Learning Parametrized Convex Functions

Maximilian Schaller   Alberto Bemporad   Stephen Boyd

IMT Lucca, 5 June 2025

Parametrized convex functions

a **parametrized convex function** (PCF) has the form

$$f : \mathbf{R}^n \times \Theta \to \mathbf{R}^d$$

- first argument $x \in \mathbf{R}^n$ is the **variable**
- second argument $\theta \in \Theta \subseteq \mathbf{R}^p$ is the **parameter**
- $f_i(x, \theta)$ is convex in x for each $\theta \in \Theta$, $i = 1, \ldots, d$
- f is continuous in θ for each x

Disciplined convex programming (DCP)

expression representing a PCF f is DCP if

- it is expressed as an expression tree using atomic functions
- leaves are variables or parameters
- convex composition rule holds at each node

example: $f(x, \theta) = \theta x^2$, $\Theta = \mathbf{R}_+$

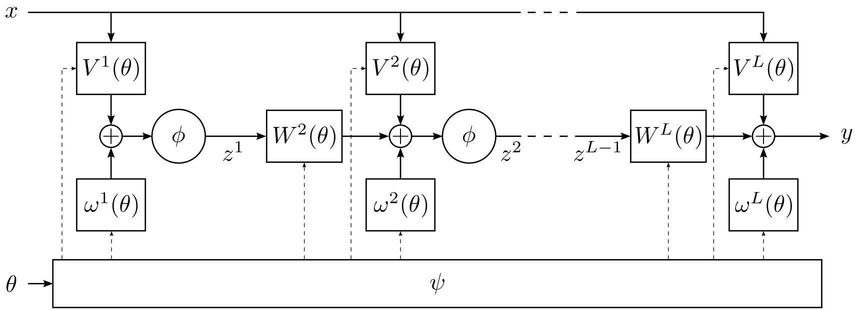
```
import cvxpy as cp

x = cp.Variable()
theta = cp.Parameter(nonneg=True)

f = theta * cp.square(x)
f.is_dcp()
# True
```

Neural PCF architecture

- represent f as neural network with weights V^i, W^i, biases ω^i, and activation ϕ



- weights and activation are such that y is DCP convex in x for every $\theta \in \Theta$
- ψ is a (sub-)network, with weights $w \in \mathbf{R}^q$

Guaranteeing PCF is DCP

we require

- activations ϕ are nondecreasing and convex (*e.g.*, ReLU, logistic)
- weights W^i are nonnegative for all θ (readily enforced in ψ network)

guarantees y is DCP PCF

Fitting a DCP PCF to data

- we are given data

$$(x^k, \theta^k) \in \mathbf{R}^n \times \Theta, \quad y^k \in \mathbf{R}^d, \quad k = 1, \ldots, N$$

- we use loss function $\ell : \mathbf{R}^q \times \mathbf{R}^n \times \Theta \times \mathbf{R}^d \to \mathbf{R}$ and regularizer $r : \mathbf{R}^q \to \mathbf{R}$
- choose w to (approximately) minimize regularized average loss,

$$\frac{1}{N} \sum_{k=1}^{N} \ell(w; x^k, \theta^k, y^k) + \lambda r(w)$$

- $\lambda \geq 0$ is a hyper-parameter, chosen via out-of-sample or cross validation

The LPCF package

- open-source Python package for fitting a PCF to given data
- customizable neural network architecture
- customizable loss, regularization, and learning algorithm
- emits f as
  - a JAX function for fast evaluation
  - a CVXPY expression for use in optimization models

Using the LPCF package

```python
from lpcf.pcf import PCF

# observed data
Y = ...         # shape (N, d)
X = ...         # shape (N, n)
Theta = ...     # shape (N, p)

# fit PCF to data
pcf = PCF()
pcf.fit(Y, X, Theta)

# export PCF to CVXPY
x = cp.Variable((n, 1))
theta = cp.Parameter((p, 1))
pcf_cvxpy = pcf.tocvxpy(x=x, theta=theta)
```

Extensions

- add (convex) quadratic term to the neural network
- require components of f to be monotone in x
- require $h(\theta) \in \partial f(g(\theta), \theta)$ (*i.e.*, that $f(x, \theta) - h(\theta)^T x$ is minimized at $x = g(\theta)$)
- fit a parametrized convex set $C(\theta) = \{x \mid f(x, \theta) \leq 0\}$

Example: Piecewise affine function on $\mathbf{R}$

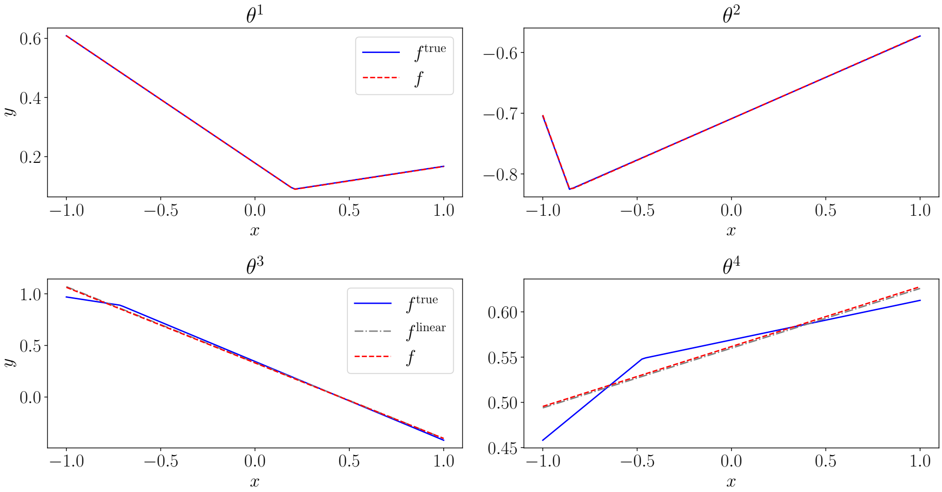
- generate data from piecewise affine function

$$f^{\text{true}}(x,\theta) = s_+ \max\{0, x-m\} + s_- \max\{0, m-x\} + v$$

  with $x \in \mathbf{R}$, $\theta = (s_+, s_-, m, v) \in \mathbf{R}^4$
- f^{true} is a PCF when $s_+ \geq -s_-$ (but we also consider data with $s_+ < -s_-$)

- fit f to 10^5 data points with x sampled from $[-1,1]$ and θ sampled from $[-1,1]^4$

Some PCF fits



Example: Battery aging

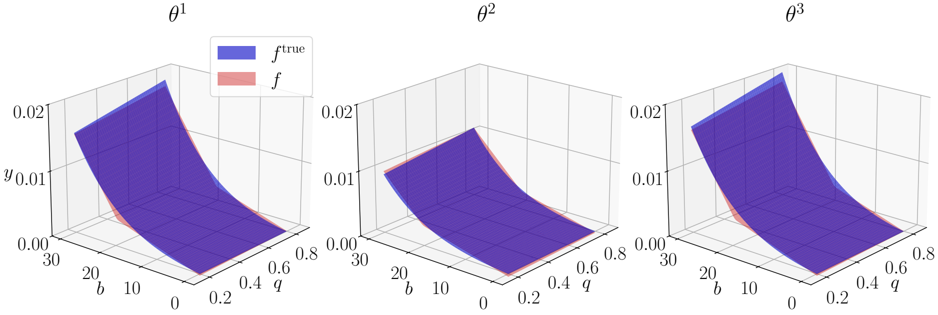
- generate data from a semi-physical model for battery aging rate

$$f^{\text{true}}(x,\theta) = zA^{z-1} b(\alpha q/Q + \beta) \exp\left(\frac{-E_a + \eta b/Q}{R_g(T_0 + T)}\right)$$

- variable is $x = (q, b) \in \mathbf{R}^2_+$, consisting of charge q and (absolute) charge rate b
- parameter is $\theta = (A, Q, T) \in \mathbf{R}^3_+$ where A is accumulated charge throughput, Q is battery capacity, and T is temperature
- other symbols are battery-specific or physical constants

- fit f to 10^5 data points: $q \in [0.2, 0.8]$, $b \in [0, 30]$, $A \in [0, 50]$, $Q = 1$, $T \in [10, 50]$

Some PCF fits



Example: Input-affine control

- consider input-affine dynamical system

$$z_{t+1} = F(z_t, \theta) + G(z_t, \theta)u_t, \quad t = 0, 1, \ldots$$

with state z_t, input u_t, parameter θ

- given initial state z_0, we seek inputs $u_0, u_1, \ldots$ that minimize

$$J(z_0) = \sum_{t=0}^{\infty} H(z_t, u_t, \theta)$$

(can approximate infinite sum with a sum up to a large value $t = T$)

- we can approximately minimize $J(z_0)$ using a local method

Approximate dynamic programming (ADP)

- instead of directly minimizing $J(z_0)$, we use the ADP controller

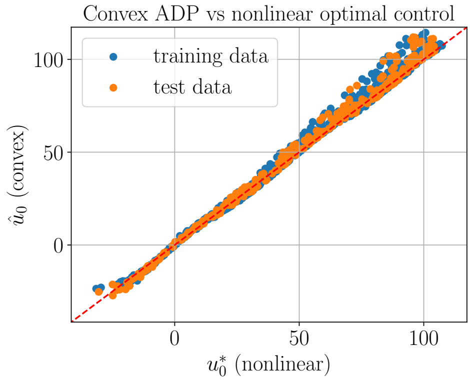
$$u_t = \operatorname*{argmin}_u \left(H(z_t, u, \theta) + \hat{V}(F(z_t, \theta) + G(z_t, \theta)u, \theta) \right), \quad t = 0, 1, \ldots$$

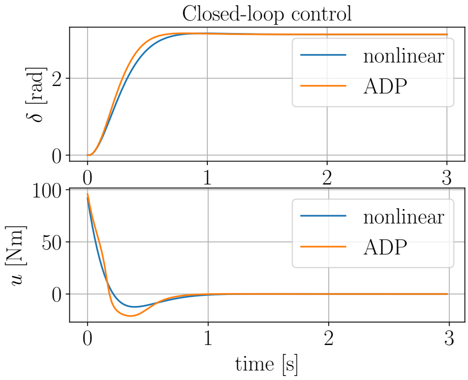
  where $\hat{V}$ is an approximate value function
- with the true value function $V(z, \theta) = \inf_{u_0, u_1, \ldots} J(z)$ this is the optimal input
- we will learn $\hat{V}$ as a PCF
- this means evaluating ADP input u_t is a convex optimization problem
- generate data using local method to approximately evaluate $V(z, \theta)$

Numerical example

- inverted pendulum with angle and angular velocity as state $z \in \mathbf{R}^2$, and parameter mass $\theta = m > 0$
- fit $f = \hat{V}$ to 1000 data points with $z \in [-\pi/6, 7\pi/6] \times [-1, 1]$ and $m \in [0.5, 2]$

PCF fit and result





Conclusions

- we show how to fit a PCF to data
- allows (parametrized) convex optimization to be (in part) ‘data driven’
- bridges a gap between learning from data and structured convex optimization
- `https://github.com/cvxgrp/lpcf`