + \alpha\Delta z .\end{aligned}$$

Solving KKT system

- ▶ Most computation effort, typically 80%, is solution of KKT system
- ▶ Each iteration requires two solves with (symmetrized) KKT matrix

$$K = \left[\begin{array}{cc|cc} Q & 0 & G^T & A^T \\ 0 & S^{-1}Z & I & 0 \\ \hline G & I & 0 & 0 \\ A & 0 & 0 & 0 \end{array} \right]$$

- ▶ Quasisemidefinite: block diagonals PSD, NSD
- ▶ Use permuted LDL^T factorization with diagonal D , unit lower-triangular L

Solving KKT system: **Permutation issues**

- ▶ Factorize $PKP^T = LDL^T$, with permutation matrix P
- ▶ L, D unique, if they exist
- ▶ P determines nonzero count of L , thus computation time
- ▶ Standard method: choose P at solve time
 - ▶ Uses numerical values of K
 - ▶ Maintains stability
 - ▶ Slow (complex data structures, branching)
- ▶ CVXGEN: choose P at development time
 - ▶ Factorization does not even exist, for some P
 - ▶ Even if factorization exists, stability highly dependent on P
 - ▶ **How do we fix this?**

Solving KKT system: **Regularization**

- ▶ Use regularized KKT system \tilde{K} instead
- ▶ Choose regularization constant $\epsilon > 0$, then instead factor:

$$P \left(\left[\begin{array}{cc|cc} Q & 0 & G^T & A^T \\ 0 & S^{-1}Z & I & 0 \\ \hline G & I & 0 & 0 \\ A & 0 & 0 & 0 \end{array} \right] + \left[\begin{array}{c|c} \epsilon I & 0 \\ 0 & -\epsilon I \end{array} \right] \right) P^T = P\tilde{K}P^T = LDL^T$$

- ▶ \tilde{K} now quasidefinite: block diagonals PD, ND
- ▶ Factorization always exists (Gill et al, '96)

Solving KKT system: **Selecting the permutation**

- ▶ Select P at development time to minimize nonzero count of L
- ▶ Simple greedy algorithm:
 - Create an undirected graph from \tilde{K} .
 - While nodes remain, repeat:
 1. For each uneliminated node, calculate the fill-in if it were eliminated next.
 2. Eliminate the node with lowest induced fill-in.
- ▶ Can prove that P determines signs of D_{ii} (will come back to this)

Solving KKT system: **Solution**

- ▶ Algorithm requires two solutions ℓ with different residuals r , of

$$K\ell = r$$

- ▶ Instead, solve

$$\ell = \tilde{K}^{-1}r = P^T L^{-T} D^{-1} L^{-1} P r$$

- ▶ Use cached factorization, forward- and backward-substitution
- ▶ **But: solution to wrong system**
- ▶ Use iterative refinement

Solving KKT system: Iterative refinement

- ▶ Want solution to $K\ell = r$, only have operator $\tilde{K}^{-1} \approx K^{-1}$

- ▶ Use iterative refinement:

$$\text{Solve } \tilde{K}\ell^{(0)} = r.$$

Want correction $\delta\ell$ such that $K(\ell^{(0)} + \delta\ell) = r$. Instead:

1. Compute approximate correction by solving $\tilde{K}\delta\ell^{(0)} = r - K\ell^{(0)}$.
 2. Update iterate $\ell^{(1)} = \ell^{(0)} + \delta\ell^{(0)}$.
 3. Repeat until $\ell^{(k)}$ is sufficiently accurate.
- ▶ Iterative refinement with \tilde{K} provably converges
 - ▶ CVXGEN uses only one refinement step

Solving KKT system: **Eliminating failure**

- ▶ Regularized factorization cannot fail with exact arithmetic
- ▶ Numerical errors can still cause divide-by-zero exceptions
- ▶ Only divisions in algorithm are by D_{ii}
- ▶ Factorization computes $\widehat{D}_{ii} \neq D_{ii}$, due to numerical errors
- ▶ Therefore, given sign ξ_i of D_{ii} , use

$$D_{ii} = \xi_i((\xi_i \widehat{D}_{ii})_+ + \epsilon)$$

- ▶ Makes division 'safe'
- ▶ Iterative refinement still provably converges

Code generation

- ▶ Code generation converts symbolic representation to compilable code
- ▶ Use templates [color key: C code, control code, C substitutions]

```
void kkt_multiply(double *result, double *source) {  
  - kkt.rows.times do |i|  
    result[#{i}] = 0;  
    - kkt.neighbors(i).each do |j|  
      - if kkt.nonzero? i, j  
        result += #{kkt[i,j]}*source[#{j}];  
      }  
}
```

- ▶ Generate extremely explicit code

Code generation: Extremely explicit code

- ▶ Embedded constants, exposed for compiler optimizations:

```
// r3 = -Gx - s + h.  
multbymG(r3, x);  
for (i = 0; i < 36; i++)  
    r3[i] += -s[i] + h[i];
```

- ▶ Computing single entry in factorization:

$$\begin{aligned} L[265] = & (- L[254]*v[118] - L[255]*v[119] - L[256]*v[120] - L[257]*v[121] \\ & - L[258]*v[122] - L[259]*v[123] - L[260]*v[124] - L[261]*v[125] \\ & - L[262]*v[126] - L[263]*v[127] - L[264]*v[128])*d_inv[129]; \end{aligned}$$

- ▶ Parameter stuffing:

```
b[4] = params.A[4]*params.x_0[0] + params.A[9]*params.x_0[1]  
      + params.A[14]*params.x_0[2] + params.A[19]*params.x_0[3]  
      + params.A[24]*params.x_0[4];
```

Part IV: Verification

1. Computation speed
2. Reliability

Computation speeds

- ▶ Maximum execution time more relevant than average
- ▶ Test millions of problem instances to verify performance

Computation speeds: Examples

	<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

Computation speeds: Examples

	<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

Computation speeds: Examples

	<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

Reliability testing

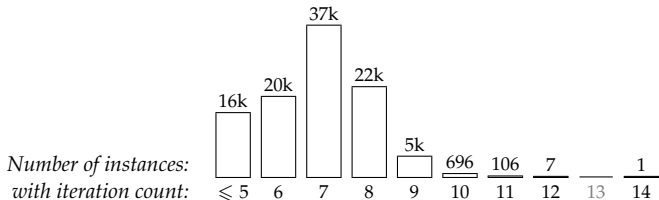
- ▶ Analyzed millions of instances from many problem families
- ▶ Goal: tune algorithm for total reliability, high speed
- ▶ Investigated:
 - ▶ **Algorithms**: primal-barrier, primal-dual, primal-dual with Mehrotra
 - ▶ **Initialization methods** including two-phase, infeasible-start, least-squares
 - ▶ **Regularization** and iterative refinement
 - ▶ **Algebra**: dense, library-based, sparse, flat; all with different solution methods
 - ▶ **Code generation**, using profiling to compare strategies
 - ▶ **Compiler integration**, using profiling and disassembly

Reliability testing: **Example**

- ▶ Computation time proportional to iteration count
- ▶ Thus, simulate many instances and record iteration count
- ▶ Example: ℓ_1 -norm minimization with box constraints

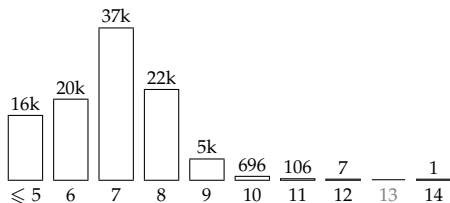
Reliability testing: Example

- ▶ Computation time proportional to iteration count
- ▶ Thus, simulate many instances and record iteration count
- ▶ Example: ℓ_1 -norm minimization with box constraints
- ▶ Iteration count with default settings:

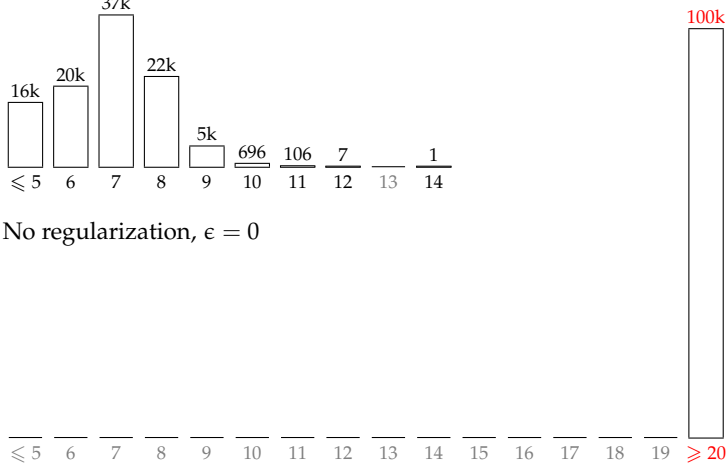


Reliability testing: No KKT regularization

- ▶ Default regularization, $\epsilon = 10^{-7}$

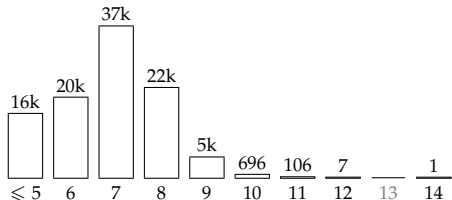


- ▶ No regularization, $\epsilon = 0$

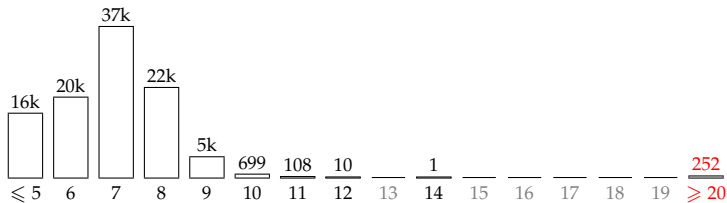


Reliability testing: Decreased KKT regularization

- ▶ Default regularization, $\epsilon = 10^{-7}$

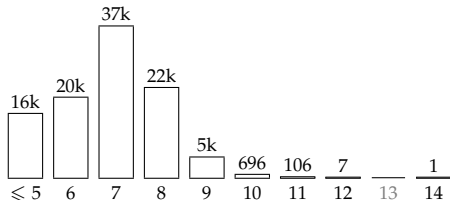


- ▶ Decreased regularization, $\epsilon = 10^{-11}$

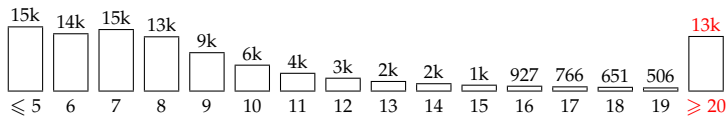


Reliability testing: Increased KKT regularization

- ▶ Default regularization, $\epsilon = 10^{-7}$

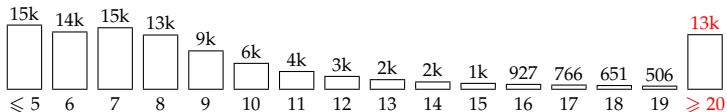


- ▶ Increased regularization, $\epsilon = 10^{-2}$

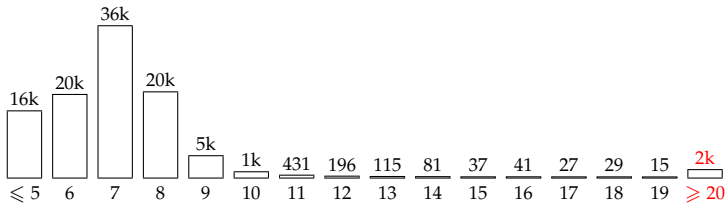


Reliability testing: Iterative refinement

- ▶ Default of 1 iterative refinement step, with $\epsilon = 10^{-2}$



- ▶ Increased to 10 iterative refinement steps, with $\epsilon = 10^{-2}$



Reliability testing: Summary

- ▶ Regularization and iterative refinement allow reliable solvers
- ▶ Iteration count relatively insensitive to parameters

Part V: Final notes

1. Conclusions
2. Contributions
3. Extensions
4. Publications
5. Acknowledgements

Conclusions

Contributions

- ▶ Framework for embedded convex optimization
- ▶ Design and demonstration of reliable algorithms
- ▶ First application of code generation to convex optimization

CVXGEN

- ▶ Fastest solvers ever written
- ▶ Already in use

Extensions

- ▶ Blocking, for larger problems
- ▶ More general convex families
- ▶ Different hardware

Publications

- ▶ CVXGEN: A Code Generator for Embedded Convex Optimization, J. Mattingley and S. Boyd, manuscript
- ▶ Code Generation for Receding Horizon Control, J. Mattingley, Y. Wang and S. Boyd, in review, *IEEE Control Systems Magazine*
- ▶ Code Generation for Receding Horizon Control, J. Mattingley, Y. Wang and S. Boyd, *Proceedings IEEE Multi-Conference on Systems and Control*, pp. 985–992, Yokohama, Japan, September 2010
- ▶ Real-Time Convex Optimization in Signal Processing, J. Mattingley and S. Boyd, *IEEE Signal Processing Magazine*, 27(3):50–61, May 2010
- ▶ Automatic Code Generation for Real-Time Convex Optimization, J. Mattingley and S. Boyd, chapter in *Convex Optimization in Signal Processing and Communications*, Y. Eldar and D. Palomar, Eds., Cambridge University Press, 2009