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

\[
\begin{align*}
\text{minimize} & \quad f_0(x) \\
\text{subject to} & \quad f_i(x) \leq 1, \quad i = 1, \ldots, m \\
& \quad g_i(x) = 1, \quad i = 1, \ldots, p,
\end{align*}
\]

where \( g_i : \mathbb{R}^n \rightarrow \mathbb{R} \) are log-log affine: \( G_i(u) = \log g_i(e^u) \) is affine.

\( f_i : \mathbb{R}^n \rightarrow \mathbb{R} \) 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 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

```python
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 function
objective_fn = 1/(x+y+z)

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

problem = cp.Problem(objective, constraints)

print(problem.is_dcp(dpps=True))
```

**PyTorch and TensorFlow integration**

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

```python
from cvxpylayers.torch import CvxpyLayer

import torch

layer = CvxpyLayer(problem, parameters=(a, b, c), variables=(x, y, z), gp=True)

a_torch = torch.tensor(2.0, requires_grad=True)
b_torch = torch.tensor(1.0, requires_grad=True)
c_torch = torch.tensor(0.5, requires_grad=True)

x_star, y_star, z_star = layer(a_torch, b_torch, c_torch)

pred_solution = x_star + y_star + z_star

eval_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)

**References**