`cvxfit`: From Data to a Convex Model

EE364b Final Report
Spring 2013-14

Mainak Chowdhury Alon Kipnis Milind Rao

June 4, 2014

Abstract

We have developed a Python package (`cvxfit`) which takes as input values of functions evaluated at points in the function domain and returns a convex fit to the given data. The result is a concise model that describes the data and can be readily used in larger problems which may benefit from the convexity and the simplicity of representation. The package is available at `https://pypi.python.org/pypi/cvxfit/`.

1 Introduction

In many real-life optimization scenarios, we have very limited data about the functions used in the optimization models, especially if the objective or constraint functions are expensive to evaluate, *e.g.*, when function computations require running a complicated experiment. The choices of functions used in those cases are dictated more often by analytical convenience than modeling fidelity. In such situations it may be helpful to come up with *automatic* approximations based on the limited measurements. Such approximations may either be used standalone or may be used to inform future function computations.

1.1 Basic problem

We assume that given a "ground truth" function $f : \mathbf{R}^n \to \mathbf{R}$, the information about f is only given by a set of N possibly noisy measurements $\{x_i, y_i\}_{i=1}^N$ satisfying $x_i \in \mathbf{R}^n, y_i \in \mathbf{R}$ and $y_i = f(x_i) + w_i$ for all i where w_i is independent zero mean noise. A fitting problem that generalizes [MB09] may be formulated as follows:

$$\text{minimize} \quad \sum_{i=1}^{N} L\left(f_l(x_i) - y_i\right) \tag{1}$$

where θ (implicit in f_l) is the optimization variable representing the parameters of f_l, and l specifies a class of convex functions f_l. L is a convex loss function which penalizes mismatch from the given measurements. The function class f_l and the penalty function L are in general supplied by the end user, although heuristics based on cross validation and user specified error criteria can help determine some of them. The problem (1) is non-convex in general. We consider some special forms of this general problem in our project:

- Piecewise linear (PWL): In this case the optimization problem (1) looks like

$$\text{minimize} \quad \sum_{i=1}^{N} L\left(\left(\max_{j\in\{1,\ldots,k\}} a_j^T x_i + b_j\right) - y_i\right) \tag{2}$$

  where $\theta_k = \{a_j, b_j\}_{j=1}^k$ are the optimization variables for model order $k \in \mathbf{Z}_+$ with $a_j \in \mathbf{R}^n$, $b_j \in \mathbf{R}$.

- Piecewise quadratic (PWQ): The corresponding optimization problem looks like

$$\text{minimize} \quad \sum_{i=1}^{N} L\left(\left(\max_{j\in\{1,\ldots,k\}} x_i^T P_j x_i + q_j^T x_i + r_j\right) - y_i\right) \tag{3}$$

  where $\theta_k = \{P_j, q_j, r_j\}_{j=1}^k$ are the optimization variables for model order $k \in \mathbf{Z}_+$ with $P_j \in \mathbf{S}_+^n$, $q_j \in \mathbf{R}^n$, $r_j \in \mathbf{R}$.

2 Implementation

We have implemented the convex function approximation module `cvxfit` in Python using the scientific computation library `scipy`. The current implementation includes a class `CvxFit` which constructs a convex approximation to the given measurements. The package can be downloaded from `https://pypi.python.org/pypi/cvxfit/`.

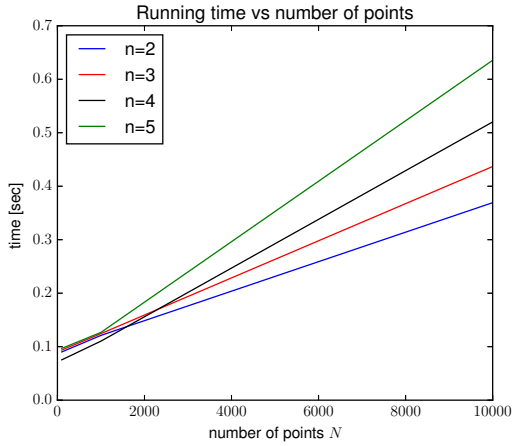
2.1 Code Example

Given a set of points stored as an $N \times n$ `scipy` array `X` and corresponding function values in the array `Y`, an approximation with 10 hyperplanes is obtained by running the following code:

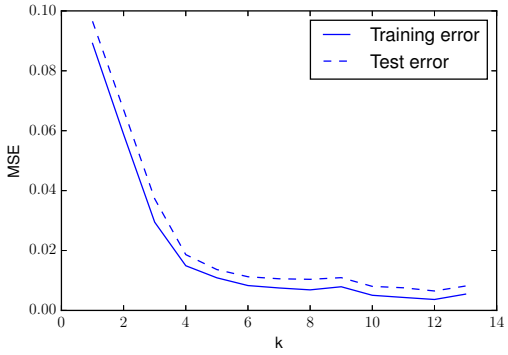
```
from cvxfit import CvxFit

cvxfit_obj = CvxFit(X, Y, k=10, type='pwl')
cvxfit_obj.fit()
```

We can get the value of the fit at a point `x` by calling `cvxfit_obj.evaluate(x)`. The actual coefficients of the fit can be had by calling `cvxfit_obj.get_coefficients()`.

Performance of the algorithm for a specific example is illustrated in Figure 1.



(a) Running time



(b) Root mean square error (RMSE) with increasing fit order k, for $n = 2$.

Figure 1: Approximating the function $f(x) = \|x\|_2$.

2.2 Technical details

The fitting algorithm is similar to a K-means clustering algorithm similar to the one described in [MB09], and involves a sequential convex programming approach. It consists of an initialization step involving clustering points in the domain and two main steps, repeated till convergence:

1. Clustering: Each $(x_i, y_i) \in \mathbf{R}^{n+1}$ is assigned to one of k clusters. This is done by choosing the cluster identity c_i associated with a point (x_i, y_i) to be

$$c_i = c\left((x_i, y_i)\right) = \text{argmax}_{j=1,\ldots,k}\, a_j^T x_i + b_j.$$

2. Fitting: A separate affine function is fit to the data associated with each cluster. This is done by minimizing an objective similar to the one defined in (1). Note that if L is convex, then this step involves solving a convex problem. For the cluster c, in the piecewise linear case, this takes the following form:

$$\text{minimize} \quad \sum_{i:c_i=c} L\left(a_j^T x_i + b_j - y_i\right), \quad i = 1, \ldots, k. \tag{4}$$

   For the piecewise quadratic case, we get the following:

$$\text{minimize} \quad \sum_{i:c_i=c}^{N} L\left(x_i^T P_j x_i + q_j^T x_i + r_j - y_i\right), \quad i = 1 \ldots, k. \tag{5}$$

In this step, we can add a regularization term to the objective while solving for the coefficients $\{a_j, b_j\}$ for the piecewise linear case and to $\{P_j, q_j, r_j\}$ for the piecewise quadratic case. For

the piecewise linear fits, we apply the regularization $\lambda(\|a_j\|^2+b_j^2)$ to the objective function in (4), whereas for the piecewise quadratic fit, we apply the regularization term $\lambda\|P_j\|_F^2$ to the objective function in (5). An initial clustering is obtained by applying K-means^{++} [AV07] in $\mathbf{R}^n$ to the points $\{x_i\}_{i=1}^N$.

3 Examples

In this section we suggest a few use cases of convex fits to data.

Example 1. *Model order reduction:* One distinctive feature of our fitting software, is that the user can specify the order of the model, *e.g.*, the number of hyperplanes k used in the approximation. Given a convex function $f(x) : \mathbf{R}^n \to \mathbf{R}$ specified by a large number of hyperplanes, a simple model reduction can be obtained as follows:

(i) Evaluate the function $f(x)$ at sampling points $\{x_i\}_{i=1}^N$.

(ii) Use `cvxfit` with $\{x_i, f(x_i)\}_{i=1}^N$ and the desired new model order k as an input.

This can be used to obtain an order k approximation to the convex hull of a set of points on the graph of a convex function. An assymetric loss function can be used in order to preserve the overfit of the convex hull with respect to the underlying function. This is illustrated in Figure 2.

Example 2. *Alternative to DCP:* Assume that one of the constraints in a convex optimization problem is known to be convex, but writing it according to a given DCP rule set can be complicated or requires reformulation of the problem. If one can accept an approximate solution, she can replace this constraint with an approximation obtained using `cvxfit`. This can also reduce the computation and memory load in solving the original problem in some scenarios.

Example 3. *Approximate dynamic programming:* Consider the following problem:

Problem: Find optimal initial control action $u_0^*(x)$ for an infinite horizon control problem with the (convex) value function

$$\begin{aligned} V_0(x) = \text{minimize} & \sum_{t=0}^{\infty} \|x_t\|_1 + \sum_{t=0}^{\infty} \|u_t\|_1 \\ \text{subject to } & x_{t+1} = Ax_t + Bu_t, x_0 = x, \end{aligned} \tag{6}$$

where $u_t \in \mathbf{R}, x_t \in \mathbf{R}^3$, for $0 \le t < \infty$, are the optimization variables.

Approach: Either solve full problem in (6) or using an approximation $\tilde{V}_0$ to V_0, solve

$$\tilde{u}_0^*(x) = \underset{u}{\text{argmin}}(\|u\|_1 + \|x\|_1 + \tilde{V}_0(Ax + Bu)). \tag{7}$$

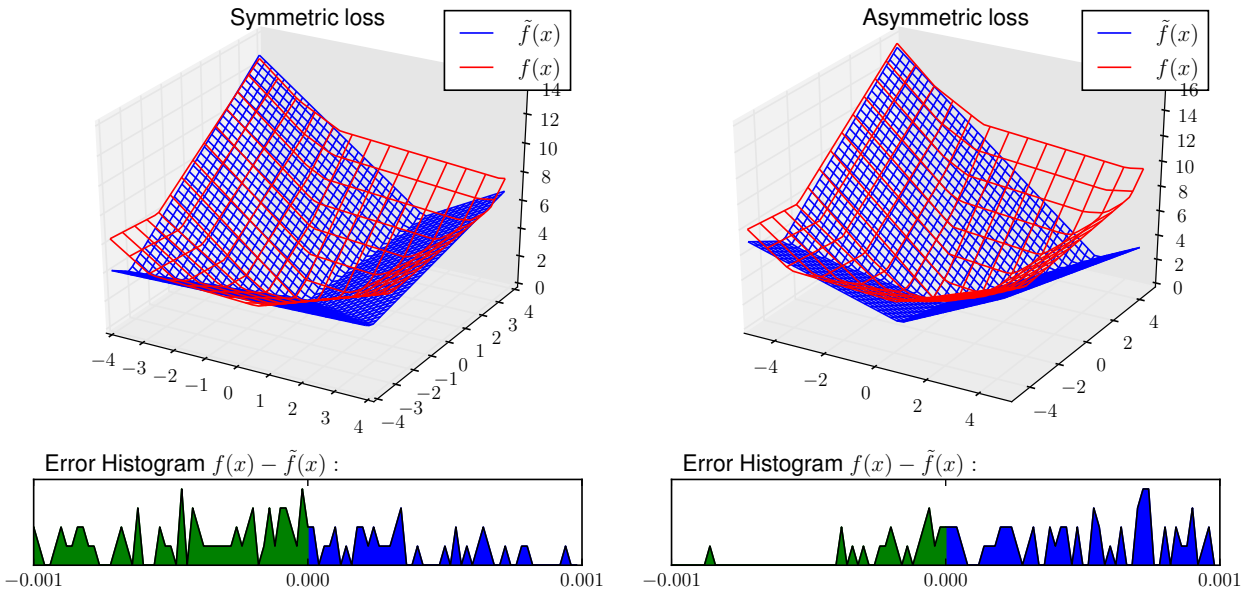


Figure 2: The underlying function $f(x)$ is piecewise linear of order 15. $\tilde{f}(x)$ is a piecewise linear approximation of order $k = 3$, obtained using ℓ_1 loss (left) and 'asymmetric ℓ_1' loss (right).

In real-time applications where solving (6) is too slow, a speed-up can be obtained as follows: First compute $\tilde{V}_0$ offline, which is done by solving (6) for a sample of N points from the state space. The optimal policy can then be computed at each time instance t by solving (7). We plot the optimal policies (both exact and approximate) in Figure 3 for various values of the state given $A = \begin{pmatrix} 1 & 0.1 & 0 \\ 0 & 1 & 0.1 \\ 0 & 0 & 0 \end{pmatrix}$ and $B = \begin{pmatrix} 0 \\ 0 \\ 2 \end{pmatrix}$. We see that the approximate policy is quite similar to the exact one.

4 Concluding thoughts and future work

In this section, we outline several features we are working on, in addition to testing our code on datasets.

- **Dimensionality:** Shown examples are for low dimensions; the general question of how convexity helps in higher dimensions remains open.
- **Parameter selection:** Although we choose defaults based on cross validation with sample problems, in the end the user has to use his domain specific knowledge to choose the type of fit or the parameters involved.

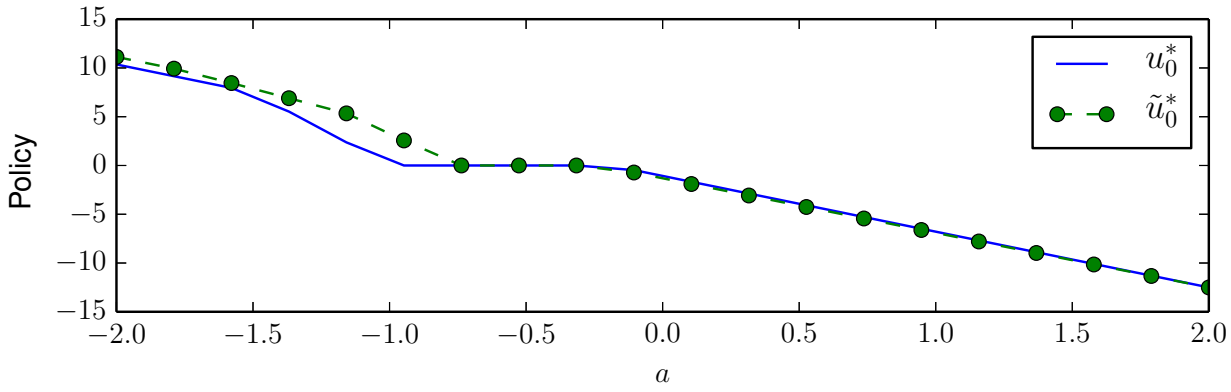


Figure 3: Optimal time zero policy u_0^* obtained by solving (6) and the approximated policy $\tilde{u}_0^*$ obtained by solving (7). The values of the initial state vector is $x = (0.3, a, 0)$, where $a \in \mathbf{R}$ varies between -2 and 2.

- **Regularization:** We use regularization to improve the fitting and to impose priors about the unknown coefficients (*e.g.*, if there are insufficient points in a cluster). The regularization parameter was chosen based on sample problems.
- **Loss functions:** We experimented with different loss functions L like square, huber, ℓ_1 norm. While we did not see much qualitative variations in the quality of fit, specific choices of loss functions can be critical for noisy data in general.
- **General convex models:** We tried preliminary fits with some different models, *e.g.*, second order cones (SOC). The quality of fit in general depends on the "closeness" of f to the chosen model.

References

[AV07] David Arthur and Sergei Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[MB09] Alessandro Magnani and Stephen P Boyd. Convex piecewise-linear fitting. *Optimization and Engineering*, 10(1):1–17, 2009.