Journal of Data Science 0 (0), 1–11
??? 2025
DOI: 10.6339/24-JDS1162
Computing in Data Science

ReLU-ReHU Representations of Piecewise Linear-Quadratic Losses

Tingxian Gao[1], Ben Dai[1,*], and Yixuan Qiu[2]

[1]*Department of Statistics, The Chinese University of Hong Kong, China*
[2]*School of Statistics and Management, Shanghai University of Finance and Economics, China*

Abstract

Piecewise linear-quadratic (PLQ) functions are a fundamental function class in convex optimization, especially within the Empirical Risk Minimization (ERM) framework, which employs various PLQ loss functions. This paper provides a workflow for decomposing a general convex PLQ loss into its ReLU-ReHU representation, along with a Python implementation designed to enhance the efficiency of presenting and solving ERM problems, particularly when integrated with ReHLine (a powerful solver for PLQ ERMs). Our proposed package, `plqcom`, accepts three representations of PLQ functions and offers user-friendly APIs for verifying their convexity and continuity. The Python package is available at https://github.com/keepwith/PLQComposite.

Keywords *empirical risk minimization; optimization; piecewise linear-quadratic function; Python package; ReHLine*

1 Introduction

A piecewise linear-quadratic (PLQ) function is a continuous function defined on the union of finitely many polyhedral sets, where it assumes a linear or quadratic form on each set (Rockafellar, 1988). PLQ functions have extensive applications across various fields, including economics (Garcia-Rubio et al., 2014), finance (Jensen and King, 1992), transportation (Benine-Neto et al., 2011), and optimization (Gardiner and Lucet, 2010). A key application is that many loss functions in various machine learning (ML) ERM problems (Vapnik, 2006) can be expressed as PLQ functions. Specifically, the formulation given a PLQ loss function $L_i(\cdot) : \mathbb{R} \to \mathbb{R}_0^+$ is as follows:

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^d} \sum_{i=1}^{n} L_i\left(\mathbf{x}_i^\intercal \boldsymbol{\beta}\right) + \frac{1}{2}\|\boldsymbol{\beta}\|_2^2, \tag{1}$$

where $\mathbf{x}_i \in \mathbb{R}^d$ is the feature vector for the i-th observation, and $\boldsymbol{\beta} \in \mathbb{R}^d$ is an unknown coefficient vector. Typical PLQ losses include the absolute loss used in average absolute deviation (Portnoy and Koenker, 1997), the hinge loss used in SVMs (Vapnik, 1998), the Huber loss used in the Huber regression (Huber, 1964); see more examples in Table 3 of Dai and Qiu (2024).

Our objective is to transform the form of the PLQ loss function $L_i(\cdot)$ in (1) into the sum of a finite number of rectified linear units (ReLU) (Fukushima, 1969) and rectified Huber units

*Corresponding author. Email: bendai@cuhk.edu.hk.

© 2025 The Author(s). Published by the School of Statistics and the Center for Applied Statistics, Renmin University of China. Open access article under the CC BY license.
Received September 27, 2024; Accepted December 25, 2024

(ReHU) (Dai and Qiu, 2024) as follows.

$$L_i(z) = \sum_{l=1}^{L} \mathrm{ReLU}(u_{li} z + v_{li}) + \sum_{h=1}^{H} \mathrm{ReHU}_{\tau_{hi}}(s_{hi} z + t_{hi}), \tag{2}$$

where u_{li}, v_{li} and s_{hi}, t_{hi}, τ_{hi} are the ReLU-ReHU loss parameters for $L_i(\cdot)$, and the ReLU and ReHU functions are defined as

$$\mathrm{ReLU}(z) = \max(z, 0),$$
$$\mathrm{ReHU}_\tau(z) = \begin{cases} 0, & \text{if } z \leqslant 0, \\ z^2/2, & \text{if } 0 < z \leqslant \tau, \\ \tau(z - \tau/2), & \text{if } z > \tau. \end{cases}$$

The benefits of the transformation are fourfold: (i) as indicated in Theorem 1 of Dai and Qiu (2024), there exists an equivalence between these two forms, ensuring that the ReLU-ReHU representation preserves the properties of PLQ functions; (ii) compared to other forms of PLQ, as discussed in Section 2, the ReLU-ReHU representation is more standardized, making it more programming-friendly; (iii) there is existing literature discussing the properties of ReLU and ReHU (Hein et al., 2019; Bandler et al., 1993), and the characteristics of the representation facilitate the analysis of PLQ functions; (iv) most importantly, the ReLU-ReHU representation allows for the direct use of ReHLine (Dai and Qiu, 2024), a powerful solver for addressing ReLU-ReHU form PLQ ERM in (1).

In this paper, we propose `plqcom` to transform the PLQ loss to its ReLU-ReHU representation with implementation in Python. Our package `plqcom` has three main contributions:

1. Support for three input formats of general PLQ loss functions, along with property checks for convexity and continuity, to ensure the validity of the input functions.
2. An efficient algorithm to automatically decompose a general PLQ loss function into its ReLU-ReHU representation, allowing users to seamlessly utilize the ReHLine solver.
3. An affine transformation capability for the ReLU-ReHU representation, enabling the connection of all data points and facilitating the solution of the ERM problem.

The remainder of this paper is organized as follows. Section 2 presents the algorithms underpinning the package, including the transformation between different input forms of the PLQ function, as well as checks for convexity and continuity of PLQ functions. It also details the conversion of a convex PLQ function to its ReLU-ReHU loss representation. Section 3 offers a comprehensive explanation of the implementation of these algorithms and the structure of the Python software package. Section 4 illustrates the usage of the package through various examples. Finally, Section 5 summarizes the package's capabilities, discusses its limitations, and outlines directions for future work.

2 Methodology

This section explores the essential steps of the algorithms, offering a detailed explanation of the key components.

2.1 Representation of PLQ Functions

We consider three distinct representations of the PLQ functions, which are enumerated as follows.

`plq`: specifying the coefficients of each piece with cutoffs (3).

$$L(z) = \begin{cases} a_1 z^2 + b_1 z + c_1, & \text{if } z \leqslant d_1, \\ \quad \vdots & \\ a_j z^2 + b_j z + c_j, & \text{if } d_{j-1} < z \leqslant d_j, \ \ j = 2, \ldots, m-1, \\ \quad \vdots & \\ a_m z^2 + b_m z + c_m, & \text{if } z > d_{m-1}. \end{cases} \tag{3}$$

The `plq` representation is the most widely-used representation of a PLQ function; therefore, we adopt this representation as the input for the decomposition process.

`max`: specifying the coefficients of a series of quadratic functions and taking the pointwise maximum of each function (4).

$$L(z) = \max_{j=1,2,\ldots m} \left(a_j z^2 + b_j z + c_j\right). \tag{4}$$

The `max` representation only requires the coefficients of each linear or quadratic function, and the pointwise maximum operation is convex preserving. To facilitate this representation, we propose Algorithm 1 as a straightforward approach to convert the `max` form to the `plq` form (3) before the decomposition step.

`points`: constructing piecewise linear functions based on a series of given points (5).

$$L(z) = \begin{cases} q_1 + \frac{q_2 - q_1}{p_2 - p_1}(z - p_1), & \text{if } z \leqslant p_1, \\ \quad \vdots & \\ q_{j-1} + \frac{q_j - q_{j-1}}{p_j - p_{j-1}}(z - p_{j-1}), & \text{if } p_{j-1} < z \leqslant p_j, \ \ j = 2, \ldots, m, \\ \quad \vdots & \\ q_m + \frac{q_m - q_{m-1}}{p_m - p_{m-1}}(z - p_m), & \text{if } z > p_m, \end{cases} \tag{5}$$

where $\{(p_1, q_1), (p_2, q_2), \ldots, (p_m, q_m)\}$ are a series of given points and $m \geqslant 2$.

The `points` representation can only express piecewise linear functions. Notably, the `points` representation can also be viewed as a special case of the `plq` form, where $a_j = 0$, $b_j = \frac{q_j - q_{j-1}}{p_j - p_{j-1}}$, $c_j = q_{j-1} - p_{j-1}\frac{q_j - q_{j-1}}{p_j - p_{j-1}}$, $d_j = p_j$.

2.2 Decompose to ReLU-ReHU Representation

In the previous section, we introduce three representations of PLQ functions and provide algorithms to convert the `max` and `point` representations to the `plq` representation. In this section, the primary objective is to transform a convex PLQ function $L(z)$ with the `plq` representation (3) into its ReLU-ReHU representation (2), as outlined in Algorithm 2. The following lemma demonstrates that this transformation is valid as long as the PLQ function $L(z)$ is convex.

Lemma 1. *If a univariate Piecewise Linear Quadratic (PLQ) function $L(z)$ is convex, it can be decomposed into a ReLU-ReHU representation.*

Algorithm 1: Convert `max` representation to `plq` representation.

Input: The coefficients of a series of functions $a_j,\ b_j,\ c_j,\ j = 1, 2, \ldots, m$

// Find all intersection points as knots

for $j = 1, 2, \ldots, m-1$ **do**
  for $k = j+1,\ j+2,\ \ldots,\ m$ **do**
    Solve $a_j w^2 + b_j w + c_j = a_k w^2 + b_k w + c_k$

// Find the largest piece between adjacent knots

Sort and drop duplicate solutions for w and get $\widetilde{m}$ solutions $w_1 < w_2 < \cdots < w_{\widetilde{m}}$

$\widetilde{d}_1 \leftarrow w_1,\ \widetilde{d}_2 \leftarrow w_2,\ \ldots \widetilde{d}_m \leftarrow w_{\widetilde{m}}$

$\widetilde{d}_0 \leftarrow \widetilde{d}_1 - 1,\ \widetilde{d}_{\widetilde{m}+1} \leftarrow \widetilde{d}_{\widetilde{m}} + 1$

for $j = 1, \ldots, \widetilde{m}+1$ **do**
  $\eta \leftarrow \frac{\widetilde{d}_{j-1} + \widetilde{d}_j}{2},\ y_{\max} \leftarrow a_1\eta^2 + b_1\eta + c_1$
  $\widetilde{a}_j \leftarrow a_1,\ \widetilde{b}_j \leftarrow b_1,\ \widetilde{c}_j \leftarrow c_1$
  for $k = 2, 3, \ldots, m$ **do**
    if $a_k\eta^2 + b_k\eta + c_k > y_{\max}$ **then**
      $\widetilde{a}_j \leftarrow a_k,\ \widetilde{b}_j \leftarrow b_k,\ \widetilde{c}_j \leftarrow c_k,\ y_{\max} \leftarrow a_k\eta^2 + b_k\eta + c_k$

// Merge adjacent pieces if they are the same and reindex

Drop $\widetilde{a}_j,\ \widetilde{b}_j,\ \widetilde{c}_j,\ \widetilde{d}_j$ if $\widetilde{a}_j = \widetilde{a}_{j+1},\ \widetilde{b}_j = \widetilde{b}_{j+1},\ \widetilde{c}_j = \widetilde{c}_{j+1}$ for all j from 1 to $\widetilde{m}$, get $\widetilde{m}'$ terms left and reindex

Output: The coefficients of each piece $\widetilde{a}_j,\ \widetilde{b}_j,\ \widetilde{c}_j,\ j = 1, 2, \ldots, \widetilde{m}'$
The cutoffs $\widetilde{d}_j,\ j = 1, 2, \ldots, \widetilde{m}' - 1$

The main idea of Algorithm 2 is as follows. First, we examine the continuity and convexity of the PLQ function. Next, we identify the minimum knot and separate the function into left and right segments. For each segment, we subtract the tangent line at each knot to obtain the ReLU-ReHU representation for each piece. Finally, we remove any ReLU and ReHU segments with zero coefficients, resulting in the ReLU-ReHU representation of the PLQ function.

Through Algorithm 2, we can at least perform ReLU-ReHU decomposition for each sample-specific loss $L_i(\cdot)$ in (1), thereby obtaining the corresponding ReLU-ReHU parameters. In practice, a sample-specific loss function $L_i(\cdot)$ can often be derived from a *prototype loss* $L(\cdot)$ through affine transformation, which further simplifies the process of ReLU-ReHU decomposition, see Section 2.3.

2.3 Affine Casting

Note that, in practice, $L_i(\cdot)$ in (1) can typically be obtained through affine transformation of a single *prototype loss* $L(\cdot)$, that is,

$$L_i(z) = C_i L(p_i z + q_i),$$

where $C_i > 0$ is the sample weight for the i-th instance, and p_i and q_i are constants. For example,

- for classification problems:
$$L_i\left(\mathbf{x}_i^\intercal \boldsymbol{\beta}\right) = C_i L\left(y_i \mathbf{x}_i^\intercal \boldsymbol{\beta}\right);$$
- for regression problems:
$$L_i\left(\mathbf{x}_i^\intercal \boldsymbol{\beta}\right) = C_i L\left(y_i - \mathbf{x}_i^\intercal \boldsymbol{\beta}\right).$$

Algorithm 2: Decompose a convex PLQ function to its ReLU-ReHU representation.

Input: The coefficients of each piece $a_j,\ b_j,\ c_j,\ j = 1, 2, \ldots, m$
The cutoffs $d_j,\ j = 1, 2, \ldots, m-1$

// Check the continuity
for $j = 1, 2, \ldots, m-1$ **do**
  if $a_j d_j^2 + b_j d_j + c_j \neq a_{j+1} d_j^2 + b_{j+1} d_j + c_{j+1}$ **then**
    The function is not continuous; Break
// Check the convexity
for $j = 1, 2, \ldots, m-1$ **do**
  if $2a_j d_j + b_j > 2a_{j+1} d_j + b_{j+1}$ *or* $a_j < 0$ *or* $a_{j+1} < 0$ **then**
    The function is not convex; Break
// Find the minimum knot and decompose to ReLU-ReHU
Find the minimum knot $d_{j^*} \in \{d_1, \ldots d_{n-1}\}$ with $L(d_{j^*}) = \min\{L(d_1), \ldots L(d_{m-1})\}$
$d_0 \leftarrow -\infty,\ d_m \leftarrow \infty$
for $h = j^* + 1, j^* + 2, \ldots, m$ **do**
  $\tau_h \leftarrow \sqrt{2a_h}(d_h - d_{h-1}),\ s_h \leftarrow \sqrt{2a_h},\ t_h \leftarrow -d_{h-1} s_h$
  $u_h \leftarrow 2d_{h-1}(a_h - a_{h-1}) + (b_h - b_{h-1}),\ v_h \leftarrow -d_{h-1} u_h$
for $h = 0, 1, \ldots, j^* - 1$ **do**
  $\tau_h \leftarrow \sqrt{2a_h}(d_{h+1} - d_h),\ s_h \leftarrow \sqrt{2a_h},\ t_h \leftarrow -d_{h+1} s_h$
  $u_h \leftarrow 2d_{h+1}(a_h - a_{h+1}) + (b_h - b_{h+1}),\ v_h \leftarrow -d_{h+1} u_h$
Drop $u_l,\ v_l$ if $u_l = v_l = 0$ for all l from 1 to m, get L ReLU terms and reindex
Drop $s_h,\ t_h,\ \tau_h$ if $s_h = t_h = 0$ for all h from 1 to m, get H ReHU terms and reindex

Output: ReLU coefficients and intercepts $u_l,\ v_l,\ l = 1, 2, \ldots L$
ReHU coefficients, intercepts, and cutoffs $s_h,\ t_h,\ \tau_h,\ h = 1, 2, \ldots H$

As indicated in Proposition 1 of Dai and Qiu (2024), the composite ReLU-ReHU function class is closed under affine transformations, with the ReLU-ReHU parameters specified accordingly. Thus, we can leverage this affine property to obtain ReLU-ReHU representations for all sample-specific PLQ losses from the prototype loss. Specifically, suppose the ReLU-ReHU representation of the prototype loss is given as:

$$L(z) = \sum_{l=1}^{L} \mathrm{ReLU}(u_l z + v_l) + \sum_{h=1}^{H} \mathrm{ReHU}_{\tau_h}(s_h z + t_h).$$

Then, the ReLU-ReHU representation of $L_i(\cdot)$ can be obtained directly via *affine casting*:

$$L_i(z) = \sum_{l=1}^{L} \mathrm{ReLU}(C_i p_i u_l z + C_i q_i u_l + C_i v_l) + \sum_{h=1}^{H} \mathrm{ReHU}_{\sqrt{C_i}\tau_h}\big(\sqrt{C_i} p_i s_h z + \sqrt{C_i}(q_i s_h + t_h)\big). \quad (6)$$

In other words, for most ERM problems, we only need to perform ReLU-ReHU decomposition on the prototype loss; the resulting representations can then be easily extended to all instances via affine casting.

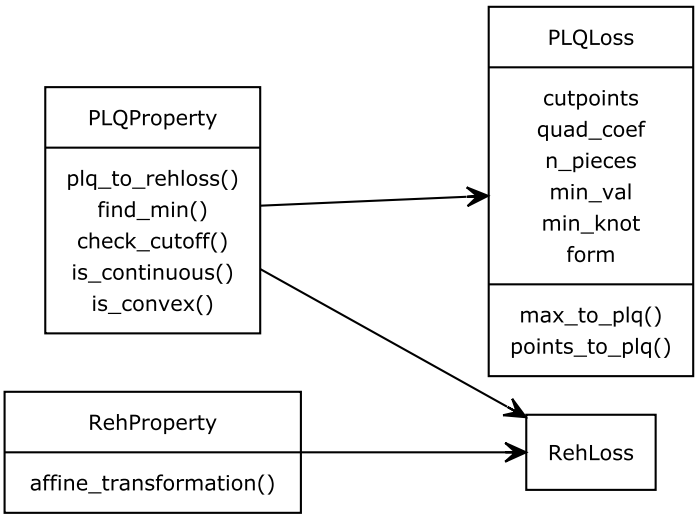

Figure 1: The UML class diagram of the `plqcom` package.

3 Implementation and Software Architecture

Our proposed method is implemented in Python using an object-oriented design. The source code is publicly available on GitHub at https://github.com/keepwith/PLQComposite. Notably, the code is written in pure Python and has minimal dependencies, relying only on the NumPy and ReHLine packages.

Figure 1 is the unified modeling language (UML) class diagram of `plqcom`, illustrating the structure of the package. In this diagram, each table represents a class, with class names in the table headers, and class attributes and operations in the table rows. In particular, operations are marked with parentheses. The arrows between classes indicate associations between them. For the sake of simplicity, we have omitted access modifiers, operation parameters, class constructors, and details about the `RehLoss` class in the ReHLine package from this diagram. The `PLQLoss` and `RehLoss` classes represent the loss functions in different forms, respectively. The `PLQProperty` and `RehProperty` classes contain operations that perform computations on the loss functions. These operations implement the algorithms described in Section 2, with additional user-friendly checks to prevent illegal inputs and operations.

4 Examples

This section presents two illustrative examples to show the usage of the `plqcom` package: a classification problem and a portfolio optimization problem. For a more comprehensive range of examples, please refer to our documentation in Jupyter Notebook at https://plqcomposite.readthedocs.io/en/latest/examples.html.

4.1 Hinge Loss and Square Loss

In this example, we consider a classification problem with a prototype loss that simultaneously incorporates the hinge loss and the square loss, with the prototype loss being a piecewise linear-quadratic (PLQ) function. Specifically, $L(z) = \max(\phi(z), \psi(z))$, where $\phi(z) = \max(1 - z, 0)$ is

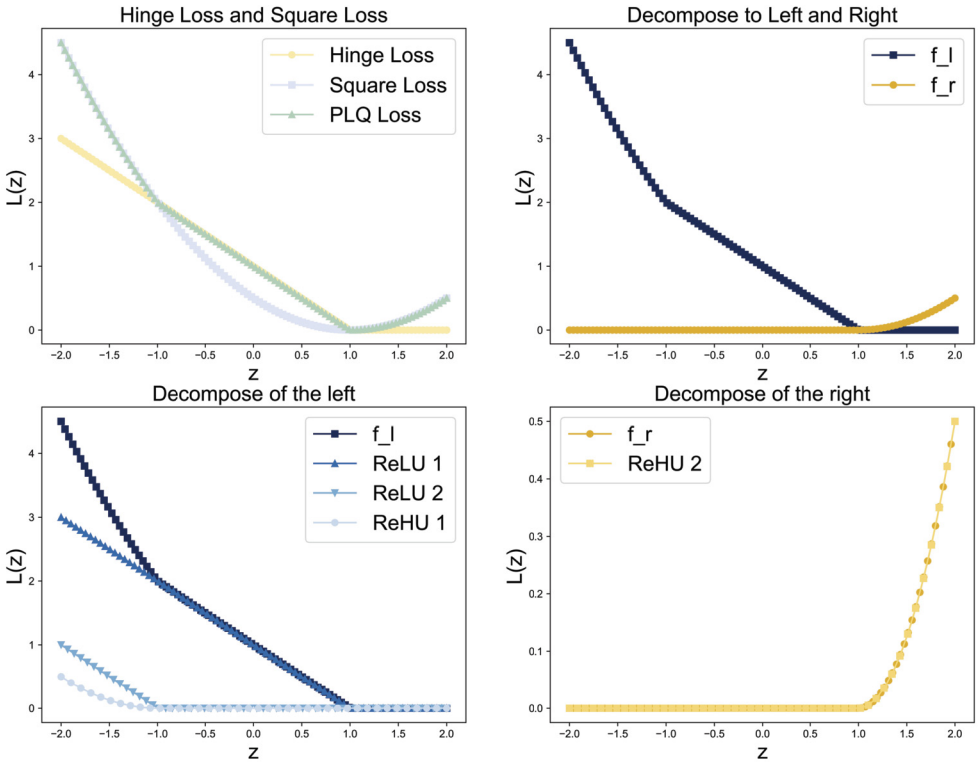

Figure 2: Transformation of prototype loss to ReLU-ReHU representation. The top-left subplot illustrates the hinge loss, square loss, and composite PLQ loss. The top-right subplot presents the PLQ loss, segmented into two components: f_l and f_r. The bottom-left and bottom-right subplots showcase f_l and f_r with the ReLU-ReHU pieces produced from their piecewise decomposition, respectively.

the hinge loss, and $\psi(z) = \frac{1}{2}(1-z)^2$ is the square loss. The prototype loss is depicted in green in the top-left subplot of Figure 2.

To obtain the ReLU-ReHU representation of a convex PLQ loss function by `plqcom`, we follow the steps outlined in Section 2. Specifically, we first input the PLQ function in its constituent representations. Next, we apply the `plq_to_rehloss()` function to perform the decomposition. Finally, we use the `affine_transformation()` function to get the ReLU-ReHU representations of sample-specific PLQ loss $L_i(z)$ and solve the ERM problem by ReHLine. The workflow is straightforward and easy to implement.

Example 1: Hinge loss and square loss

```
# Step 0: data generation
import numpy as np
from plqcom import PLQLoss, plq_to_rehloss, affine_transformation
from rehline import ReHLine
n, d, C = 1000, 3, 1.0
np.random.seed(1024)
X = np.random.randn(n, d)
```

```
beta = np.random.randn(d)
y = np.sign(X.dot(beta) + np.random.randn(n))

# Step 1: specify the prototype PLQ loss L(z)
plqloss = PLQLoss( quad_coef={'a': np.array([0., 0., 0.5]), 'b': np.array([0.,
    -1., -1.]), 'c': np.array([0., 1., 0.5])}, form='max')

# Step 2: decompose to ReHU-ReLU representation
rehloss = plq_to_rehloss(plqloss)

# Step 3: affine casting to produce sample-specific loss
rehloss = affine_transformation(rehloss, n=X.shape[0], c=C, p=y, q=0)

# Step 4: use the rehline to solve the problem
clf = ReHLine(C=C)
clf.U, clf.V = rehloss.relu_coef, rehloss.relu_intercept
clf.S, clf.T, clf.Tau = rehloss.rehu_coef, rehloss.rehu_intercept, rehloss.rehu_cut
clf.fit(X=X)
```

4.2 Portfolio Optimization

In this example, we consider a portfolio optimization problem as follows:

$$\min_{\omega_1,\ldots,\omega_n} \sum_{i=1}^{n}\left(\rho(\omega_i)+\frac{1}{2}\omega_i^2\right), \qquad \text{s.t. } \sum_{i=1}^{n}\omega_i = 1, \text{ and } \sum_{i=1}^{n}\omega_i\gamma_i \geqslant \alpha, \tag{7}$$

where n is the number of stocks, $\omega_i \in \mathbb{R}$ is the weight of the i-th stock with $\omega_i < 0$ meaning shorting the stock and $\omega_i > 0$ longing the stock, $\gamma_i \in \mathbb{R}$ is the expected return of the i-th stock, $\alpha \in \mathbb{R}$ is the minimum requirements for the expected return of the portfolio, and $\rho : \mathbb{R} \to \mathbb{R}^+$ is the transaction cost given by a univariate convex PLQ function. The workflow is similar to the previous example except for the input representation of the prototype PLQ function.

Example 2: Portfolio optimization

```
# Step 0: data generation
import numpy as np
from plqcom import PLQLoss, plq_to_rehloss, affine_transformation
from rehline import ReHLine
n, C = 10, 1.0
np.random.seed(1024)
X = np.eye(n)
r = -0.5 +np.random.rand(10)

# Step 1: specify the prototype PLQ loss L(z)
plqloss = PLQLoss(points=np.array([[-0.75, 0.6], [-0.5, 0.3],[-0.25, 0.1],[0, 0],
    [0.25, 0.1], [0.5, 0.3], [0.75, 0.6]]),form='points')
```

```
# Step 2: decompose to ReHU-ReLU representation
rehloss = plq_to_rehloss(plqloss)

# Step 3: affine casting to produce sample-specific loss
rehloss = affine_transformation(rehloss, n=X.shape[0], c=C, p=1, q=0)

# Step 4: use the rehline to solve the problem
A = np.array([r,np.ones(10)])
b = np.array([-0.3, -1])
clf = ReHLine(C=C)
clf.U, clf.V, = rehloss.relu_coef, rehloss.relu_intercept
clf.A, clf.b = A, b
clf.fit(X=X)
```

5 Summary

This paper presents `plqcom`, an open-source Python package designed to transfer convex PLQ functions into their ReLU-ReHU representations. `plqcom` features a fully object-oriented design with user-friendly APIs, and its source code and detailed documentation are publicly available on GitHub. While our algorithms for converting PLQ loss forms can be further improved, we plan to extend our toolkit by releasing an R package in the future to cater to a broader range of researchers. In addition to this, both `plqcom` and ReHLine only focus on one-dimensional PLQ functions concerning $\mathbf{x}^\intercal\boldsymbol{\beta}$. An interesting avenue for future work would be to extend the form of the loss function to higher-dimensional PLQ functions, as well as to explore various combinations of $\mathbf{x}$ and $\boldsymbol{\beta}$. We hope that the data science community will engage with our work, utilize our toolkit in conjunction with ReHLine, and contribute to the development of ERM problem solvers.

Supplementary Material

The data, code, and README are all available at https://github.com/keepwith/PLQComposite.

A Technical Proofs

A.1 Proof of Lemma 1

Proof. Without loss of generality, if the minimum value of $L(z)$ is not equal to 0, we can subtract it to make the minimum value equal to zero. Let $z = d_{j^*}$ as the point where the minimum is achieved, with $L(d_{j^*}) = 0$, and $d_{j^*} \in \{d_1, \ldots, d_{m-1}\}$. Define $d_0 = -\infty, d_m = \infty$. We can decompose $L(z)$ into $f_l(z)$ and $f_r(z)$ as follows.

$$f_l(z) = \begin{cases} L(z), & \text{if } z \leqslant d_{j^*}, \\ 0, & \text{if } z > d_{j^*}, \end{cases} \quad f_r(z) = \begin{cases} 0, & \text{if } z < d_{j^*}, \\ L(z), & \text{if } z \geqslant d_{j^*}. \end{cases}$$

We have $L(z) = f_l(z) + f_r(z)$, where $z \in \mathbb{R}$ and both $f_l(z)$, $f_r(z)$ are convex PLQ functions. For $h = j^* + 1, j^* + 2, \ldots, m$, we define $g_h(z)$ as follows.

$$g_h(z) = \begin{cases} 0 & \text{if } z < d_{h-1}, \\ f_r(z) - [f'_{r,-}(d_{h-1})(z - d_{h-1}) + f_r(d_{h-1})] & \text{if } d_{h-1} \leqslant z \leqslant d_h, \\ f'_{r,-}(d_h)(z - d_h) + f_r(d_h) - [f'_{r,-}(d_{h-1})(z - d_{h-1}) + f_r(d_{h-1})] & \text{if } z > d_h, \end{cases}$$

where d_h, d_{h-1} are the h^{th} and the $(h-1)^{th}$ cutpoint of $L(z)$. From the definition of $g_h(z)$, easy to verify $\sum_{h=j^*+1}^{m} g_h(z) = f_r(z)$. Then it suffices to check each $g_h(z)$ is a combination of ReLU and ReHU functions.

Case (i) If $d_{h-1} \leqslant z \leqslant d_h$,

$$\begin{aligned} g_h(z) &= f_r(z) - \big[f'_{r,-}(d_{h-1})(z - d_{h-1}) + f_r(d_{h-1})\big] \\ &= a_h z^2 + b_h z + c_h - \big[(2a_{h-1}d_{h-1} + b_{h-1})(z - d_{h-1}) + a_h d_{h-1}^2 + b_h d_{h-1} + c_h\big] \\ &= \frac{(\sqrt{2a_h}z - \sqrt{2a_h}d_{h-1})^2}{2} + \big[2d_{h-1}(a_h - a_{h-1}) + (b_h - b_{h-1})\big](z - d_{h-1}) \\ &= \mathrm{ReHU}_{\tau_h}(s_h z + t_h) + \mathrm{ReLU}(u_h z + v_h), \end{aligned}$$

where $f'_{r,-}(z)$ is the left derivative of f_r, $\tau_h = \sqrt{2a_h}(d_h - d_{h-1})$, $s_h = \sqrt{2a_h}$, $t_h = -d_{h-1}s_h$, $u_h = 2d_{h-1}(a_h - a_{h-1}) + (b_h - b_{h-1})$, $v_h = -d_{h-1}u_h$, a_h, b_h, c_h, a_{h-1}, b_{h-1}, c_{h-1} are the coefficients for the h^{th} and the $(h-1)^{th}$ piece of $L(z)$.

Case (ii) If $z > d_h$,

$$\begin{aligned} g_h(z) &= \big[f'_{r,-}(d_h)(z - d_h) + f_r(d_h)\big] - \big[f'_{r,-}(d_{h-1})(z - d_{h-1}) + f_r(d_{h-1})\big] \\ &= \big[(2a_h d_h + b_h)(z - d_h) + a_h d_h^2 + b_h d_h + c_h\big] \\ &\quad - \big[(2a_{h-1}d_{h-1} + b_{h-1})(z - d_{h-1}) + a_h d_{h-1}^2 + b_h d_{h-1} + c_h\big] \\ &= 2(a_h d_h - a_{h-1}d_{h-1})z + (b_h - b_{h-1})z + \big(2a_{h-1}d_{h-1}^2 - a_h d_h^2 - a_h d_{h-1}^2\big) - (b_h - b_{h-1})d_{h-1} \\ &= \sqrt{2a_h}(d_h - d_{h-1})\left(\sqrt{2a_h}(z - d_{h-1}) - \frac{1}{2}\sqrt{2a_h}(d_h - d_{h-1})\right) \\ &\quad + \big[2d_{h-1}(a_h - a_{h-1}) + (b_h - b_{h-1})\big](z - d_{h-1}) \\ &= \mathrm{ReHU}_{\tau_h}(s_h z + t_h) + \mathrm{ReLU}(u_h z + v_h). \end{aligned}$$

Case (iii) If $z < d_{h-1}$,

$$g_h(z) = 0 = \mathrm{ReHU}_{\tau_h}(s_h z + t_h) + \mathrm{ReLU}(u_h z + v_h).$$

Thus, $g_h(z)$ is a combination of ReLU and ReHU.

$$f_r(z) = \sum_{h=j^*+1}^{m} g_h(z) = \sum_{h=1}^{L_r} \mathrm{ReLU}(u_h z + v_h) + \sum_{h=1}^{H_r} \mathrm{ReHU}_{\tau_h}(s_h z + t_h),$$

where L_r is the number of nonzero ReLU functions in the right part, and H_r is the number of nonzero ReHU functions in the right part. A similar result applies to $f_l(z)$:

$$f_l(z) = \sum_{h=0}^{j^*-1} g_h(z) = \sum_{h=1}^{L_l} \mathrm{ReLU}(u_h z + v_h) + \sum_{h=1}^{H_l} \mathrm{ReHU}_{\tau_h}(s_h z + t_h).$$

Combining $f_l(z)$ and $f_r(z)$, we obtain a ReLU-ReHU representation for $L(z)$.

$$L(z) = f_l(z) + f_r(z) = \sum_{h=1}^{L} \mathrm{ReLU}(u_h z + v_h) + \sum_{h=1}^{H} \mathrm{ReHU}_{\tau_h}(s_h z + t_h).$$

□

References

Bandler J, Chen SH, Biernacki R, Gao L, Madsen K, Yu H (1993). Huber optimization of circuits: a robust approach. *IEEE Transactions on Microwave Theory and Techniques*, 41(12): 2279–2287. https://doi.org/10.1109/22.260718

Benine-Neto A, Scalzi S, Mammar S (2011). Vehicle lane keeping control based on piecewise affine regions. In: *International IEEE Conference on Intelligent Transportation Systems (ITSC)*, 907–912. IEEE.

Dai B, Qiu Y (2024). ReHLine: regularized composite ReLU-ReHU loss minimization with linear computation and linear convergence. *Advances in Neural Information Processing Systems*, 36.

Fukushima K (1969). Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Transactions on Systems Science and Cybernetics*, 5(4): 322–333. https://doi.org/10.1109/TSSC.1969.300225

Garcia-Rubio R, Bayón L, Grau JM (2014). Generalization of the firm's profit maximization problem: An algorithm for the analytical and nonsmooth solution. *Computational Economics*, 43(1): 1–14. https://doi.org/10.1007/s10614-013-9378-7

Gardiner B, Lucet Y (2010). Convex hull algorithms for piecewise linear-quadratic functions in computational convex analysis. *Set-Valued and Variational Analysis*, 18(3–4): 467–482. https://doi.org/10.1007/s11228-010-0157-5

Hein M, Andriushchenko M, Bitterwolf J (2019). Why relu networks yield high-confidence predictions far away from the training data and how to mitigate the problem. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 41–50. IEEE Computer Society, Los Alamitos, CA, USA.

Huber PJ (1964). Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, 35(1): 73–101. https://doi.org/10.1214/aoms/1177703732

Jensen DL, King AJ (1992). Frontier: A graphical interface for portfolio optimization in a piecewise linear-quadratic risk framework. *IBM Systems Journal*, 31(1): 62–70. https://doi.org/10.1147/sj.311.0062

Portnoy S, Koenker R (1997). The Gaussian hare and the Laplacian tortoise: Computability of squared-error versus absolute-error estimators. *Statistical Science*, 12(4): 279–300. https://doi.org/10.1214/ss/1030037960

Rockafellar RT (1988). First- and second-order epi-differentiability in nonlinear programming. *Transactions of the American Mathematical Society*, 307(1): 75–108. https://doi.org/10.1090/S0002-9947-1988-0936806-9

Vapnik V (1998). *Statistical Learning Theory*, volume 2, 831–842. John Wiley & Sons.

Vapnik V (2006). *Estimation of Dependences Based on Empirical Data*. Springer Science & Business Media.