




Piecewise Linearization of Bivariate Nonlinear Functions: Minimizing the Number of Pieces Under a Bounded Approximation Error

Aloïs Duguet^(✉)  and Sandra Ulrich Ngueveu

LAAS-CNRS, Université de Toulouse, CNRS, INP, Toulouse, France
 alois.duguet@laas.fr , ngueveu@laas.fr

Abstract. This work focuses on the approximation of bivariate functions into piecewise linear ones with a minimal number of pieces and under a bounded approximation error. Applications include the approximation of mixed integer nonlinear optimization problems into mixed integer linear ones that are in general easier to solve. A framework to build dedicated linearization algorithms is introduced, and a comparison to the state of the art heuristics shows their efficiency.

Keywords: Piecewise linear approximation · Bivariate nonlinear functions · Mixed integer nonlinear programming · Heuristics

1 Problem Description and State of the Art

Let (\mathcal{P}) be the optimization problem of approximating a nonlinear function f of two variables by a piecewise linear (PWL) function g subject to approximation error constraints on domain \mathbb{D} represented by functions l and u satisfying $l(x, y) \leq f(x, y) \leq u(x, y)$ for all $(x, y) \in \mathbb{D}$:

$$(\mathcal{P}) \quad \begin{cases} \min & n & (1) \\ \text{subject to} & l(x, y) \leq g(x, y) \leq u(x, y) \quad \forall (x, y) \in \mathbb{D} \subset \mathbb{R}^2 & (2) \\ & g \text{ is a PWL function with } n \text{ pieces} & (3) \end{cases}$$

Constraints (2) are pointwise approximation constraints, which gives an infinite number of constraints because \mathbb{D} is a continuous domain. Piecewise linear approximation are more commonly minimizing an approximation error with a fixed number of pieces [9], but our objective is to minimize the number of pieces of g so that an MILP formulation of g introduces less binary variables [10]. It can be useful for the approximation of a Mixed Integer Nonlinear Programming problem (MINLP) with a Mixed Integer Linear Programming (MILP) by substituting each nonlinear functions by a PWL one. Moreover, it was shown in [4]

that in some cases, MINLP can be solved by applying only techniques from MILP.

Heuristics and exact methods exist to approximate univariate nonlinear functions ($\mathbb{D} \subset \mathbb{R}$) [1, 6, 8]. We are interested in bivariate functions ($\mathbb{D} \subset \mathbb{R}^2$) and to the best of our knowledge, only two papers address this case, with heuristics only:

- The authors of [7] propose two heuristics to solve problem (\mathcal{P}) with continuous PWL functions. The first heuristic is based on an iterative subdivision of the domain \mathbb{D} into triangles (2-simplexes) until for each subdomain a linear function satisfying the approximation error has been found. The verification that a given linear function fits the subdomain is made by solving a nonlinear programming problem (NLP). The second heuristic can be used if the contribution of the two variables in the function can be separated in two univariate functions (linearly or nonlinearly). In this case, an algorithm finds the two optimal continuous univariate PWL functions and combine them to build a single two-variable PWL function.
- In [5] an iterative process attempts to find a continuous PWL function written as a Difference of Convex Continuous PWL functions (DC CPWL) that satisfies the approximation error. The idea is to iteratively solve an MILP relaxation of (\mathcal{P}) and then to find lazy constraints to add to the relaxation until a solution found is feasible for (\mathcal{P}). The relaxation consists in replacing the infinite number of constraints (2) with a finite number of them.

After the introduction of definitions used throughout the paper in Sect. 2, the three key ideas of a framework for piecewise linearization are detailed in Sect. 3. It is followed by explanations on the instantiation of crucial parts of this framework to create different heuristics in Sect. 4, and finally, numerical experiments comparing the state of the art to our best heuristics are shown in Sect. 5.

2 Definitions

The vocabulary used throughout this work is presented below. They are in part extensions to $\mathbb{D} \subset \mathbb{R}^2$ of definitions from [1] for $\mathbb{D} \subset \mathbb{R}$.

Definition 1 (Polytope). *A polytope \mathcal{P} is the convex hull of some points $X_i \in \mathbb{R}^m$: $\mathcal{P} = \{x \in \mathbb{R}^m, x = \sum_i \lambda_i X_i, \sum_i \lambda_i = 1, \lambda_i \geq 0 \forall i\}$.*

Throughout the paper, a *piece* will refer to a polytope that composes the graph of a PWL function, and $\llbracket 1, n \rrbracket := \{1, \dots, n\}$.

Definition 2 (PWL function). *Let \mathbb{D} be a compact set of \mathbb{R}^2 . A function $g : \mathbb{D} \mapsto \mathbb{R}$ is a PWL function with n pieces if and only if there exists $\{a_i\}_{i \in \llbracket 1, n \rrbracket} \subset \mathbb{R}^2$, $\{b_i\}_{i \in \llbracket 1, n \rrbracket} \subset \mathbb{R}$ and a family of polytopes $\{D_i\}_{i \in \llbracket 1, n \rrbracket} \subset \mathbb{R}^2$ such that $\mathbb{D} = \cup_{i \in \llbracket 1, n \rrbracket} D_i$, and for $i \neq j$ the polytopes D_i and D_j can only intersect on their boundary and g is defined by $g(x, y) = \min\{a_i \cdot (x, y)^T + b_i \mid (x, y) \in D_i \forall i \in \llbracket 1, n \rrbracket\}$, with \cdot denoting the standard scalar product.*

Precautions were taken to allow g to be *not necessarily continuous* at the boundary of a polytope \mathbb{D}_i . To prevent $g(x, y)$ to have multiple definitions because $\{D_i\}_{i \in \llbracket 1, n \rrbracket}$ can intersect on their boundary, $g(x, y)$ is chosen as the minimum over all possible definitions, so that it is a lower semicontinuous function.

Definition 3 (Corridor). Let \mathbb{D} be a compact set of \mathbb{R}^2 . Let $u, l : \mathbb{D} \mapsto \mathbb{R}$ be two continuous functions verifying $u(x, y) > l(x, y), \forall (x, y) \in \mathbb{D}$. Define the set $\mathcal{C} = \{(x, y, z) \in \mathbb{R}^3 | (x, y) \in \mathbb{D}, l(x, y) \leq z \leq u(x, y)\}$ as the corridor between u and l . If $\mathbb{D} \subset \mathbb{R}^2$, we call the area of \mathbb{D} the domain area of \mathcal{C} .

A similar definition can be made for \mathbb{D} an interval $[a, b]$, in which case we call $b - a$ the domain length of \mathcal{C} .

Definition 4 (Corridor domain). Let \mathcal{C} be a corridor, $\mathcal{C} \subset \mathbb{R}^3$. The domain of corridor \mathcal{C} noted $D(\mathcal{C})$ is the projection of \mathcal{C} on its two first coordinates, which is also the domain on which u and l need to be defined.

Definition 5 (Piece within a corridor). A polytope $\mathcal{P} \subset \mathbb{R}^3$ is within a corridor \mathcal{C} if and only if there exists a linear function $g : \mathbb{D} \subset D(\mathcal{C})$, such that $\mathcal{P} = \{(x, y, g(x, y)), (x, y) \in \mathbb{D}\}$ and $\mathcal{P} \subset \mathcal{C}$.

Definition 6 (Fitting). A PWL function g fits a corridor \mathcal{C} if and only if the pieces of g (polytopes $\{P\}_{i \in \llbracket 1, n \rrbracket}$ of the graph of g) are within \mathcal{C} and g is defined on the entire domain $D(\mathcal{C})$.

Definition 7 (PWL corridor). A corridor \mathcal{C} is called a PWL corridor if and only if u and l defining \mathcal{C} are both PWL functions.

Definition 8 (Inner corridor). Let \mathcal{C}_0 be a corridor between u_0 and l_0 . Let \mathcal{C} be a corridor between u and l . We call \mathcal{C} an inner corridor of \mathcal{C}_0 if and only if $D(\mathcal{C}) = D(\mathcal{C}_0)$ and $l_0(x, y) \leq l(x, y) < u(x, y) \leq u_0(x, y)$.

Definition 9 (\mathbb{R}^2 -corridor fitting problem). The \mathbb{R}^2 -corridor fitting problem consists in finding a PWL function g of two variables fitting a corridor \mathcal{C} such that its number of pieces is minimized.

(\mathcal{P}) is equivalent to an \mathbb{R}^2 -corridor fitting problem with corridor \mathcal{C} between u and l . Moreover, by extending the previous definitions to dimension $m \geq 1$, it is possible to define the \mathbb{R}^m -corridor fitting problem that refers to the problem with $D(\mathcal{C}) \subset \mathbb{R}^m$.

Definition 10 (Truncated-corridor). Let $\mathcal{C}_1, \mathcal{C}_2 \in \mathbb{R}^2$ be two corridors both defined by functions $u, l : \mathbb{R} \mapsto \mathbb{R}$ on the intervals $[a, b_1]$ and $[a, b_2]$. We call \mathcal{C}_2 a truncated-corridor of \mathcal{C}_1 if and only if $[a, b_2] \subset [a, b_1]$ ($b_2 \leq b_1$).

Definition 11 (Maximal linear segment). A maximal linear segment in a corridor \mathcal{C} is a linear segment within \mathcal{C} that induces a truncated-corridor of maximal domain length.

Definition 12 (Truncated-corridor in direction d). Let \mathcal{C}_1 and \mathcal{C}_2 be two corridors defined by the same functions u and l with compact corridor domains in \mathbb{R}^2 . Let $d \in \mathbb{R}^2 \setminus \{0\}$. We call \mathcal{C}_2 a truncated-corridor of \mathcal{C}_1 in direction d if and only if there exists $\sigma \in \mathbb{R}$ for which $D(\mathcal{C}_2) = D(\mathcal{C}_1) \cap \{(x, y) \in \mathbb{R}^2, (x, y) \cdot d \leq \sigma\}$, i.e. $D(\mathcal{C}_2)$ is the intersection of $D(\mathcal{C}_1)$ with a half-plane.

Definition 13 (Maximal piece in direction d). A maximal piece in direction $d \in \mathbb{R}^2 \setminus \{0\}$ of a corridor \mathcal{C} is a polytope within \mathcal{C} that induces a truncated-corridor of \mathcal{C} in direction d that is of maximal domain area.

3 A Framework for Solving the \mathbb{R}^2 -Corridor Fitting Problem

We present in this section a framework to create efficient algorithms for the \mathbb{R}^2 -corridor fitting problem. The instantiation chosen for different parts of the framework are described in Sect. 4 as well as some details on the implementation.

Three key ideas are followed in the framework. The first two are used to avoid drawbacks encountered in [7], whereas the third one is meant to render a subproblem more tractable:

- **Key idea 1:** Pieces should be chosen among general convex polygons instead of triangles to constrain less the pieces chosen, possibly decreasing the number of pieces
- **Key idea 2:** Choose pieces that are good (ideally optimal) solutions of a maximal piece in direction d problem, to aim for a domain covered by a piece that is “as large as possible” because in [7] the size of triangles is fixed which increases the number of pieces necessary
- **Key idea 3:** Compute a good feasible solution of the maximal piece in direction d problem with a series of LP problems obtained after substituting \mathcal{C} with a PWL inner corridor of \mathcal{C}

The remainder of this section builds upon these principles.

3.1 Key Idea 1: Management of the Corridor Domain

The corridor domain should be tiled with shapes as general as possible provided that they can be formulated in an MILP. Such shapes are polygons, but we further restrict those shapes to convex polygons because formulating a non-convex polygon in an MILP introduces additional binary variables. It is expected that allowing convex polygons instead of only triangles as done in [7] will lead to a lower number of pieces.

The procedure that manages pieces and the remaining corridor domain is described in Algorithm 1. At each iteration, one piece is computed for each vertex of $D(\mathcal{C})$ by function `compute_piece`, and a function `score` selects the “most” suitable to obtain a PWL function with few pieces, see Sect. 4.1. Function `update_domain(\mathcal{C}, p)` removes the part of $D(\mathcal{C})$ on which p is defined. This function also divides the new reduced corridor \mathcal{C} in two corridors if polygon $D(\mathcal{C})$ only has angles at the vertices greater than 90 to avoid a bad behaviour.

Algorithm 1. Finding a PWL function fitting a corridor \mathcal{C} with a low number of pieces

```

1: function PWL_2D_FITTING( $\mathcal{C}$ )
2:    $\mathcal{P} \leftarrow \emptyset$  ▷ list of chosen pieces
3:    $\mathcal{Q} \leftarrow \{\mathcal{C}\}$  ▷ list of corridors with convex domains not already tiled
4:   while  $\mathcal{Q} \neq \emptyset$  do
5:      $\mathcal{C} = \text{pop}(\mathcal{Q})$ 
6:      $\text{candidate\_pieces} \leftarrow \emptyset$ 
7:     for  $v$  vertex of  $D(\mathcal{C})$  do
8:        $d \leftarrow \text{choose\_progress\_direction}(\mathcal{C}, v)$ 
9:        $\text{candidate\_pieces} \leftarrow \text{candidate\_pieces} \cup \{\text{compute\_piece}(\mathcal{C}, v, d)\}$ 
10:    end for
11:     $p \leftarrow \text{argmax}_{p \in \text{candidate\_pieces}} \text{score}(p)$ 
12:     $\mathcal{P} \leftarrow \mathcal{P} \cup \{p\}$ 
13:     $\mathcal{Q} = \text{update\_domain}(\mathcal{C}, p)$ 
14:  end while
15:  return  $\mathcal{P}$ 
16: end function

```

3.2 Key Idea 2: The Maximal Piece in Direction d Problem

We chose to find a new piece by covering an area starting from point v and extending as far as possible in direction d . This direction d points to the interior of $D(\mathcal{C})$ when starting from v and is computed via function *choose_progress_direction* of Algorithm 1. The hypothesis of starting from a vertex instead of any point of the border of the polygon $D(\mathcal{C})$ is made. Computing the piece consists in solving a maximal piece in direction d problem (MP_d):

$$(MP_d) \begin{cases} \max \sigma \\ \text{s.t. } \alpha x + \beta y + \gamma \in \mathcal{C}_\sigma^d \quad \forall (x, y) \in D(\mathcal{C}_\sigma^d) \\ \alpha, \beta, \gamma, \sigma \in \mathbb{R} \end{cases} \quad (4)$$

where $D(\mathcal{C}_\sigma^d) = D(\mathcal{C}) \cap \{(x, y) \in \mathbb{R}^2 \mid (x, y)^T \cdot d \leq \sigma\}$ is the domain of \mathcal{C}_σ^d , the truncated-corridor of \mathcal{C} in direction d . (MP_d) is a generalized semi-infinite programming problem (GSIP). Indeed, the number of pointwise constraints is infinite and depends on variable σ , while the number of variables is 4: a real variable σ for the half-plane intersection as well as 3 real variables (α, β, γ) to describe the linear function $g(x, y) = \alpha x + \beta y + \gamma$.

3.3 Key Idea 3: Computing a Feasible Solution of a Maximal Piece in Direction d Problem

As we want a computationally cheap solution to the GSIP (MP_d) because of the high number of times such a problem has to be solved, a feasible solution of (MP_d) is computed via a series of LP problems, as described below, using a PWL inner corridor of the original corridor \mathcal{C} , because it allows to replace the

infinite number of nonlinear constraints by a finite number of linear constraints while ensuring feasibility.

Let corridor $\mathcal{C}_{PWL\sigma}^d$ be a PWL inner corridor of \mathcal{C}_σ^d with associated functions \tilde{u} and \tilde{l} for readability; note $(D_{\tilde{u}}^i)_{i \in I}$ and $(D_{\tilde{l}}^j)_{j \in J}$ the subdomains of corridor $\mathcal{C}_{PWL\sigma}^d$ on which \tilde{u} and \tilde{l} are linear, indexed by I and J respectively. Then, (MP_d) is a relaxation of the following problem because \mathcal{C} is replaced by an inner corridor.

$$(MP'_d) \begin{cases} \max \sigma \\ s.t. \\ \alpha x + \beta y + \gamma - \tilde{u}_i(x, y) \leq 0 \quad \forall (x, y) \in D(\mathcal{C}_\sigma^d) \cap D_{\tilde{u}}^i, \forall i \in I \\ \alpha x + \beta y + \gamma - \tilde{l}_j(x, y) \geq 0 \quad \forall (x, y) \in D(\mathcal{C}_\sigma^d) \cap D_{\tilde{l}}^j, \forall j \in J \\ \alpha, \beta, \gamma, \sigma \in \mathbb{R} \end{cases} \quad (5)$$

Note that on each $D_{\tilde{u}}^i$, $g(x, y) - \tilde{u}(x, y)$ is a linear function, thus it suffices to check $g(X) - \tilde{u}(x, y) \leq 0$ for each vertex of convex polygonal domain $D_{\tilde{u}}^i$ to ensure constraint $g(x, y) - \tilde{u}(x, y) \leq 0$ on $D_{\tilde{u}}^i$. A similar reasoning leads to the same result for constraints involving \tilde{l} . Thus, the following problem is equivalent to (MP'_d) but has the advantage of having only a finite number of linear constraints.

$$(MP''_d) \begin{cases} \max \sigma \\ s.t. \\ \alpha x + \beta y + \gamma - \tilde{u}_i(x, y) \leq 0 \quad \text{for each vertex } (x, y) \text{ of } D(\mathcal{C}_\sigma^d) \cap D_{\tilde{u}}^i, \forall i \in I \\ \alpha x + \beta y + \gamma - \tilde{l}_j(x, y) \geq 0 \quad \text{for each vertex } (x, y) \text{ of } D(\mathcal{C}_\sigma^d) \cap D_{\tilde{l}}^j, \forall j \in J \\ \alpha, \beta, \gamma, \sigma \in \mathbb{R} \end{cases} \quad (6)$$

Finally, problem (MP''_d) has constraints involving polygon intersections depending nonlinearly on variable σ , thus it is not an LP problem. Parameterizing (MP''_d) with σ , the following LP feasibility problem is obtained which can be repeatedly solved until a satisfactory σ value has been found.

$$(MP''_{d,\sigma}) \begin{cases} \alpha x + \beta y + \gamma - \tilde{u}_i(x, y) \leq 0 \quad \text{for each vertex } (x, y) \text{ of } D(\mathcal{C}_\sigma^d) \cap D_{\tilde{u}}^i, \forall i \in I \\ \alpha x + \beta y + \gamma - \tilde{l}_j(x, y) \geq 0 \quad \text{for each vertex } (x, y) \text{ of } D(\mathcal{C}_\sigma^d) \cap D_{\tilde{l}}^j, \forall j \in J \\ \alpha, \beta, \gamma \in \mathbb{R} \end{cases} \quad (7)$$

4 Framework Key Points Instantiation

In this section, choices made on key points of the framework of Sect. 3 are described: the function *score* of Algorithm 1 in Sect. 4.1, the choice of a direction d in Sect. 4.2 and the computation of a PWL inner corridor in Sect. 4.3. \mathcal{C} refer to a corridor between u and l in the remainder of the section.

4.1 Scoring the Quality of Pieces

In Algorithm 1, a function *score* ranks the quality of candidate pieces and the piece with highest score is kept. Two scoring functions have been implemented:

- *Area* measures the area of the domain covered by piece p
- *Partial Derivatives Total Variation* (PaD) approximates the sum of the *total variation* of each partial derivative of u and l on the domain covered by p

The total variation of a function g is a measure of how much that function varies on its domain \mathcal{D} . The total variation of g on \mathcal{D} is equal to $\int_{\mathcal{D}} \|\nabla g(x)\|_2 dx$. The interest of *PaD* needs the introduction of the *pointwise height* of a corridor.

Definition 14 (pointwise height). We call $\mathcal{C}_{PH}(x, y) = u(x, y) - l(x, y)$ the *pointwise height of corridor \mathcal{C} at point (x, y)* .

Remark 1. The most commonly used type of approximation error for a function f is the absolute error. It induces a corridor \mathcal{C} such that $l(x, y) = f(x, y) - \delta$ and $u(x, y) = f(x, y) + \delta$ with $\delta > 0$, that has constant pointwise height.

Area is a straightforward and simple idea to evaluate the piece quality, it will serve as a reference to evaluate other scoring functions. PaD is thought to be an adaptation for two-variable functions of Theorem 1 of [3]. Indeed, it states that for a corridor \mathcal{C} of constant pointwise height 2δ with $D(\mathcal{C}) = [a, b]$, when $\delta \rightarrow 0$, the minimum number of pieces $s(\delta)$ satisfies the asymptotic approximation $s(\delta) \sim \frac{1}{4\delta} \int_a^b \sqrt{|u''(x, y)|} dx$. Note first that $u''(x, y) = l''(x, y)$ because it is a corridor with constant pointwise height, and second that the integral computed is the total variation of function u' (and l'). It is thus expected that PaD performs better than Area for small values of pointwise height. For large values, the total variation should be less relevant to estimate the difficulty of fitting a piece, thus PaD could be less efficient.

4.2 Choose a Progress Direction

Line 8 of Algorithm 1 selects progress direction d knowing starting vertex v . Two options were tested for this choice.

- the direction going along the bisector of the two edges of $D(\mathcal{C})$ having v as starting point, denoted *bd* for Bisector Direction
- Compute two maximal linear segments starting from v and following each edge of $D(\mathcal{C})$ having v as endpoint. The direction orthogonal to the line joining the two ends of the maximal linear segments is chosen as the progress direction d . It is denoted *med* for Mean progress along Edges Direction

The first is a naive option, while the second is meant to take into account the “difficulty” of progressing along the two extremal directions given by the two edges starting at v .

4.3 Inner Approximation of a Corridor

In the function *compute_piece*, the computation of a PWL inner corridor \mathcal{C}_{PWL} of \mathcal{C} boils down to computing PWL functions \tilde{u} and \tilde{l} verifying $l(x, y) \leq \tilde{l}(x, y) \leq \tilde{u}(x, y) \leq u(x, y)$, for all $(x, y) \in D(\mathcal{C})$.

To compute \tilde{u} (a similar method works for \tilde{l}), a basic idea is to divide $D(\mathcal{C})$ into rectangular pieces of same sizes, and then to compute a third coordinate $\tilde{u}(x, y)$ to each vertex $v = (x, y)$ of each rectangular piece such that $\tilde{u}(x, y) \leq u(x, y)$. *Interval analysis* on the gradient of u suffices to compute the values of $\tilde{u}(x, y)$ such that \tilde{u} is an underestimation of u , as explained in Proposition 1.

Proposition 1. *Let $u \in C^1$ defined on $\mathcal{D} = [a, b] \times [c, d] \subset \mathbb{R}^2$. Let $l_x = b - a$ and $l_y = d - c$. Let ∇u be the gradient of u . Let $[D_x^{low}, D_x^{high}] \times [D_y^{low}, D_y^{high}]$ be such that $\nabla u(x, y) \in [D_x^{low}, D_x^{high}] \times [D_y^{low}, D_y^{high}]$ for all $(x, y) \in \mathcal{D}$. Let $(M_x, M_y) = (\frac{a+b}{2}, \frac{c+d}{2})$. Define:*

$$u_{(a,c)}^- := u(M_x, M_y) - D_x^{high} \cdot l_x - D_y^{high} \cdot l_y \quad (8)$$

$$u_{(b,c)}^- := u(M_x, M_y) + D_x^{low} \cdot l_x - D_y^{high} \cdot l_y \quad (9)$$

$$u_{(b,d)}^- := u(M_x, M_y) + D_x^{low} \cdot l_x + D_y^{low} \cdot l_y \quad (10)$$

$$u_{(a,d)}^- := u(M_x, M_y) - D_x^{high} \cdot l_x + D_y^{low} \cdot l_y \quad (11)$$

If a linear function f satisfies:

$$f(a, c) \leq u_{(a,c)}^-, f(b, c) \leq u_{(b,c)}^-, f(b, d) \leq u_{(b,d)}^- \text{ and } f(a, d) \leq u_{(a,d)}^- \quad (12)$$

Then $f(x, y) \leq u(x, y)$ for all $(x, y) \in \mathcal{D}$.

Proof. Let f be a linear function satisfying the 4 inequalities (12). Let $M = (M_x, M_y, f(M_x, M_y))$ be the point on the surface defined by u corresponding to the middle of \mathcal{D} . Let $(x_0, y_0) \in \mathcal{D}$. We have:

$$\begin{aligned} u(M_x, M_y) - D_x^{high} \cdot (M_x - x_0) - D_y^{high} \cdot (M_y - y_0) &\leq u(x_0, y_0) && \text{if } x_0 \leq M_x, y_0 \leq M_y \\ u(M_x, M_y) + D_x^{low} \cdot (x_0 - M_x) - D_y^{high} \cdot (M_y - y_0) &\leq u(x_0, y_0) && \text{if } x_0 \geq M_x, y_0 \leq M_y \\ u(M_x, M_y) + D_x^{low} \cdot (x_0 - M_x) + D_y^{low} \cdot (y_0 - M_y) &\leq u(x_0, y_0) && \text{if } x_0 \geq M_x, y_0 \geq M_y \\ u(M_x, M_y) - D_x^{high} \cdot (M_x - x_0) + D_y^{low} \cdot (y_0 - M_y) &\leq u(x_0, y_0) && \text{if } x_0 \leq M_x, y_0 \geq M_y \end{aligned}$$

because $[D_x^{low}, D_x^{high}] \times [D_y^{low}, D_y^{high}]$ are bounds of ∇u on domain \mathcal{D} . Now, define a PWL function f_{PWL} with four rectangle pieces with vertices positionned at $\{(a, c), (b, c), (b, d), (a, d), (\frac{a+b}{2}, \frac{c+d}{2})\}$ and height the left-hand sides of (12) as well as $u(M_x, M_y)$ respectively. In particular, $f_{PWL} \leq u$ on \mathcal{D} . In addition, direct computations show that $f(x, y) \leq f_{PWL}(x, y)$ for $(x, y) \in \{(a, c), (b, c), (b, d), (a, d), (\frac{a+b}{2}, \frac{c+d}{2})\}$. Finally, as f and f_{PWL} are linear on the four pieces domain of f_{PWL} , we have $f \leq f_{PWL} \leq u$ on \mathcal{D} . \square

To build a PWL inner corridor exploiting Proposition 1 in our algorithm, a method called *efficiency refinement* is used. It is described after the introduction of the bounding efficiency η .

Definition 15 (bounding efficiency η). *Let \mathcal{C} be a corridor with $D(\mathcal{C})$ a polygonal domain of \mathbb{R}^2 . Let \mathcal{C}_{PWL} be a PWL inner corridor of \mathcal{C} . We say that \mathcal{C}_{PWL} achieves a bounding efficiency for \mathcal{C} of $\eta \in [0, 1]$ if the pointwise height (PH) ratio $\frac{(\mathcal{C}_{PWL})_{PH}(x,y)}{\mathcal{C}_{PH}(x,y)}$ is greater or equal to η for each $(x, y) \in D(\mathcal{C})$.*

Proposition 2. *Let \mathcal{C} be a corridor with $D(\mathcal{C})$ a polygonal domain of \mathbb{R}^2 and let $\eta \in [0, 1]$. If \mathcal{C} has constant pointwise height, then a PWL inner corridor \mathcal{C}_{PWL} has a bounding efficiency for \mathcal{C} of η if the pointwise height ratio $\frac{(\mathcal{C}_{PWL})_{PH}(x,y)}{\mathcal{C}_{PH}(x,y)}$ is greater or equal to η for each (x, y) vertex of a piece domain of \tilde{u} or l .*

Proof. \mathcal{C} has constant pointwise height. Thus for each piece of \mathcal{C}_{PWL} , the minimum pointwise height ratio is on an extreme point of the piece domain, that is to say on a vertex of the piece domain, which is lower bounded by η by hypothesis. \square

The efficiency refinement procedure builds a PWL inner corridor \mathcal{C}_{PWL} of \mathcal{C} achieving a bounding efficiency of η if it has a constant pointwise height, but without this property, it only checks that the pointwise height ratio at the vertices of each piece is η .

After having found a rectangle \mathcal{D} containing $D(\mathcal{C})$, it creates an initial PWL corridor likely invalid ($l \not\leq u$) with only one rectangular piece on \mathcal{D} , and then iteratively refines the pieces that do not satisfy a bounding efficiency of η at the 4 vertices into 4 new pieces until each piece satisfies the efficiency, which implies the validity of the PWL inner corridor as well.

Parameter η needs to be adjusted depending on the quality of PWL inner corridor \mathcal{C}_{PWL} wanted. To produce a really good approximation of corridor \mathcal{C} , η near 1 shall be used, but the number of pieces forming \mathcal{C}_{PWL} increases consequently, thus increasing the computation time of $(MP''_{d,\beta})$ to be solved later on.

5 Numerical Experiments

The performance of our framework is compared to the state of the art. Our best heuristic (found with experiments in [2]) called DN99 uses scoring function PaD, starting vertex set to med and $\eta = 0.99$. Also, to illustrate the effects of parameter η , results of a heuristic which uses the same parameters as DN99 but with $\eta = 0.95$, are shown. This second heuristic is called DN95.

Heuristics from the state of the art are described in articles [1, 5, 7]. Recall that [7] proposes two heuristics: one based on triangulation (RK2D), and one based on a decomposition of the bivariate function into a sum of one variable functions to approximate separately (RK1D). Heuristic LinA2D is based on the same decomposition as RK1D, but uses library LinA from [1] to compute the approximation of univariate functions. Obviously LinA2D outperforms RK1D since LinA computes optimal non necessarily continuous univariate PWL functions instead of optimal continuous univariate PWL functions for RK1D. The heuristic of [5] is denoted *KL2D*.

Those 5 heuristics are compared on the benchmark instances of [7], which consist of 45 instances obtained from 9 functions and 5 different absolute approximation errors for each function. An absolute approximation error of δ for function f means that the corresponding corridor \mathcal{C} is between functions u and l

satisfying $l(x, y) = f(x, y) - \delta$ and $u(x, y) = f(x, y) + \delta$. The first two functions of the benchmark are linearly separable and referred to as L1, L2. The remaining seven are not linearly separable and thus referred to as N1, ..., N7. The expression and domain of each function are described in Table 1.

For our heuristics as well as LinA2D, JuMP (v. 0.21.8) is used as modeling language, Gurobi (v. 9.1) is the (MI)LP solver and a CPU of 4.4 GHz using a single core and 32 GB RAM. Moreover, heuristic RK2D uses the modeling language GAMS (v. 23.6), the global optimization solver LindoGlobal (v. 23.6.5), a CPU intel i7 with a single core, 2.93 GHz and 12 GB RAM. Finally, KL2D uses Pyomo (v. 5.6.4) as modeling language, CPLEX (v. 12.8) and 10 threads to solve MILPs, COUENNE (v.0.5.8) and 1 thread to solve NLPs, a CPU with 3.6 GHz and 32 GB RAM. As for the time limit, LinA2D and RK2D do not have one, KL2D allows 3600s seconds for each MILP problem, while our heuristics allow 3600s seconds for each LP problem. The differences in computation time between heuristics are in several orders of magnitude, therefore these differences are mainly due to the algorithms differences and are only marginally impacted by the differences in hardware.

Table 1. Expression and domain of benchmark functions

Ref	Expression	Domain	Ref	Expression	Domain
L1	$x^2 - y^2$	$[0.5, 7.5] \times [0.5, 3.5]$	N3	$x \sin(y)$	$[1.0, 4.0] \times [0.05, 3.1]$
L2	$x^2 + y^2$	$[0.5, 7.5] \times [0.5, 3.5]$	N4	$\frac{\sin(x)}{x} y^2$	$[1.0, 3.0] \times [1.0, 2.0]$
			N5	$x \sin(x) \sin(y)$	$[0.05, 3.1] \times [0.05, 3.1]$
N1	xy	$[2.0, 8.0] \times [2.0, 4.0]$	N6	$(x^2 - y^2)^2$	$[1.0, 2.0] \times [1.0, 2.0]$
N2	$x e^{-x^2 - y^2}$	$[0.5, 2.0] \times [0.5, 2.0]$	N7	$e^{-10(x^2 - y^2)^2}$	$[1.0, 2.0] \times [1.0, 2.0]$

Results are shown in Table 2. For each of the five heuristics we present the number of pieces n obtained by the heuristic as well as its computation time in seconds. Bold integer highlight the best solution found for each instance, i.e. the ones with the minimum number of pieces. “TO” means that the heuristic has stopped because of a time out, thus without any valid solution. In this case, “-” means that no solutions were found.

In terms of minimum number of pieces, DN99 performs the best with 28 best solutions out of 45 instances, followed by KL2D with 19 out of 45, DN95 with 15 out of 45, LinA2D with 8 out of 45 and finally RK2D with 1 out of 45. As for the computation time, the hardware and software used for the different heuristics are of different quality, thus it will not be a precise tool to compare the time needed for each heuristics. However, it can be said that LinA2D takes less than 1 s to execute, RK2D and DN95 take seconds to minutes, while KL2D and DN99 take seconds to hours.

A more in-depth analysis shows that LinA2D is the best only for instances with linearly separable functions (L1 and L2), but it does really poorly on the

Table 2. Comparison with state of the art heuristics

Ref.	δ	DN95		DN99		RK2D		LinA2D		KL2D	
		n	time	n	time	n	time	n	time	n	time
L1	1.5	8	20.6	7	44.7	16	30.8	6	0.0	6	87.1
	1.0	10	28.2	9	67.6	20	84.4	8	0.0	6	148.9
	0.5	22	54.5	20	175.1	48	150.4	15	0.0	–	TO
	0.25	46	110.9	42	258.2	80	272.6	21	0.0	–	TO
	0.1	113	356.8	106	762.9	224	380.6	60	0.0	–	TO
L2	1.5	8	29.6	6	39.1	24	26.8	6	0.0	–	TO
	1.0	13	39.7	11	101.2	28	7.4	8	0.0	6	1495.5
	0.5	26	47.6	27	117.9	84	38.0	15	0.0	–	TO
	0.25	59	114.0	53	279.7	121	35.8	21	0.0	–	TO
	0.1	146	287.2	136	642.5	351	171.7	60	0.0	–	TO
N1	1.0	4	9.5	3	21.2	4	0.8	6	0.0	4	18.5
	0.5	10	22.4	6	52.4	12	72.4	8	0.0	6	416.8
	0.25	16	70.2	14	116.1	20	4.7	18	0.0	12	14762.5
	0.1	32	115.8	28	324.5	59	59.3	45	0.0	–	TO
	0.05	68	293.2	56	564.6	94	45.3	91	0.0	–	TO
N2	0.1	1	4.3	1	4.1	2	0.3	4	0.0	1	6.0
	0.05	2	14.6	2	47.8	6	18.7	9	0.0	2	13.4
	0.03	4	24.6	4	57.0	10	12.7	12	0.0	4	39.0
	0.01	11	87.9	11	450.4	31	54.6	30	0.0	–	TO
	0.001	105	1041.9	96	3832.1	350	652.6	238	0.0	–	TO
N3	1.0	3	10.4	3	17.1	5	1.0	12	0.0	–	TO
	0.5	4	10.0	4	37.4	8	13.1	16	0.0	–	TO
	0.25	7	19.5	6	42.2	16	30.0	22	0.0	8	342.6
	0.1	18	57.5	16	306.1	44	74.6	68	0.0	–	TO
	0.05	34	193.0	30	681.2	85	141.9	120	0.0	–	TO
N4	0.5	2	15.5	2	23.2	2	1.8	3	0.0	2	13.3
	0.25	3	9.2	4	57.8	4	1.0	8	0.0	4	19.6
	0.1	7	41.2	6	230.0	9	25.8	21	0.0	6	66.9
	0.05	14	161.7	11	632.6	23	14.4	27	0.0	12	7246.5
	0.03	21	272.7	19	1066.4	40	161.4	44	0.0	–	TO
N5	1.0	2	8.3	1	5.0	6	7.7	20	0.0	1	6.3
	0.5	8	33.5	5	96.5	6	1.3	42	0.0	4	86.9
	0.25	12	49.9	14	242.7	21	30.8	72	0.0	5	2091.2
	0.1	26	103.5	26	646.9	96	73.0	168	0.0	–	TO
	0.05	56	313.4	51	954.9	274	305.5	340	0.0	–	TO
N6	1.0	3	11.7	3	51.4	6	22.8	9	0.3	3	23.3
	0.5	5	22.7	5	78.6	9	15.6	9	0.0	4	127.2
	0.25	9	42.7	8	188.2	12	22.9	16	0.0	6	305.5
	0.1	16	136.7	16	803.9	40	202.8	49	0.0	–	TO
	0.05	34	516.9	28	1986.8	87	174.1	81	0.0	–	TO
N7	1.0	1	8.1	1	23.8	2	0.8	4	0.0	1	2.9
	0.5	2	32.8	2	234.8	4	66.6	4	0.0	2	9.5
	0.25	4	145.7	4	1370.6	6	4.4	9	0.0	4	35.5
	0.1	6	462.4	7	6210.7	84	231.5	16	0.0	5	377.7
	0.05	10	2050.0	–	TO	86	57.8	36	0.0	–	TO

other functions. KL2D is the best 19 times out of the 24 instances where it terminates with a solution, which shows a clear limit to the size of the instance it can tackle, due to the increasing size of MILPs it solves. The effect of the change of parameter η from 95% to 99% in heuristics DN95 and DN99 is on average a decrease of 9.1% of the number of pieces of the solutions, at the cost of an increase of 301% of the computation time.

6 Conclusion

We introduced a framework to create linearization algorithms based on the solution of the \mathbb{R}^2 -corridor fitting problem. Convex polygons were used to tile the domain instead of triangles, a scoring function ranking candidate pieces was developed and a good feasible solution of a GSIP was computed via a series of LP feasibility problems. Finally, numerical experiments showed that our heuristics outperforms the state of the art on not linearly separable functions. Further work could attempt to diminish the relatively high computation time of the heuristics, search for better instantiation of the different parts of the framework or apply those heuristics to approximate real-world MINLPs to show their practical usage.

References

1. Codsi, J., Ngueveu, S.U., Gendron, B.: Lina: a faster approach to piecewise linear approximations using corridors and its application to mixed-integer optimization. Technical report (2021)
2. Duguet, A.: Appendix of "piecewise linearization of bivariate nonlinear functions: minimizing the number of pieces under a bounded approximation error": determining the best heuristic (2022). <https://homepages.laas.fr/sungueve/2dpwl.html>
3. Frenzen, C., Sasao, T., Butler, J.T.: On the number of segments needed in a piecewise linear approximation. *J. Comput. Appl. Math.* **234**(2), 437–446 (2010). <https://doi.org/10.1016/j.cam.2009.12.035>
4. Geißler, B., Martin, A., Morsi, A., Schewe, L.: Using piecewise linear functions for solving MINLPs. In: *Mixed Integer Nonlinear Programming. The IMA Volumes in Mathematics and its Applications*, vol. 154, pp. 287–314. Springer, Heidelberg (2012). https://doi.org/10.1007/978-1-4614-1927-3_10
5. Kazda, K., Li, X.: Nonconvex multivariate piecewise-linear fitting using the difference-of-convex representation. *Comput. Chem. Eng.* **150**, 107310 (2021). <https://doi.org/10.1016/j.compchemeng.2021.107310>
6. Ngueveu, S.U.: Piecewise linear bounding of univariate nonlinear functions and resulting mixed integer linear programming-based solution methods. *Eur. J. Oper. Res.* **275**(3), 1058–1071 (2019). <https://doi.org/10.1016/j.ejor.2018.11.021>
7. Rebennack, S., Kallrath, J.: Continuous piecewise linear delta-approximations for bivariate and multivariate functions. *J. Optim. Theory Appl.* **167**(1), 102–117 (2014). <https://doi.org/10.1007/s10957-014-0688-2>
8. Rebennack, S., Krasko, V.: Piecewise linear function fitting via mixed-integer linear programming. *INFORMS J. Comput.* **32**(2), 507–530 (2020). <https://doi.org/10.1287/ijoc.2019.0890>

9. Toriello, A., Vielma, J.P.: Fitting piecewise linear continuous functions. *Eur. J. Oper. Res.* **219**(1), 86–95 (2012). <https://doi.org/10.1016/j.ejor.2011.12.030>
10. Vielma, J.P., Ahmed, S., Nemhauser, G.: Mixed-integer models for nonseparable piecewise-linear optimization: Unifying framework and extensions. *Oper. Res.* **58**(2), 303–315 (2010). <https://doi.org/10.1287/opre.1090.0721>