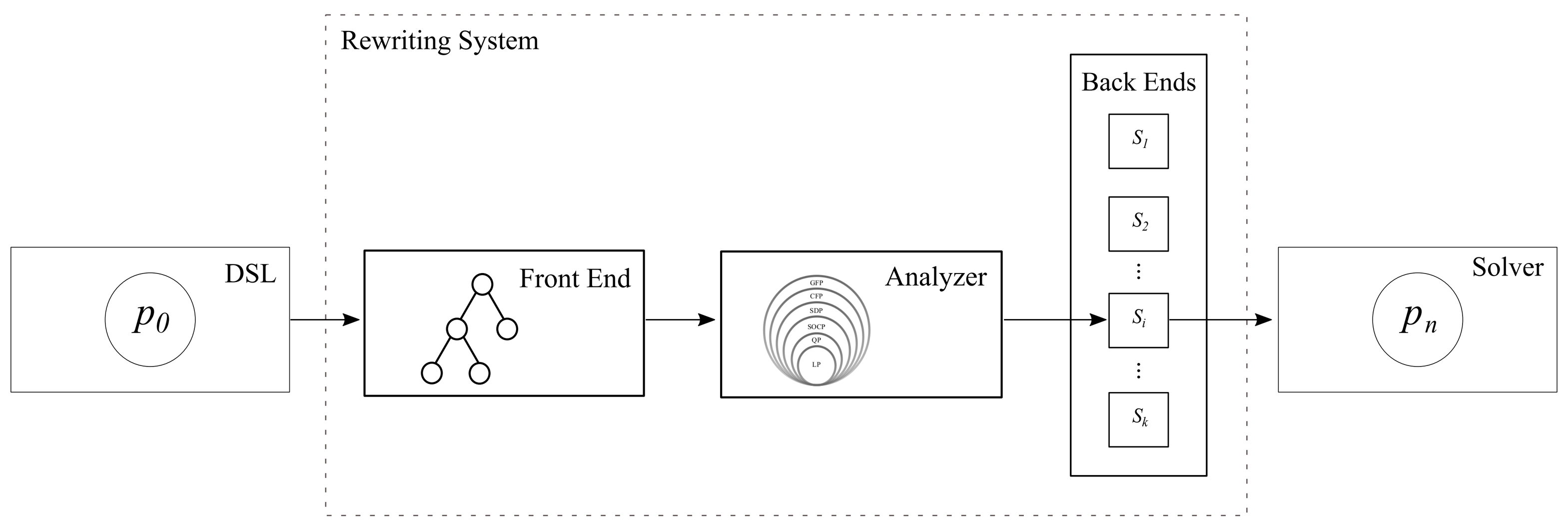
Differentiating through Log-Log Convex Programs

Akshay Agrawal   Stephen Boyd

Stanford University, Electrical Engineering

Domain-specific languages (DSLs) for convex optimization

- DSLs for convex optimization make it easy to specify and solve convex problems
- Modern DSLs (CVXPY, CVXR, Convex.jl, CVX) based on disciplined convex programming (DCP) [7] and disciplined geometric programming (DGP) [2]
- DCP, DGP are libraries of functions (atoms) with known curvature and monotonicity, and composition rules for combining them.



Log-log convex programming

A log-log convex program (LLCP) [2] is an optimization problem

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 1, \quad i = 1, \ldots, m \\ & g_i(x) = 1, \quad i = 1, \ldots, p, \end{array}$$

- $g_i : \mathbf{R}^n_{++} \to \mathbf{R}$ are log-log affine: $G_i(u) = \log g_i(e^u)$ is affine.
- $f_i : \mathbf{R}^n_{++} \to \mathbf{R}$ are log-log convex: $F_i(u) = \log f_i(e^u)$ is convex.

Solving a LLCP reduces to solving a convex optimization problem, so LLCPs can be solved reliably and efficiently. LLCPs generalize the well-known class of geometric programs (GPs), which have applications to [4]:

- chemical engineering
- circuit design
- transformer design
- aircraft design
- mechanical engineering
- communications

LLCPs can be specified and solved using CVXPY (see `cvxpy.org`).

Differentiation

Our recent work [1] lets you get the gradient of the solution of an LLCP with respect to the parameters.

This lets you calculate the sensitivity of the solution with respect to parameters in the objective function and constraints:

- if the constraints were altered slightly, how would the solution change?
- if the objective were altered slightly, how would the solution change?

It also lets you *backpropagate* through LLCPs, letting you use them as tunable layers in end-to-end learning pipelines.

We have implemented the derivative of LLCPs in CVXPY, and in PyTorch and TensorFlow using our extension package CVXPY Layers.

Example

`https://www.cvxpy.org/examples/derivatives/fundamentals.html`

```
import cvxpy as cp

x = cp.Variable(pos=True)
y = cp.Variable(pos=True)
z = cp.Variable(pos=True)

a = cp.Parameter(pos=True)
b = cp.Parameter(pos=True)
c = cp.Parameter()

objective_fn = 1/(x*y*z)
objective = cp.Minimize(objective_fn)
constraints = [a*(x*y + x*z + y*z) <= b, x >= y**c]
problem = cp.Problem(objective, constraints)

print(problem.is_dgp(dpp=True))
```

```
a.value = 2.0
b.value = 1.0
c.value = 0.5
problem.solve(gp=True, requires_grad=True)
```

```
print(x.value)
print(y.value)
print(z.value)
```

```
0.5612147353889386
0.31496200373359456
0.36892055859991446
```

```
a.delta = da
b.delta = db
c.delta = dc
problem.derivative()
```

```
x_hat = x.value + x.delta
y_hat = y.value + y.delta
z_hat = z.value + z.delta

a.value += da
b.value += db
c.value += dc
problem.solve(gp=True)

print('x: predicted {0:.5f} actual {1:.5f}'.format(x_hat, x.value))
print('y: predicted {0:.5f} actual {1:.5f}'.format(y_hat, y.value))
print('z: predicted {0:.5f} actual {1:.5f}'.format(z_hat, z.value))
```

```
x: predicted 0.55729 actual 0.55732
y: predicted 0.31783 actual 0.31781
z: predicted 0.37179 actual 0.37178
```

```
x.gradient = dx
y.gradient = dy
z.gradient = dz
problem.backward()
```

PyTorch and TensorFlow integration

Our software lets you drop LLCPs into PyTorch or TensorFlow with just one line

This lets you tune the parameters in an LLCP using gradient descent.

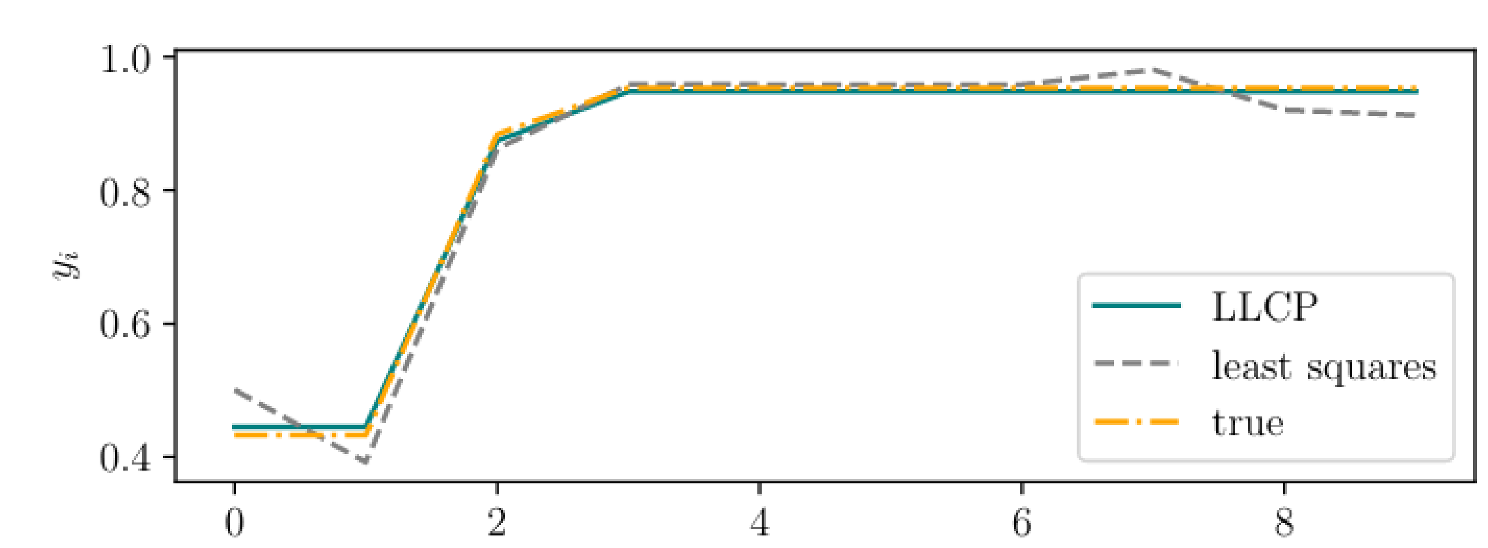
```
from cvxpylayers.torch import CvxpyLayer
import torch

layer = CvxpyLayer(problem, parameters=[a, b, c],
                   variables=[x, y, z], gp=True)
a_tch = torch.tensor(2.0, requires_grad=True)
b_tch = torch.tensor(1.0, requires_grad=True)
c_tch = torch.tensor(0.5, requires_grad=True)

x_star, y_star, z_star = layer(a_tch, b_tch_c_tch)
sum_of_solution = x_star + y_star + z_star
sum_of_solution.backward()
```

Structured prediction

Using CVXPY Layers, we can learn LLCPs for structured prediction, in which the output is known to satisfy constraints (like monotonicity)



`cvxpy.org/examples/derivatives/structured_prediction.html`

References

[1] Akshay Agrawal and Stephen Boyd.
Differentiating through log-log convex programs.
arXiv, 2020.

[2] Akshay Agrawal, Steven Diamond, and Stephen Boyd.
Disciplined geometric programming.
Optimization Letters, 2019.

[3] Akshay Agrawal, Robin Verschueren, Steven Diamond, and Stephen Boyd.
A rewriting system for convex optimization problems.
Journal of Control and Decision, 5(1):42–60, 2018.

[4] Stephen Boyd, Seung-Jean Kim, Lieven Vandenberghe, and Arash Hassibi.
A tutorial on geometric programming.
Optimization and engineering, 8(1):67, 2007.

[5] Edward Burnell and Warren Hoburg.
GPkit software for geometric programming.
`https://github.com/convexengineering/gpkit`, 2018.
Version 0.7.0.

[6] Richard Duffin, Elmor Peterson, and Clarence Zener.
Geometric programming — theory and application, 1967.

[7] Michael Grant, Stephen Boyd, and Yinyu Ye.
Disciplined convex programming.
In *Global optimization*, pages 155–210. Springer, 2006.

[8] Constantin Niculescu.
Convexity according to the geometric mean.
Mathematical Inequalities and Applications, 3(2):155–167, 04 2000.