Journal of Convex Analysis
Volume 32 (2025), No. 2, 467–486

Closest Univariate Convex Linear-Quadratic Function Approximation with Minimal Number of Pieces

Namrata Kundu, Yves Lucet*
Dept. of Computer Science, I. K. Barber Faculty of Science, University of British Columbia Okanagan, Kelowna, Canada
yves.lucet@ubc.ca

Dedicated to Ralph Tyrrell Rockafellar on the occasion of his 90th birthday.

Received: March 31, 2024
Accepted: October 6, 2024

We compute the closest convex piecewise linear-quadratic (PLQ) function with minimal number of pieces to a given univariate piecewise linear-quadratic function. The Euclidean norm is used to measure the distance between functions. First, we assume that the number and positions of the breakpoints of the output function are fixed, and solve a convex optimization problem. Next, we assume the number of breakpoints is fixed, but not their position, and solve a nonconvex optimization problem to determine optimal breakpoints placement. Finally, we propose an algorithm composed of a greedy search preprocessing and a dichotomic search that solves a logarithmic number of optimization problems to obtain an approximation of any PLQ function with minimal number of pieces thereby obtaining in two steps the closest convex function with minimal number of pieces.
We illustrate our algorithms with multiple examples, compare our approach with a previous globally optimal univariate spline approximation algorithm, and apply our method to simplify vertical alignment curves in road design optimization. CPLEX, Gurobi, and BARON are used with the YALMIP library in MATLAB to effectively select the most efficient solver.

Keywords: Closest convex function, piecewise linear-quadratic, spline approximation, global optimization.

2020 Mathematics Subject Classification: 52A10, 65D07, 90C26.

1. Introduction

Finding the closest convex function is important in various disciplines where convexity is often a desirable property. In control theory, especially in model predictive control, the system's dynamics or the constraints can be nonconvex. Approximating these nonconvex parts with convex PLQ functions allows the application of convex optimization techniques, simplifying controller design [36]. In the domain of signal processing and compression, approximating complex signals with simpler, piecewise representations can make them easier to analyze, transmit, and store. Reducing the number of pieces in the approximation can lead to more efficient compression

This work was partly supported by the second author Discovery grant RGPIN-2018-03928 (Lucet) from the Natural Sciences and Engineering Research Council of Canada (NSERC).
*Corresponding author.

ISSN 0944-6532 / $ 2.50

algorithms. For instance, methods like wavelet compression, which are akin to PLQ representations, have seen widespread use in image compression [29].

In machine learning, particularly in regression and classification tasks, a convex loss function can be efficiently optimized. Nonconvex loss functions can be approximated by convex PLQ functions to leverage the benefits of convex optimization [41]. For algorithms like decision trees and certain neural networks, piecewise approximations can serve as activation functions or decision boundaries. Simplifying these approximations can accelerate training and inference, and potentially reduce overfitting [14].

In economics, utility functions, production functions, and other vital functions might not always be convex. Approximating them with convex PLQ functions can simplify the analysis and lead to more tractable models. This is especially important in game theory, where convexity plays an important role in equilibrium concepts [34]. In financial modeling, especially in risk management where risks often manifest as nonconvexities in models, finding the closest convex PLQ function can be used for portfolio optimization, where the objective is often to maximize returns while ensuring a convex risk profile [19, 30].

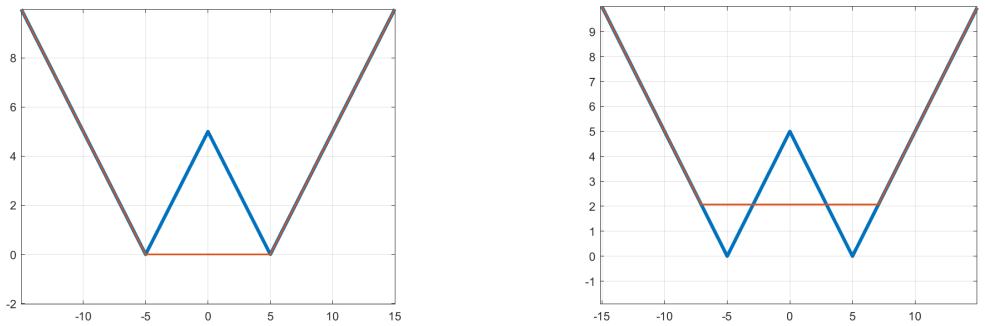
In areas such as quantum mechanics or thermodynamics, complex behaviors can sometimes be approximated using PLQ functions, especially when a high level of precision is not required. These approximations can simplify calculations and make certain analytical solutions possible.

Beyond the above application areas, one of our motivation is to build a toolbox for manipulating convex functions and operators in computational convex analysis [9, 10, 11, 16, 18, 25, 26, 27, 28]. Due to floating point errors, composing multiple operators results in nonconvex functions in practice, even though such functions, like the convex conjugate, should always be convex in theory. This issue prevents applying any algorithm that relies on convexity. After investigating convexity tests for piecewise functions [2, 42], the present paper proposes optimization models with convexity constraints to compute the closest convex function.

Our approach is not restricted to convexity and allows to simplify piecewise functions by approximating them with a piecewise function having a minimal number of pieces. In Section 5, we apply our algorithm to the vertical alignment problem in road design where the computation time of calculating an optimal road design depends directly on the number of intervals (called segments in road design) composing the vertical alignment. Decreasing the number of intervals greatly improves the computation time [1, 4, 20, 32].

The closest convex function should not be confused with the convex envelope [7, 25]; see Figure 1.We are not aware of any previous work to compute the closest convex function of a given piecewise linear-quadratic (PLQ) function. Previous work on computing piecewise functions with minimal number of intervals focused on piecewise-constant functions [23] where the authors propose polynomial-time algorithms leveraging shortest-path and dynamic programming techniques.

Piecewise-linear approximations have also been investigated with the Ramer–Douglas–Peucker algorithm [8, 35], Visvalingam-Whyatt algorithm [45], and the Reumann-Witkam algorithm [38]. The authors of [43] find that the Ramer–Douglas–Peucker runs in $O(n \log n)$ but is not optimal.



(a) Convex envelope (red) (b) Closest convex function (red)

Figure 1. PLQ function (blue) compared to its convex envelope and closest convex function.

For the Haussdorff distance, [21] improved by [6] is optimal in $O(n^2)$ while an optimal algorithm for the Fréchet distance runs in $O(n^3)$ [12]. Other research in this domain includes [5, 13, 22, 37, 46]. Our numerical experiments show that our models are solved with a $O(n^2)$ complexity.

Our research tackles the problem of finding the closest convex function for a given univariate PLQ function from multiple perspectives. We have formulated four optimization algorithms, each with distinct objectives and methodologies. All our algorithms are implemented in MATLAB using the YALMIP [24] numerical library so one can easily change the solver used.

The first algorithm aims to determine the closest convex function with the same breakpoints as the input univariate PLQ function. The resulting convex quadratic programming optimization problem is solved effectively using CPLEX, and we obtain accurate results in polynomial time.

The second algorithm deals with a more complex scenario where the number of pieces in the output convex function is set as part of the input, but their positions, that is, the breakpoints, become variable. This flexibility introduces a nonconvex nonlinear programming problem. By leveraging the state-of-the-art global optimization solver BARON [40], we are able to attain accurate results.

The above two algorithms compute the closest convex PLQ function, but may result in a representation with numerous redundant pieces. Algorithm 3 focuses on computing an optimal number of pieces to minimize the size of the function representation. It employs a dichotomy search strategy that solves an optimization problem at each step. It takes a convex PLQ function as input, which may be obtained from the output of the second algorithm. The overall objective is to determine the closest convex PLQ function with the minimum number of pieces, while adhering to a predefined error tolerance. To accomplish this, we solve an optimization problem using BARON at each step of the dichotomy search.

Lastly, we extend and generalize our algorithm 3 to approximate with minimal number of pieces any (not necessarily convex) PLQ function.

Noting n (resp. m)the number of pieces of the input (output) function, algorithm 1 (resp. 2, 3, 4) requires $n \leq m$ (resp. $n \leq m$, $n \geq m$, $n \geq m$).

The paper is organized as follows. In the next section, we introduce certain definitions, notations and theorems that will be used throughout. In Section 3, we formulate our first optimization models to determine the closest convex function first with fixed breakpoints and next with variable breakpoints. Section 4 focuses on minimizing the number of pieces with one algorithm preserving convexity and the other providing nonconvex approximation. Numerical experiments are reported in Section 5 including a comparison with [31], and an application to simplifying the representation of a vertical alignment in road design. Section 6 summarizes our key findings and discusses avenues for future research.

2. Preliminaries and notations

We set our notations and recall definitions and properties used.

We denote as usual $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$. The (effective) *domain* of a function $f : \mathbb{R}^d \to \mathbb{R} \cup \{+\infty\}$ is the set $\operatorname{dom} f = \{x \in \mathbb{R}^d : f(x) < +\infty\}$. A function f is called *proper* if for all x, $f(x) > -\infty$ and $\operatorname{dom} f \neq \emptyset$.

A function $f : \mathbb{R}^d \to \overline{\mathbb{R}}$ is called *piecewise linear-quadratic* (PLQ) if $\operatorname{dom} f$ can be represented as the union of finitely many polyhedral sets, relative to each of which $f(x)$ is given by a quadratic expression [39, Definition 10.20]. We call *breakpoints* a set $\{x_i : i = 1, \ldots, m\}$ with $x_i < x_{i+1}$. We allow $x_1, x_m \in \mathbb{R} \cup \{+\infty\}$ while $x_i \in \mathbb{R}$, $i = 2, \ldots, m-1$. A *univariate* PLQ function f is defined on the real line as

$$f(x) = \begin{cases} a_1x^2 + b_1x + c_1, & x \in (x_1, x_2], \\ a_2x^2 + b_2x + c_2, & x \in (x_2, x_3], \\ \vdots & \\ a_mx^2 + b_mx + c_m, & x \in (x_m, x_{m+1}]; \end{cases} \tag{1}$$

where $a_i, b_i, c_i \in \mathbb{R}$ for $i = 1, ..., m$. To be consistent with the definition in higher dimensions, we always impose that PLQ functions are continuous on the relative interior of their domain, i.e., the only points where f may be discontinuous are x_1 when $f(x) = +\infty$ for $x < x_1$, and x_m when $f(x) = +\infty$ when $x > x_m$.

We call a *piece* of f, a pair $([x_i, x_{i+1}], f_i)$ although we slightly abuse that notation to sometimes refer to an interval $[x_i, x_{i+1}]$ as a piece.

The *2-norm* (also called L_2 *norm* or *Euclidean norm*) of a measurable function $f : \mathbb{R} \to \mathbb{R}$ is noted

$$\|f\|_2 = \sqrt{\int_{\operatorname{dom} f} [f(x)]^2 \, dx}.$$

We say that a function $f : \mathbb{R} \to \mathbb{R} \cup \{+\infty\}$ is *coercive* if $\lim_{|x| \to +\infty} f(x) = +\infty$. A function f is *lower semicontinuous* (lsc) at a point x if for every sequence $x_i \to x$, we have $f(x) \leq \liminf_i f(x_i)$.

We recall the following fundamental existence result.

Theorem 2.1. ([3]) *Let a function f be a lsc coercive proper function. Then f has a (global) minimizer, that is, there exists $\bar{x}$ such that $f(\bar{x}) = \min_x f(x)$.*

We will also use the following corollary.

Corollary 2.2. ([3]) *Assume $f : \mathbb{R} \to \mathbb{R} \cup \{+\infty\}$ is coercive and lower semicontinuous. Let C be a closed subset of $\mathbb{R}$, and assume that $C \cap \operatorname{dom} f \neq \emptyset$. Then $f|_C$ has a minimizer.*

Finally, we call a *spline* a function $f : \mathbb{R} \to \mathbb{R} \cup \{+\infty\}$ defined by a set of breakpoints x_i such that (i) on each interval $[x_i, x_{i+1}]$, $f(x)$ is a polynomial of degree k, and (ii) $f(x)$ has continuous derivatives up to order $k-1$ at each x_i, for $i = 2, ..., m-1$.

We note the following set of PLQ functions

$$S(x) = \{f : f \text{ is a convex PLQ function with breakpoints } x \in \mathbb{R}^m\},$$

and consider the following optimization problem for a given $f \in S(x)$

$$\min_{\rho \in S(x)} \|f - \rho\|_2^2 \tag{2}$$

whose solution is the closest convex PLQ function that has the same breakpoints as f. We will also consider variable breakpoints by solving for a given m

$$\min_{\eta \in \overline{\mathbb{R}}^{m+1}, \rho \in S(\eta)} \|f - \rho\|_2^2. \tag{3}$$

3. Closest convex PLQ function

Given a PLQ function, we propose two algorithms to compute its closest convex PLQ function. Algorithm 1 solves (2); it requires that the breakpoints of the output PLQ function ρ are the same as the breakpoints of the input function f. By contrast, algorithm 2 only requires the number of breakpoints to solve (3).

A function f given by (1) is represented by

$$\mathbf{C}(f) = \begin{bmatrix} a_1 & a_2 & ... & a_m \\ b_1 & b_2 & ... & b_m \\ c_1 & c_2 & ... & c_m \end{bmatrix}, \quad \mathbf{P}(f) = [x_1, x_2, ..., x_{m+1}],$$

where $a = [a_1, a_2, ..., a_m]^T$, $b = [b_1, b_2, ..., b_m]^T$, and $c = [c_1, c_2, ..., c_m]^T$ are the vectors of the quadratic, linear, and constant coefficients of f respectively; and $\mathbf{P}(f)$ is the list of breakpoints of f.

The output of our algorithms (the closest convex PLQ function) is a PLQ function that we denote ρ. It is defined on n pieces (algorithms 1 and 2 enforce $n \geq 2m$) and is stored as

$$\mathbf{C}(\rho) = \begin{bmatrix} \alpha_1 & \alpha_2 & ... & \alpha_n \\ \beta_1 & \beta_2 & ... & \beta_n \\ \gamma_1 & \gamma_2 & ... & \gamma_n \end{bmatrix}, \mathbf{P}(\rho) = [\eta_1, \eta_2, ..., \eta_{n+1}]. \tag{4}$$

In the remainder of the paper, we refer to functions defined on a piece i as f_i, which represents $a_i x^2 + b_i x + c_i$ defined over an interval $[x_i, x_{i+1}]$.

Now, we consider the boundedness of the domain of the input PLQ function f. When the input function equals a quadratic function on an unbounded interval, the closest convex function is equal to that quadratic on that interval. (If the input

function has a negative quadratic coefficient on any unbounded interval, then the Euclidean distance to any convex function is infinity; it is trivial to detect and deal with this case.) Otherwise, we adjust the domain of the output function ρ to match the domain of the input function f.

Then Algorithm 1 returns a solution to the optimization problem (all variables are written in blue).

$$\begin{cases} \min\limits_{\alpha_i,\beta_i,\gamma_i} & \sum\limits_{i=1}^{n} \int_{\eta_i}^{\eta_{i+1}} (f(x) - \rho(x))^2 dx \\ \text{subject to} & \alpha_i \geq 0, \qquad i = 1, \ldots, n; \\ & \rho_i(\eta_{i+1}) = \rho_{i+1}(\eta_{i+1}), \qquad i = 1, \ldots, n-1; \\ & \rho_i'(\eta_{i+1}) \leq \rho_{i+1}'(\eta_{i+1}), \qquad i = 1, \ldots, n-1. \end{cases} \tag{5}$$

All 3 constraints are linear and enforce the convexity of the output function ρ. The resulting optimization problem is a convex quadratic programming problem. Note that the optimization model only considers the function ρ on a bounded domain making the integral in the objective function finite since all functions are continuous.

Algorithm 2 sets ρ_1 and ρ_n like algorithm 1, i.e., they are equal to the input function on any unbounded interval. It then proceeds to solve for a given tolerance $\delta > 0$

$$\begin{cases} \min\limits_{\alpha_i,\beta_i,\gamma_i,\eta_i} & \sum\limits_{i=1}^{n} \int_{\eta_i}^{\eta_{i+1}} (f(x) - \rho(x))^2 dx \\ \text{subject to} & \alpha_i \geq 0, \qquad i = 1, \ldots, n; \\ & \rho_i(\eta_{i+1}) = \rho_{i+1}(\eta_{i+1}), \qquad i = 1, \ldots, n-1; \\ & \rho_i'(\eta_{i+1}) \leq \rho_{i+1}'(\eta_{i+1}), \qquad i = 1, \ldots, n-1; \\ & \eta_i \leq \eta_{i+1} - \delta, \qquad i = 1, \ldots, n-1; \\ & \mathbf{P}(f) \subset \mathbf{P}(\rho). \end{cases} \tag{6}$$

The constraints in (6) include the convexity constraints similar to (5), but adds that the breakpoints must be an increasing vector. It also requires the breakpoint vector η to include the breakpoints x of the input function f.

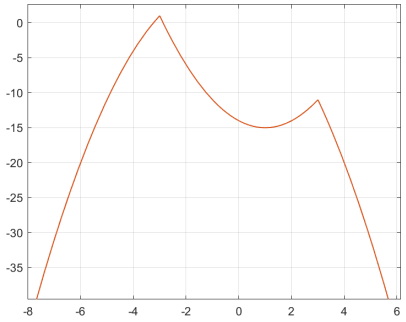
Problem (6) is a nonconvex nonlinear optimization problem that requires a global solver like BARON to solve. To reduce the search space, we add the following bounds

$$0 \leq \alpha_i \leq \max(a), \tag{7a}$$

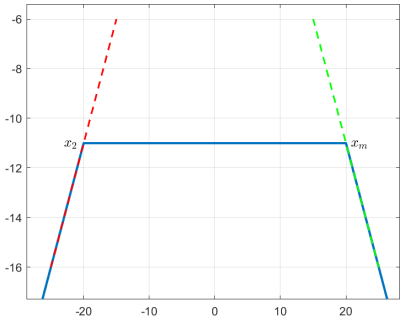
$$\min(b) - N \leq \beta_i \leq \max(b) + N, \tag{7b}$$

$$\min(t_1, t_2) \leq \gamma_i \leq \max(f(x_{bounded})); \tag{7c}$$

where t_1 represents the intercept of the tangent drawn at $f_1(x_{0.5})$ and t_2 represents the intercept of the tangent drawn at $f_m(x_{m+0.5})$ with $x_{1.5}$, $x_{m+0.5}$ as per Lemma 3.4. (If the conditions of the lemma are not satisfied, use $x_{1.5} = x_1$ and $x_{m+0.5} = x_m$.) The integer N is an arbitrarily large number (default to $N = 1,000$). In practice, we apply the above bounds and if we find that any bound is tight, we increase it and resolve the problem until none of the bound constraints (7) is active.



(a) Case (8a): concave quadratic.



(b) Case (8b): decreasing derivative.

Figure 2: PLQ functions admitting no closest convex function.

We now turn our attention to the correctness of algorithms 1 and 2 by considering the following cases.

$$(x_1 = -\infty \text{ and } a_1 < 0), \text{ or } (x_{m+1} = +\infty \text{ and } a_m < 0), \tag{8a}$$

$$x_1 = +\infty, x_{m+1} = +\infty, a_1 = a_m = 0, \text{ and } f_1'(x_2) > f_m'(x_m). \tag{8b}$$

In case (8a), the input function f has a strictly concave quadratic part on an unbounded interval (see Figure 3.1a), while in case (8b) the derivative values at x_2 and x_{m-1} make it impossible to build a convex approximation whose derivative has to be nondecreasing (see figure 3.1b).

Lemma 3.1. *Assume f is univariate proper PLQ. Then the following are equivalent:*

(i) *f admits a closest convex function,*
(ii) *neither* (8a) *nor* (8b) *hold,*
(iii) *there exists an affine minorant to f,*
(iv) $\operatorname{dom} f^* \neq \emptyset$,
(v) $\operatorname{co} f \not\equiv -\infty$.

Proof. If there exists an affine function minoring f then $\operatorname{dom} f^* \neq \emptyset$; so we have $\operatorname{co} f = f^{**} \not\equiv -\infty$. The later cannot hold if either (8a) or (8b) hold. The last implication follow because f is PLQ. Hence, (ii)–(v) hold. Lemma 3.3 below concludes the proof. □

Remark 3.2. The assumption that f is PLQ is required. Consider $f(x) = 1/\sqrt{-x}$ if $x \leq -1$, $f(x) = 1$ if $-1 < x \leq 1$ and $f(x) = 1/\sqrt{x}$ if $x > 1$. Then conditions (ii)–(iv) hold, but $\|f\| = +\infty$ and f does not admit a closest convex function. □

Lemma 3.3. *Assume f satisfies* (8a) *or* (8b). *Then $||f - g|| = \infty$ for any convex function g.*

Proof. For case (8a), assume that $x_1 = -\infty$ and $a_1 < 0$, and note g any convex function. Consider the linear function L that goes through x_2, $g(x_2)$ with slope $g'(x_2)$. Then $\infty = ||L - f|| \leq ||f - g||$. The symmetric case $x_{m+1} = \infty$ is proven similarly.

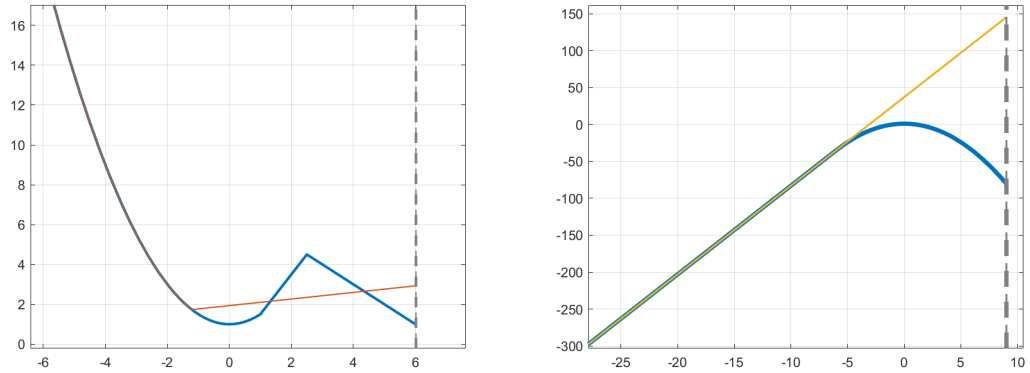
Now consider case (8b), and assume g is a convex function. If $g \neq f_1$ on $(x_1, x_2]$ or $g \neq f_m$ on $[x_m, x_{m+1})$, then $\|f - g\| = \infty$. But if $g = f_1$ on $(x_1, x_2]$ and $g = f_m$ on $[x_m, x_{m+1})$ then g cannot be convex because $g'(x_2) = f_1'(x_2) > f_m'(x_m) = g'(x_m)$. □

Note that in some cases, we have to modify the breakpoints to obtain a closest convex function.

Lemma 3.4. *Assume* $x_1 = -\infty$, $x_{m+1} = +\infty$, $f'(x_2) > f'(x_{m-1})$, *and* $a_1, a_m > 0$. *Then there exists* $x_1 < x_{1.5} < x_2$, *and* $x_{m+0.5} > x_m$ *such that* $f'(x_{1.5}) \leq f'(x_{m+0.5})$.

Proof. We have$f'(x) \to -\infty$ when $x \to -\infty$. So there exists $x_{1.5} \in \mathbb{R}$ such $f'(x_{1.5}) < f'(x_2)$. Similarly, there exists $x_{m+0.5} \in \mathbb{R}$ with $f'(x_m) < f'(x_{m+0.5})$. □

Lemma 3.4 is illustrated in Figure 3.



(a) Added $x_{0.5}$ to breakpoint list to compute closest convex function.

(b) Concave function with right-bounded domain.

Figure 3: PLQ functions admitting a closest convex function after applying Lemma 3.4.

We can now replace $\mathbf{P}(f)$ with $[x_{1.5}, \mathbf{P}(f)[2:m], x_{m+0.5}]$ to obtain a different data representation for the same function f. With this new data representation, we can solve (2) and (3).

Proposition 3.5. *Assume* f *does not satisfy* (8a) *nor* (8b). *Then problems* (2) *and* (3) *admit solutions. Moreover, algorithm* 1 *provides a solution to* (2) *while algorithm* 2 *gives a solution to* (3).

Proof. Invoking lemma 3.4 if needed, if $x_1 = -\infty$ (resp. $x_{m+1} = +\infty$), algorithm 1 sets $\rho_1 = f_1$ (resp. $\rho_m = f_n$) and reduces the calculation of the cost function to computing a finite integral. The resulting function is a solution to (2). The fact that (2) has a solution is a consequence of theorem 2.1. Similarly, (3) has (at least) one solution and algorithm 2 returns a solution. Finally, corollary 2.2 shows that (3) still admits a solution after we add constraints (7). □

Remark 3.6. Algorithm 2 impose $n \geq 2m$ because our optimization models require $\mathbf{P}(f) \subset \mathbf{P}(\rho)$ in order to find breakpoints in ρ which might fall in between existing breakpoints in f. Thus we change the representation of f by including more breakpoints (the function f remains unchanged). □

Remark 3.7. For both algorithms 1 and 2, the number of constraints and decision variables increase linearly with the number of pieces n in the output convex function.

The optimization model formulation (2) is a convex quadratic programming problem. We use MATLAB [44] with YALMIP [24] to interface with the CPLEX [33]

solver to compute solutions. (We also tried using Gurobi [15] as a solver, however, we found Gurobi was unable to solve certain cases, whereas CPLEX successfully solved for all inputs.)

4. Minimal number of pieces

Algorithms 1 and 2 in the previous section require as input the number of pieces of the output function. That number is selected conservatively thereby obtaining a closest convex function representation with more pieces than necessary. This nonoptimal representation increases any computation involving such functions. Working with functions composed of fewer pieces simplifies calculations and decreases computational complexity. With this motivation, the objective of the present section is to develop an algorithm that takes as input a univariate convex PLQ function ρ, and outputs a univariate convex PLQ function g that approximates ρ within an ε error tolerance and has a minimum number of piece representation. Afterwards, we drop the convexity assumption and propose an algorithm for any PLQ function.

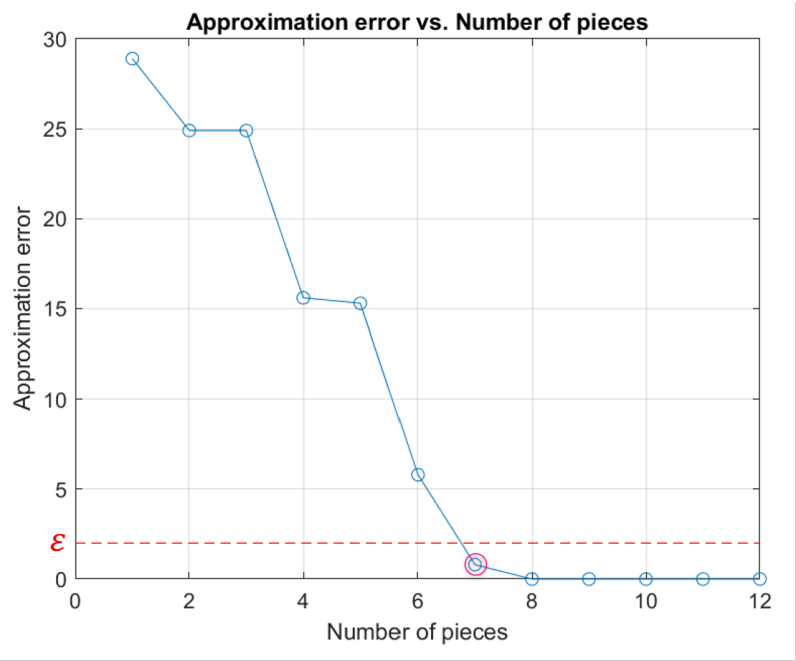


Figure 4. Approximation error vs. number of pieces. Using the nonincreasing property of the graph, we want to compute r corresponding to the red point, i.e., minimal number of pieces with approximation error lower than the error tolerance ε.

Our input is stored as the output of the algorithm of the previous section, namely (4). Our output is a PLQ function

$$g(x) = \begin{cases} \mathfrak{a}_1 x^2 + \mathfrak{b}_1 x + \mathfrak{c}_1 & \chi_0 < x \leq \chi_1, \\ \mathfrak{a}_2 x^2 + \mathfrak{b}_2 x + \mathfrak{c}_2 & \chi_1 < x \leq \chi_2, \\ \vdots & \\ \mathfrak{a}_r x^2 + \mathfrak{b}_r x + \mathfrak{c}_r & \chi_r < x < \chi_{r+1}; \end{cases} \tag{9}$$

defined on r pieces ($r \leq n$) and stored as

$$\mathbf{C}(g) = \begin{bmatrix} \mathfrak{a}_1 & \mathfrak{a}_2 & ... & \mathfrak{a}_r \\ \mathfrak{b}_1 & \mathfrak{b}_2 & ... & \mathfrak{b}_r \\ \mathfrak{c}_1 & \mathfrak{c}_2 & ... & \mathfrak{c}_r \end{bmatrix}, \mathbf{P}(g) = [\chi_1, \chi_2, ..., \chi_r, \chi_{r+1}].$$

Our first step is to build an algorithm that takes r as input and returns a PLQ approximation with minimal error. Similar to algorithm 2, we deal with unbounded intervals first, and enforce that the output function is equal to the input function on any unbounded interval. The same existence results apply as in the previous section, e.g., if $\alpha_n < 0$ then the objective function is always equal to infinity.

Next, contrary to algorithm 2, we impose $r \leq n$ leading to the following optimization problem

$$\min_{u,v,w,\mathfrak{a},\mathfrak{b},\mathfrak{c}} \quad \sum_{i=1}^{n}\sum_{j=1}^{r} u_{i,j} \int_{\eta_i}^{\eta_{i+1}} (\rho_i - g_j)^2. \tag{10}$$

$$\text{subject to} \quad \mathfrak{a}_j \geq 0,\ j = 1, \ldots, r; \tag{11}$$

$$g_j(\chi_{j+1}) = g_{j+1}(\chi_{j+1}), \quad g_j'(\chi_{j+1}) \leq g_{j+1}'(\chi_{j+1}),\ j = 1, \ldots, r; \tag{12}$$

$$\chi_j \leq \chi_{j+1} - \delta,\ j = 1, \ldots, r; \tag{13}$$

$$\mathbf{P}(g) \subset \mathbf{P}(\rho); \tag{14}$$

$$u_{i,j} \in \{0, 1\},\ v_{i,j} \in \{-1, 0, 1\},\ w_{i,j} \in \{0, 1\},\ i = 1, \ldots, n,\ j = 1, \ldots, r; \tag{15}$$

$$u_{1,1} = u_{n,r} = 1, \quad u_{0,j} = u_{n+1,j} = 0, \quad j = 1, \ldots, r; \tag{16}$$

$$\sum_{j=1}^{r} u_{i,j} = 1, \quad i = 1, \ldots, n; \tag{17}$$

$$\sum_{i=1}^{n} u_{i,j} \geq 1, \quad j = 1, \ldots, r; \tag{18}$$

$$v_{i,j} = u_{i,j} - u_{i-1,j}, \quad w_{i,j} = |v_{i,j}|,\ i = 1, \ldots, n+1,\ j = 1, \ldots, r; \tag{19}$$

$$\sum_{i} w_{i,j} = 2, \quad \sum_{i} v_{i,j} = 0, \quad j = 1, \ldots, r; \tag{20}$$

$$\sum_{j} v_{i,j} = 0, \quad i = 1, \ldots, n+1. \tag{21}$$

Constraints (11)–(12) enforce the convexity of g, and constraints (13)–(14) deal with breakpoints. Constraint (15) defines 3 sets of binary variables with u being the most important one while v is a temporary variable (that the presolver will replace; it is only here for ease of exposing), and w is introduced to store $w_{i,j} = |v_{i,j}|$. We want to impose a pattern on $u_{i,j}$ as shown in Table 1(a) where each row sums to 1 as enforced by (17). Each column should have at least one nonzero value by (18) (while that may introduce redundant pieces, we can weed them out with the dichotomy search below). Moreover, each column $u_{i,j}$ has consecutive nonzero values. To enforce that constraint, we introduce $v_{i,j} = u_{i,j} - u_{i-1,j} \in \{-1, 0, 1\}$ as the difference between consecutive values row-wise in the $u_{i,j}$ matrix. We impose that the sums over the columns of v be 0 while sums over each column of $w_{i,j}$ add up to 2 by (20). Finally, any row $i \in \{1, \ldots, n\}$ of $v_{i,j}$ has either all zeros, or a single 1 and a single -1; hence we require that their sum over the columns be zero by (21).

	g_1	g_2	g_3
Padding	0	0	0
ρ_1	1	0	0
ρ_2	1	0	0
ρ_3	1	0	0
ρ_4	1	0	0
ρ_5	0	1	0
ρ_6	0	1	0
ρ_7	0	1	0
ρ_8	0	1	0
ρ_9	0	0	1
ρ_{10}	0	0	1
ρ_{11}	0	0	1
ρ_{12}	0	0	1
Padding	0	0	0

(a) Binary variable $u_{i,j}$ including padding.

	g_1	g_2	g_3
	-	-	-
ρ_1	1	0	0
ρ_2	0	0	0
ρ_3	0	0	0
ρ_4	0	0	0
ρ_5	-1	1	0
ρ_6	0	0	0
ρ_7	0	0	0
ρ_8	0	0	0
ρ_9	0	-1	1
ρ_{10}	0	0	0
ρ_{11}	0	0	0
ρ_{12}	0	0	0
	0	0	-1

(b) Binary variable $v_{i,j}$ associated with $u_{i,j}$.

Table 1: Binary variable $u_{i,j}$ values for input function ρ with 12 pieces and output function g with 3 pieces. To correctly compute the approximation error, a pattern of contiguous ones on each column is required whereas each row should have only one nonzero value.

While $w_{i,j} = |v_{i,j}|$ is not a linear constraint, standard reformulation techniques already implemented in solvers BARON or Gurobi reformulate it as linear constraints by introducing new binary variables. For ease of presentation, we do not write such standard reformulation in our model.

As with algorithm 2, we additionally impose bound constraints to reduce the search space

$$\min(\alpha) \leq \mathfrak{a}_j \leq \max(\alpha),$$
$$\min(\beta) - N \leq \mathfrak{b}_j \leq \max(\beta) + N,$$
$$\min(t_1, t_2) \leq \mathfrak{c}_j \leq \max(\rho(\eta_{bounded})).$$

We are now in a position to present algorithm 3 to compute an approximate convex function with minimal number of pieces for a given convex PLQ function, see algorithm 3 below. The preprocessing step on line 1 merges consecutive sections into a single piece if the quadratic coefficients are close enough. In practice, it drastically cut on the number of pieces. The remaining of the algorithm is a standard dichotomy search to identify the minimal number of pieces by repeatedly solving (3).

Remark 4.1. The preprocessing greedy step may give different results when traversing the input function from left-to-right and right-to-left. It might also introduce discontinuities in the resulting function. However, the optimization model within the dichotomy search maintains the continuity in the output function.

Remark 4.2. While algorithm 3 was designed to take the output of algorithm 2 as input, it can accommodate a wide range of PLQ functions, regardless of their initial

convexity. In fact, convexity is not a prerequisite for the input function, only the input function restricted to unbounded intervals has to satisfy the assumptions of proposition 3.5. As a consequence when the closest convex PLQ function has less pieces than the input function, algorithm 3 successfully computes it. When more pieces are needed to approximate the closest convex PLQ function, one has to use first algorithm 2 and then algorithm 3. □

Algorithm 3: Approximate convex PLQ function with minimal number of pieces

input : n, $\mathbf{C}(\rho)$, $\mathbf{P}(\rho)$, ε, δ
output : r, $\mathbf{C}(g)$, $\mathbf{P}(g)$, `error`

// `Merge similar pieces to reduce` n
`Preprocess` (ρ) ;
// `Dichotomy search`
left =0; right =n; error =$+\infty$;
while right > left **do**
  mid = left + Floor ((right-left)/2) ;
  [C(g),P(g),error] = set g_1, g_r and solve (3) ;
  if error < ε **then**
    right = mid ;
  else
    left = mid ;
  end
end

Considering our objective function (10), the optimization problem is mixed-integer polynomial (but not quadratic) that can be solved using BARON. Alternatively, we can introduce new variables to reformulate the problem as a quadratically-constrained quadratic program and solve it with Gurobi.

To approximate a PLQ function that is not necessarily convex, we can use the same algorithm except for removing the convexity constraint $g'_j(\chi_{j+1}) \leq g'_{j+1}(\chi_{j+1})$, and skipping any feasibility verification on unbounded intervals. The resulting problem is of a similar type, and is solved by MATLAB/YALMIP/BARON. We named this final algorithm, algorithm 4.

Remark 4.3. The constraint (14) namely $\mathbf{P}(g) \subset \mathbf{P}(\rho)$ may seem restrictive. If algorithm 3 or 4 is applied to the output of algorithm 2, it is not a limitation. However, in the general case, the constraints only provides an upper bound on the minimal number of pieces. □

5. Numerical experiments

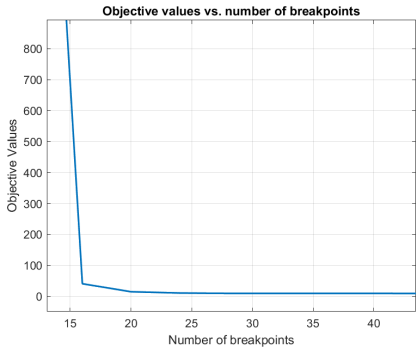
We now report numerical experiments run on algorithms 1, 2, 3, and 4. All the experiments are run on the 12^{th} Gen Intel(R) Core(TM) i7-12700H - 2.30 GHz, with 10 cores.

5.1. Algorithms 1 and 2

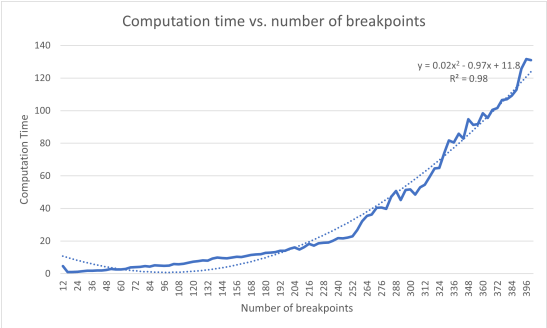
Consider the function

$$f(x) = \begin{cases} \frac{1}{2}x^2 + 1, & -\infty < x \leq 1, \\ 2x - \frac{1}{2}, & 1 < x \leq 2.5, \\ -x + 7, & 2.5 < x \leq 6, \\ x^2 - 5, & 6 < x < \infty. \end{cases} \tag{22}$$

We ran algorithm 1 multiple times while increasing the number of breakpoints within each piece. As shown in Figure 5(a), objective value kept decreasing till a certain number of breakpoints after which it remained constant. The computation time is plotted in Figure 5(b). Here computation time is the entire elapsed time to run the algorithm for a given number of output pieces - starting from problem formulation to calling the model with the solver on YALMIP. We ran the experiment thrice while clearing cache before each experiment and took the average computation time. The curve follows $O(n^2)$ increase (n represents the number of breakpoints in the output function) with $R^2 = 0.98$.



(a) Distance between function and its closest convex PLQ function with the same breakpoints. Note that the objective function value is constant after 20 pieces.



(b) Computation time increases quadratically.

Figure 5. Algorithm 1

We did similar profiling for algorithm 2 with the same input f; the computation time is shown in Figure 6. Again, we found the complexity of this algorithm grows in $O(n^2)$ with $R^2 = 0.99$. We include timings for 2 versions of BARON to emphasize that the quadratic behavior remains, but the coefficient of the quadratic curve improved with the more recent versions.

Considering that algorithms 3 and 4 are very similar, we only report numerical tests on algorithm 4.

5.2. Comparing with globally univariate spline approximation

We slightly modify the MIQCP Spline method from [31] and compare it with a slight modification of algorithm 4. We adapted the MIQCP Spline method to yield

piecewise quadratic splines instead of its original piecewise cubic splines, and we added a differentiability constraint at the breakpoints to algorithm 4 to ensure its output is a quadratic spline instead of a continuous piecewise quadratic polynomial, i.e., the output is C^1.

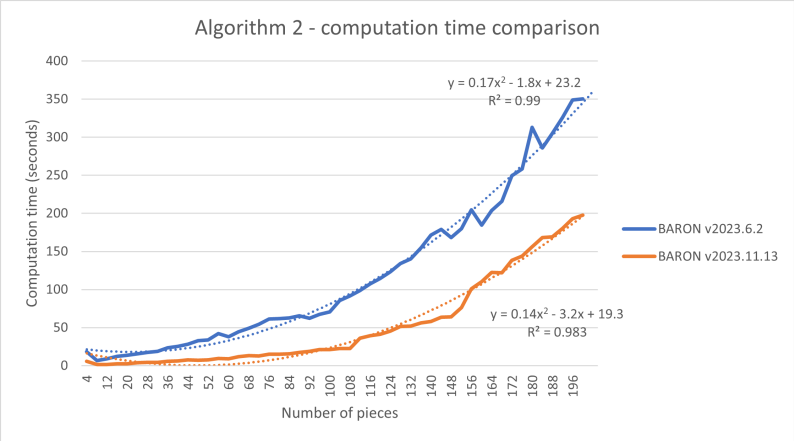


Figure 6. Algorithm 2 computation time increases quadratically. While a more recent BARON version performed better, it did not change the quadratic trend.

It is worth noting the differences between the 2 algorithms:

(i) Algorithm 4 requires $\mathbf{P}(g) \subset \mathbf{P}(\rho)$ to confine the breakpoints of the output function to be a subset of the input function's breakpoints; this constraint is not present in the MIQCP spline algorithm.

(ii) the MIQCP Spline algorithm is implemented in Python and calls Gurobi while algorithm 4 is implemented in MATLAB using YALMIP and calling BARON.

(iii) the MIQCP Spline algorithm minimizes the least-squares error between the input data points and the resultant function. On the other hand, algorithm 4 minimizes the area difference between the input and the output function using the 2-norm.

Both algorithms were executed on two distinct functions: a piecewise linear function

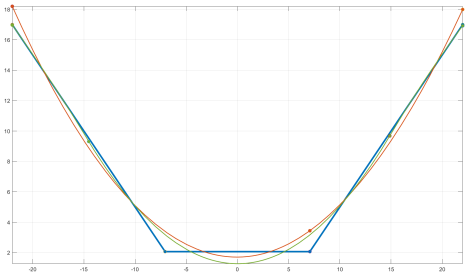
$$g(x) = \begin{cases} -x - 5, & -22 < x \leq -7.071, \\ 2.071, & -7.071 < x \leq 0.071, \\ x - 5, & 0.071 < x < 22, \end{cases}$$

and the closest convex function to the PLQ function (22).

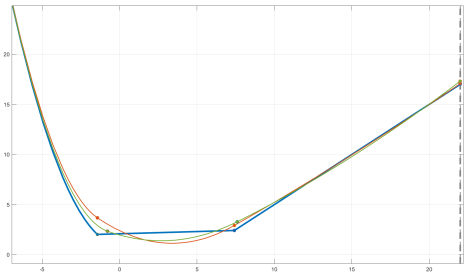
To generate data points for the MIQCP spline algorithm, we sampled 3 data points per piece. Both algorithms accept an identical number of predetermined breakpoints (called "knots" in [31]) and return a piecewise quadratic spline.

Figure 7(a) plots the quadratic spline approximations produced by the respective algorithms. The input piecewise linear function g in blue was partitioned into 36 segments. Algorithm 4 produces a quadratic spline with 3 pieces and a 2-norm error

of 11.6 with a computation time of 0.6 seconds. For the MIQCP Spline algorithm, a representation of the function was created by generating 3 data points for each piece, culminating in a set of 71 synthetic data points. This data is combined with a specification of 2 knots to obtain the quadratic spline in green, which has a 2-norm error of 7.7 and requires 4.2 seconds of computation.



(a) Approximating a piecewise linear function.



(b) Approximating a piecewise quadratic function.

Figure 7. Algorithm 4. Input function is shown in blue, algorithm 4 quadratic spline approximation in red, and MIQCP spline algorithm in green.

We also conducted experiments using the PLQ input function f, with the resulting quadratic spline approximations displayed in Figure 7(b). The input PLQ function in blue was segmented into 12 pieces. Providing this input to the adapted algorithm 4 yielded the quadratic spline depicted in red, accompanied by a 2-norm error of 12.9 and a computational duration of 2.7 seconds. To adapt the function for the MIQCP Spline algorithm, 23 synthetic data points were generated, with each piece represented by 3 data points. Utilizing this set of data points, along with a stipulation of 2 knots, the MIQCP Spline algorithm produced the quadratic spline represented in green, characterized by a 2-norm error of 6.1. The computation for this problem took 0.3 seconds.

The two algorithms return different output since they optimize different objective functions. In addition, the constraint $\mathbf{P}(g) \subset \mathbf{P}(\rho)$ makes algorithm 4 suboptimal.

Remark 5.1. As we experimented with increasing the number of pieces or breakpoints in the input function, we observed differences between solvers and between different versions of the same solver. A previous version of BARON prematurely terminated, deeming the model infeasible while the open-source solver BMIBNB for the identical model produces a solution albeit with a significantly longer computational duration (measured in hours) compared to BARON. The most recent version of BARON deemed the problem feasible. We also found discrepancies between a quadratically-constrained reformulation solved by Gurobi and BARON. Our experience suggests to use the very latest version of solvers, expect significant improvements between solver versions, and carefully validate any output. □

The MIQCP Spline algorithm excels in spline approximations, while algorithm 4 is specifically designed for optimizing PLQ function. Each algorithm is tailored for distinct objectives.

5.3. Application to road design

Designing a road involves solving three inter-related problems: drawing the horizontal alignment (a satellite view of the road), the vertical alignment (a quadratic spline drawn on a vertical plane whose x-axis is the distance from the start of the road, and the y-axis is the altitude), and the earthwork problem (indicate what material to move where to transform the ground into the vertical alignment). Our main interest is that the vertical alignment is a quadratic spline whose computation may result in a large number of pieces. This representation has an acute impact in the computation time of the road design. Consider that a road is divided into sections of length 10 to 100 meters. Each section may give rise to a piece in the vertical alignment and roads with hundreds of sections are common while roads of thousands of sections are becoming less rare due to the increase in computer performance. In addition, all computation depends on the precision of the ground measurements, which includes not only altitude but also estimations of various materials (rock, sand, dirts, etc.) volumes under the ground. Typically, optimization accounts for a 5% measurement error so there is no point in optimizing to 10 decimals.

Algorithm 4 is well suited to approximate vertical alignment curves. We only need to add the constraint

$$h'_j(\chi_{j+1}) = h'_{j+1}(\chi_{j+1})$$

to ensure that the road profile is smooth. This linear constraint has no impact on the nature of the optimization problem and little impact on the computation time.

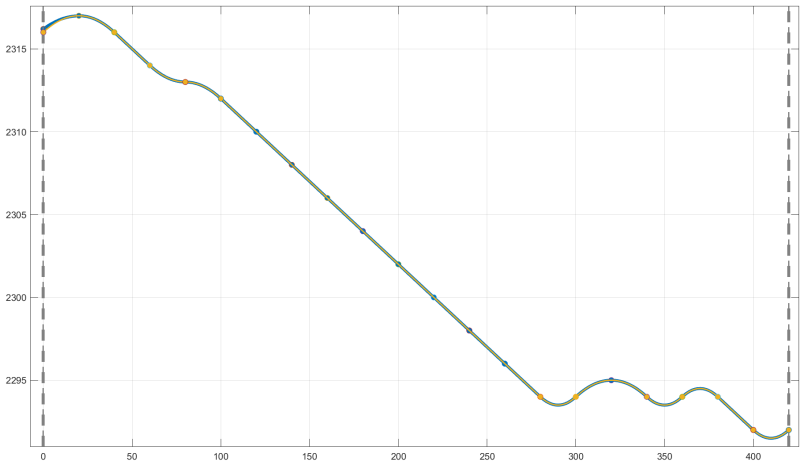


Figure 8. The input road spline f is in blue and the output approximate spline function h with minimal number of breakpoints in yellow. The breakpoints on each function are shown with circles in the same color. f has 22 breakpoints and h has 12 breakpoints.

The data for road design was provided by Softree Technical Systems Inc.[1] We apply algorithm 4 along with the new differentiability constraint to the vertical alignment road spline data to approximate it with the minimal number of breakpoints with an ε error tolerance. The results are visualized in Figure 8. The input quadratic spline f has 22 breakpoints compared to 12 breakpoints in the output quadratic spline h.

[1]http://www.softree.com

We use $\varepsilon = 0.2$ and $\delta = 0.000005$; the output spline is approximated at a distance of 0.1520 from the input spline with a computation time of 35.1 seconds (average of 3 runs) using the quadratically-constrained quadratic programming reformulation with the Gurobi solver.

6. Conclusion

We propose 4 algorithms. Two algorithms compute the closest convex PLQ function, and two approximate a PLQ function with a minimal number of pieces. We model optimization problems in MATLAB using YALMIP and use CPLEX, Gurobi or BARON to solve them. Numerical experiments validate the models, compare its performance to an optimal spline algorithm, and detail an application to road design optimization that greatly benefit from the reduction in the number of pieces.

More work is needed to understand differences in output between the various solvers. Some solvers sometime find a problem infeasible while other solvers can solve it. Resolving this discrepancy could expedite algorithm execution.

Since the optimization problems are polynomial, it would be interesting to substitute BARON with Gloptipoly [17], and measure any performance difference.

Preliminary results in reformulating our polynomial problem into a quadratically-constrained problem and solving it with Gurobi show potential for significant time reductions in addition to using a solver with a free academic license. More work is needed to systematically validate the results.

Our algorithms rely heavily on solving optimization models. Future research avenues might explore devising the closest convex function via heuristic strategies such as greedy algorithms or dynamic programming techniques. Future research could also explore parallelization techniques, e.g., to speed up the dichotomy search.

All the algorithms are specifically tailored for univariate PLQ functions. A logical progression would involve extending these algorithms to bivariate PLQ functions.

The complete code for all the algorithms is available on Github[2] under the GPL License.

References

[1] K. M. Ayman, W. Hare, Y. Lucet: *A multi-road quasi network flow model for the vertical alignment optimization of a road network*, Eng. Optimization 56/7 (2023) 1–24.

[2] H. H. Bauschke, Y. Lucet, H. M. Phan: *On the convexity of piecewise-defined functions*, ESAIM: Control, Optim. Calc. Var. 22 (2016) 728–742.

[3] H. H. Bauschke, W. M. Moursi: *An Introduction to Convexity, Optimization, and Algorithms*, MOS-SIAM Series on Optimization Vol. 34, Society for Industrial and Applied Mathematics, Philadelphia (2024).

[4] V. Beiranvand, W. Hare, Y. Lucet, S. Hossain: *Multi-haul quasi network flow model for vertical alignment optimization*, Eng. Optimization 49 (2017) 1777–1795.

[2]https://github.com/namrata-kundu/closest-convex-function-approximation

[5] Y. Bosch, P. Schäfer, J. Spoerhase, S. Storandt, J. Zink: *Consistent simplification of polyline tree bundles*, in: *Computing and Combinatorics*, C.-Y. Chen et al. (eds.), Springer, Cham (2021) 231–243.

[6] W. S. Chan, F. Chin: *Approximation of polygonal curves with minimum number of line segments*, in: *Algorithms and Computation*, T. Ibaraki et al. (eds.), Springer, Berlin (1992) 378–387.

[7] L. Contento, A. Ern, R. Vermiglio: *A linear-time approximate convex envelope algorithm using the double Legendre-Fenchel transform with application to phase separation*, Comp. Optimization Appl. 60 (2015) 231–261.

[8] D. H. Douglas, T. K. Peucker: *Algorithms for the reduction of the number of points required to represent a digitized line or its caricature*, Cartographica 10 (1973) 112–122.

[9] B. Gardiner, K. Jakee, Y. Lucet: *Computing the partial conjugate of convex piecewise linear-quadratic bivariate functions*, Comp. Optimization Appl. 58 (2014) 249–272.

[10] B. Gardiner, Y. Lucet: *Convex hull algorithms for piecewise linear-quadratic functions in computational convex analysis*, Set-Valued Var. Analysis 18 (2010) 467–482.

[11] B. Gardiner, Y. Lucet: *Graph-matrix calculus for computational convex analysis*, in: *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*, H. H. Bauschke et al. (eds.), Springer Optimization and Its Applications Vol. 49, Springer, New York (2011) 243–259.

[12] M. Godau: *A natural metric for curves – computing the distance for polygonal chains and approximation algorithms*, in: *Theoretical Aspects of Computer Science*, Proc. Symp. Hamburg 1991, Lecture Notes in Computer Science 480, Springer, Berlin (1991) 127–136.

[13] N. Goldberg, S. Rebennack, Y. Kim, V. Krasko, S. Leyffer: *MINLP formulations for continuous piecewise linear function fitting*, Comp. Optimization Appl. 79/1 (2021) 223–233.

[14] I. Goodfellow, Y. Bengio, A. Courville: *Deep Learning*, MIT Press (2016).

[15] Gurobi Optimizer Reference Manual (11.0): Gurobi Optimization LLC (2024).

[16] T. Haque, Y. Lucet: *A linear-time algorithm to compute the conjugate of convex piecewise linear-quadratic bivariate functions*, Comp. Optimization Appl. 70 (2018) 593–613.

[17] D. Henrion, J. B. Lasserre: *GloptiPoly: Global optimization over polynomials with MATLAB and SeDuMi*, ACM Trans. Math. Software 29 (2003) 165–194.

[18] J.-B. Hiriart-Urruty, Y. Lucet: *Parametric computation of the Legendre-Fenchel conjugate*, J. Convex Analysis 14 (2007) 657–666.

[19] J. Hull: *Options, Futures and Other Derivatives*, Pearson, London (2002).

[20] A. Iannantuono, W. Hare, Y. Lucet: *Optimization with regularization to create sensible vertical alignments in road design*, Decision Analytics J. 6 (2023), art.no. 100183.

[21] H. Imai, M. Iri: *Polygonal approximations of a curve – formulations and algorithms*, Machine Intell. Pattern Recog. 6 (1988) 71–86.

[22] K. Kazda, X. Li: *A linear programming approach to difference-of-convex piecewise linear approximation*, Eur. J. Oper. Res. 312 (2024) 493–511.

[23] H. Konno, T. Kuno: *Best piecewise constant approximation of a function of single variable*, Oper. Res. Letters 7 (1988) 205–210.

[24] J. Lofberg: *YALMIP: a toolbox for modeling and optimization in MATLAB*, in: Proc. IEEE Int. Conf. Robotics Automation, Taipei (2004) 284–289.

[25] Y. Lucet: *Faster than the fast Legendre transform, the linear-time Legendre transform*, Numer. Algorithms 16 (1997) 171–185.

[26] Y. Lucet: *Fast Moreau envelope computation. I: Numerical algorithms*, Numer. Algorithms 43 (2006) 235–249.

[27] Y. Lucet: *What shape is your conjugate? A survey of computational convex analysis and its applications*, SIAM Review 52 (2010) 505–542.

[28] Y. Lucet, H. H. Bauschke, M. Trienis: *The piecewise linear-quadratic model for computational convex analysis*, Comp. Optimization Appl. 43 (2009) 95–118.

[29] S. G. Mallat, E. A. A. Books: *A Wavelet Tour of Signal Processing: the Sparse Way*, 3rd ed., Elsevier / Academic Press, Amsterdam (2009).

[30] H. Markowitz: *Portfolio selection*, J. Finance 7 (1952) 77–91.

[31] R. Mohr, M. Coblenz, P. Kirst: *Globally optimal univariate spline approximations*, Comp. Optimization Appl. 85/2 (2023) 409–439.

[32] N. S. Momo, W. Hare, Y. Lucet: *Modeling side slopes in vertical alignment resource road construction using convex optimization*, Comp.-Aided Civil Infrastructure Eng. 38 (2023) 211–224.

[33] S. Nickel, C. Steinhardt, H. Schlenker, W. Burkart, M. Reuter-Oppermann: *IBM ILOG CPLEX Optimization Studio V12.10.0*, IBM (2020) 9–23.

[34] N. Nisan, T. Roughgarden, E. Tardos, V. Vazirani: *Algorithmic Game Theory*, Cambridge University Press, Cambridge (2007).

[35] U. Ramer: *An iterative procedure for the polygonal approximation of plane curves*, Comp. Graphics Image Proc. 1 (1972) 244–256.

[36] J. B. Rawlings, D. Q. Mayne, M. M. Diehl: *Model Predictive Control: Theory and Design*, Nob Hill Publishing, Madison (2009).

[37] S. Rebennack, J. Kallrath: *Continuous piecewise linear delta-approximations for univariate functions: Computing minimal breakpoint systems*, J. Optimization Theory Appl. 167 (2014) 617–643.

[38] K. Reumann, A. P. M. Witkam: *Optimizing curve segmentation in computer graphics*, in: Proc. Int. Computing Symposium, Davos 1973, North-Holland, Amsterdam (1974) 467–472.

[39] R. T. Rockafellar, R. J.-B. Wets: *Variational Analysis*, Springer, Berlin (1998).

[40] N. V. Sahinidis: *BARON Branch and Reduce Optimization Navigator*, User's Manual Version 4.0, June 2000.

[41] S. Shalev-Shwartz, S. Ben-David: *Understanding Machine Learning: From Theory to Algorithms*, Cambridge University Press, Cambridge (2013).

[42] S. Singh, Y. Lucet: *Linear-time convexity test for low-order piecewise polynomials*, SIAM J. Optimization 31 (2021) 972–990.

[43] J. Spoerhase, S. Storandt, J. Zink: *Simplification of polyline bundles*, in: Proc. 17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT'20), S. Albers (ed.), LIPIcs Vol. 162, Leibniz-Zentrum für Informatik (2020) 35:1–35:20.

[44] The MathWorks Inc.: *MATLAB*, version: 9.13.0 (R2022b) (2022).

[45] M. Visvalingam, D. Whyatt: *The Douglas-Peucker algorithm for line simplification: Re-evaluation through visualization*, Computer Graphics Forum 9 (2007) 213–225.

[46] J. A. Warwicker, S. Rebennack: *A comparison of two mixed-integer linear programs for piecewise linear function fitting*, INFORMS J. Computing 34 (2021) 1042–1047.