Piecewise linearization of bivariate nonlinear functions: minimizing the number of pieces under a bounded approximation error

Aloïs Duguet, Sandra Ulrich Ngueveu

Piecewise linearization of bivariate nonlinear functions: minimizing the number of pieces under a bounded approximation error

Aloïs Duguet Sandra Ulrich Ngueveu

April 4, 2022

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.

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 $(x,y) \in \mathbb{D}$:

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

Constraints (2) are pointwise approximation constraints therefore there is an infinite number of constraints because $\mathbb{D}$ is a continuous domain. The objective is to minimize the number of pieces of g so that a MILP formulation of g introduces less variables and constraints Vielma et al. [2010]. It is especially 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 Geißler et al. [2012] showed that in some cases, MINLP can be solved by applying only techniques from MILP.

Heuristics and exact methods exist to approximate univariate nonlinear functions ($\mathbb{D} \subset \mathbb{R}$) Codsi et al. [2021], Kong and Maravelias [2020], Ngueveu [2019],

Rebennack and Krasko [2020]. We are interested in bivariate functions ($\mathbb{D} \subset \mathbb{R}^2$) and to the best of our knowledge, only 2 papers address this case, with heuristics only:

- The authors of Rebennack and Kallrath [2015] 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 that fits it has been found. The verification that a given linear function fits the subdomain is made by solving a Non Linear Programming problem (NLP). The second heuristic can be used if the contribution of the two variables in the function can be separated (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 Kazda and Li [2021] 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 a 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 Section 2, the three key ideas of a framework for piecewise linearization are detailed in Section 3. It is followed by explanations on the instantiation of crucial parts of this framework to create different heuristics in Section 4, and finally, numerical experiments comparing the state of the art to our best heuristics are shown in Section 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 Codsi et al. [2021] for $\mathbb{D} \subset \mathbb{R}$.

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

Throughout the paper, a *piece* will refer to a polytope that composes the graph of a PWL function.

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 $\{a_i\}_{i \in [\![1,n]\!]} \subset \mathbb{R}^2$, $\{b_i\}_{i \in [\![1,n]\!]} \subset \mathbb{R}$ and a family of polytopes $\{D_i\}_{i \in [\![1,n]\!]} \subset \mathbb{R}^2$ such that $\mathbb{D} = \cup_{i \in [\![1,n]\!]} \mathbb{D}_i$, and for $i \neq j$ the polytopes $\mathbb{D}_i$ and $\mathbb{D}_j$ can only intersect on their frontier and g is defined by $g(x, y) = \min\{a_i.(x, y)^T + b_i | (x, y) \in \mathbb{D}_i \ \forall i \in [\![1, n]\!]\}$, with . denoting the standard scalar product.

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

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 $u(x,y) > l(x,y), \forall (x,y) \in \mathbb{D}$. The set $\mathcal{C} \subset \mathbb{R}^3$ is called the corridor between u and l if and only if $\mathcal{C} = \{(x,y,z) \in \mathbb{R}^3 | (x,y) \in \mathbb{D}, l(x,y) \leq z \leq u(x,y)\}$. 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 $\{P\}_{i\in[\![1,n]\!]}$ 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 $l_0(x,y) \leq l(x,y) \leq u(x,y) \leq u_0(x,y)$.

Definition 9 ($\mathbb{R}^m$-corridor fitting problem)**.** The corridor fitting problem consists in finding a PWL function g fitting a corridor $\mathcal{C}$ such that its number of pieces is minimized. $\mathbb{R}^m$-corridor fitting problem refers to the problem with $D(\mathcal{C}) \subset \mathbb{R}^m$.

($\mathcal{P}$) is equivalent to an $\mathbb{R}^2$-corridor fitting problem with corridor $\mathcal{C}$ between u and l, thus we will refer only to $\mathbb{R}^2$-corridor fitting problem from now on.

Definition 10 (Truncated-corridor)**.** Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two corridors both defined by functions u and l on the interval $[a,b_1]$ 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 corridor domains compacts of $\mathbb{R}^2$. 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 $D(\mathcal{C}_2) = D(\mathcal{C}_1) \cap \{(x,y) \in \mathbb{R}^2, (x,y).d \leq \sigma\}$, 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 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 Section 4 as well as some details on the implementation.

Three key ideas are followed in the framework. The first two are meant to avoid drawbacks encountered in Rebennack and Kallrath [2015], whereas the third one is meant to render a subproblem more tractable:

- **Key idea 1:** Reduce the number of pieces that compose the PWL function, pieces should be chosen among general convex polygons instead of triangles
- **Key idea 2:** Choose pieces that are good (ideally optimal) solutions of a maximal piece in direction d problem, so as to ensure that the domain covered by a piece is as large as possible
- **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 a MILP. Such shapes are polygons, but we further restrict those shapes to convex polygons because formulating a non-convex polygon in a MILP introduces additional binary variables. It is expected that allowing convex polygons instead of only triangles as done in Rebennack and Kallrath [2015] 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 Section 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 the least number of pieces

function PWL_2D_FITTING($\mathcal{C}$)
  $\mathcal{P} \leftarrow \emptyset$ ▷ list of chosen pieces
  $\mathcal{Q} \leftarrow \{\mathcal{C}\}$ ▷ list of corridors with convex domains not already tiled
  while $\mathcal{Q} \neq \emptyset$ **do**
    $\mathcal{C} = pop(\mathcal{Q})$
    $candidate\_pieces \leftarrow \emptyset$
    for v vertex of $D(\mathcal{C})$ **do**
      $d \leftarrow choose\_progress\_direction(\mathcal{C}, v)$
      $candidate\_pieces \leftarrow candidate\_pieces \cup compute\_piece(\mathcal{C}, v, d)$
    end for
    $p \leftarrow argmax_{p \in candidate\_pieces} score(p)$
    $\mathcal{P} \leftarrow \mathcal{P} \cup p$
    $\mathcal{Q} = update\_domain(\mathcal{C}, p)$
  end while
  return $\mathcal{P}$
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):

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

Where $D(\mathcal{C}_\sigma^d) = D(\mathcal{C}) \cap \{(x,y) \in \mathbb{R}^2 | (x,y)^T.d \leq \sigma\}$ is the domain of $\mathcal{C}_\sigma^d$, the truncated-corridor of $\mathcal{C}$ in direction d. (MP_d) can be modeled as a semi-infinite programming problem (SIP). Indeed, the number of pointwise constraints is infinite 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 an SIP is in general hard to solve exactly, a feasible solution of (MP_d) is computed via a series of LP problems, as described below, using the notion of

PWL inner corridor, because it allows to replace the infinite number of nonlinear constraints by a finite number of linear constraints.

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

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

Remark 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 (MP''_d) is equivalent to (MP'_d) but has the advantage of having only a finite number of linear constraints.

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

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, an LP feasibility problem $(MP''_{d,\sigma})$ is obtained, and to optimize σ, problem $(MP''_{d,\sigma})$ can be repeatedly solved until a satisfactory σ value has been found.

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

4 Framework Key Points Instantiation

In this section, choices made on key points of the framework of Section 3 are described: the function *score* of Algorithm 1 in Section 4.1, the choice of a direction d in Section 4.2 and the computation of a PWL inner corridor in Section 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:

- *Surface* (Surf) measures the surface 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 15. 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.

Surf 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 Frenzen et al. [2010]. Indeed, it states that for a corridor $\mathcal{C}$ of constant pointwise height 2δ with $D(\mathcal{C}) = [a, b]$, when $\delta \to 0$, the minimum number of pieces $s(\delta)$ satisfies the asymptotic approximation $s(\delta) \sim \frac{1}{4\delta} \int_a^b \sqrt{|u''(x, y)|} dx$. Remark 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 Surf 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 pointed by 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 basic idea is to divide $D(\mathcal{C})$ into rectangular pieces of same size, 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 16.

Proposition 16. *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 $[\nabla u_x^{low}, \nabla u_x^{high}] \times [\nabla u_y^{low}, \nabla u_y^{high}]$ be such that $\nabla u(x,y) \in [\nabla u_x^{low}, \nabla u_x^{high}] \times [\nabla u_y^{low}, \nabla u_y^{high}]$ for all $(x,y) \in \mathcal{D}$. Let $(M_x, M_y) = (\frac{a+b}{2}, \frac{c+d}{2})$. Define:*

$$u^-_{(a,c)} := u(M_x, M_y) - \nabla u_x^{high}.l_x - \nabla u_y^{high}.l_y \tag{8}$$

$$u^-_{(b,c)} := u(M_x, M_y) + \nabla u_x^{low}.l_x - \nabla u_y^{high}.l_y \tag{9}$$

$$u^-_{(b,d)} := u(M_x, M_y) + \nabla u_x^{low}.l_x + \nabla u_y^{low}.l_y \tag{10}$$

$$u^-_{(a,d)} := u(M_x, M_y) - \nabla u_x^{high}.l_x + \nabla u_y^{low}.l_y \tag{11}$$

If a linear function f satisfies:

$$f(a,c) \leq u^-_{(a,c)},\ f(b,c) \leq u^-_{(b,c)},\ f(b,d) \leq u^-_{(b,d)} \text{ and } f(a,d) \leq u^-_{(a,d)} \tag{12}$$

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:

$$\begin{aligned}
u(M_x,M_y) - \nabla u_x^{high}.(M_x - x_0) - \nabla u_y^{high}.(M_y - y_0) &\leq u(x_0,y_0) \quad \text{if } x_0 \leq M_x, y_0 \leq M_y \\
u(M_x,M_y) + \nabla u_x^{low}.(x_0 - M_x) - \nabla u_y^{high}.(M_y - y_0) &\leq u(x_0,y_0) \quad \text{if } x_0 \geq M_x, y_0 \leq M_y \\
u(M_x,M_y) + \nabla u_x^{low}.(x_0 - M_x) + \nabla u_y^{low}.(y_0 - M_y) &\leq u(x_0,y_0) \quad \text{if } x_0 \geq M_x, y_0 \geq M_y \\
u(M_x,M_y) - \nabla u_x^{high}.(M_x - x_0) + \nabla u_y^{low}.(y_0 - M_y) &\leq u(x_0,y_0) \quad \text{if } x_0 \leq M_x, y_0 \geq M_y
\end{aligned}$$

because $[\nabla u_x^{low}, \nabla u_x^{high}] \times [\nabla u_y^{low}, \nabla u_y^{high}]$ are bounds of ∇u on domain $\mathcal{D}$. Now, define a PWL function f_{PWL} with four rectangle pieces with vertices position on $\{(a,c),(b,c),(b,d),(a,d),(\frac{a+b}{2},\frac{c+d}{2})\}$ 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 $\mathcal{D}$. □ □

Thus, given a PWL inner corridor $\mathcal{C}_{PWL}$ of $\mathcal{C}$, the feasibility of fitting a linear function in $\mathcal{C}_{PWL}$ can be checked by solving an LP feasibility problem, with 3 variables, and constraints C for each piece of $\tilde{u}$ and $\tilde{l}$, as in $(MP''_{d,\sigma})$.

To build a PWL inner corridor exploiting Proposition 16 in our algorithms, a method called *efficiency refinement* is used. It is described after the introduction of the necessary vocabulary.

Definition 17 (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 18. *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 $\tilde{l}$.*

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. □ □

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 inner corridor 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.

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. The name of our resulting heuristics are given according to this template {scoring function}{η in %}{method to choose progress direction}. Thus, PaD95med denote the heuristic using PaD as scoring function, $\eta = 95\%$ as efficiency in the efficiency refinement and heuristic med to choose a progress direction.

Heuristics from the state of the art are described in articles Codsi et al. [2021], Kazda and Li [2021], Rebennack and Kallrath [2015]. Recall that Rebennack and Kallrath [2015] 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 LinA1D is based on the same decomposition as RK1D, but uses library LinA from Codsi

ref	expression	Domain	ref	expression	Domain
L1	$x^2 - y^2$	$[0.5, 7.5] \times [0.5, 3.5]$	N3	$xsin(y)$	$[1.0, 4.0] \times [0.05, 3.1]$
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	$xsin(x)sin(y)$	$[0.05, 3.1] \times [0.05, 3.1]$
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	$xe^{-x^2-y^2}$	$[0.5, 2.0] \times [0.5, 2.0]$	N7	$e^{-10(x^2-y^2)^2}$	$[1.0, 2.0] \times [1.0, 2.0]$

Table 1: Expression and domain of benchmark functions

et al. [2021] to compute the approximation of univariate functions. Obviously LinA1D outperforms RK1D since LinA computes optimal non necessarily continuous univariate PWL functions instead of optimal continuous univariate PWL functions for RK1D. The heuristic of Kazda and Li [2021] is denoted *KL2D*. *PaD95med* is our best heuristic with the efficiency refinement parameter η to 95%, and *PaD99med* is our best heuristic with η to 99%, when comparing them to the four possibilities made of $Surf/PaD$ and bd/med.

Those 5 heuristics are compared on the benchmark instances of Rebennack and Kallrath [2015], which consists 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 refered to as L1,L2. The remaining seven are not linearly separable and thus refered to as N1,...,N7. The expression and domain of each function are described in Table 1.

For our heuristics as well as LinA1D, 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 32GB RAM. Moreover, heuristic RK2D use 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 time limit, LinA1D and RK2D do not have one, KL2D allows 3600 seconds for each MILP problem, while our heuristics allow 3600 seconds for each LP problem.

Results are shown in Table 5. 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, PaD99med performs the best with 27 best solutions out of 45 instances, followed by KL2D with 20 out of 45, PaD95med with 16 out of 45, LinA1D 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 second to execute, RK2D and PaD95med take seconds to minutes, while KL2D and PaD99med take seconds to hours.

A more in-depth analysis shows that LinA1D is the best only for instances with linearly separable functions (L1 and L2), but it does really poorly on the other functions. KL2D is the best 20 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. The effect of the change of parameter η from 95% to 99 % is in average a decrease of 7.6% of the pieces needed, at the cost of an increase of 275% 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 SIP 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 instanciation of the different parts of the framework or apply those heuristics to approximate real-world MINLPs to show their practical usage.

References

J. Codsi, S. U. Ngueveu, and B. Gendron. Lina: A faster approach to piecewise linear approximations using corridors and its application to mixed-integer optimization. Technical report, 2021.

C. Frenzen, T. Sasao, and J. T. Butler. On the number of segments needed in a piecewise linear approximation. Journal of Computational and Applied Mathematics, 234(2):437–446, 2010. doi: 10.1016/j.cam.2009.12.035.

B. Geißler, A. Martin, A. Morsi, and L. Schewe. Mixed Integer Nonlinear Programming, chapter Using piecewise linear functions for solving MINLPs, pages 287–314. The IMA Volumes in Mathematics and its Applications, Vol 154, 2012. doi: 10.1007/978-1-4614-1927-$3_1$0.

K. Kazda and X. Li. Nonconvex multivariate piecewise-linear fitting using the difference-of-convex representation. Computers & Chemical Engineering, 150: 107310, 2021. doi: 10.1016/j.compchemeng.2021.107310.

		PaD95med		PaD99med		RK2D		LinA1D		KL2D	
ref	δ	n	time	n	time	n	time	n	time	n	time
L1	1.5	8	18.1	7	23.8	16	30.8	**6**	0.014	**6**	87.1
	1.0	10	19.9	9	35.2	20	84.4	8	0.004	**6**	148.9
	0.5	22	40.5	22	80.7	48	150.4	**15**	0.004	-	TO
	0.25	40	102.4	38	149.2	80	272.6	**21**	0.004	-	TO
	0.1	116	220.7	97	393.9	224	380.6	**60**	0.006	-	TO
L2	1.5	8	19.8	**6**	23.2	24	26.8	**6**	0.003	-	TO
	1.0	15	27.3	13	39.8	28	7.4	8	0.003	**6**	1495.5
	0.5	22	46.2	23	75.1	84	38.0	**15**	0.003	-	TO
	0.25	50	75.0	44	126.2	121	35.8	**21**	0.004	-	TO
	0.1	132	157.1	128	338.3	351	171.7	**60**	0.004	-	TO
N1	1.0	4	12.5	**3**	12.4	4	0.8	6	0.003	4	18.5
	0.5	10	21.9	**6**	27.4	12	72.4	8	0.004	**6**	416.8
	0.25	16	44.9	14	70.1	20	4.7	18	0.004	**12**	14762.5
	0.1	32	181.4	**30**	193.1	59	59.3	45	0.006	-	TO
	0.05	66	153.0	**57**	493.5	94	45.3	91	0.006	-	TO
N2	0.1	**1**	8.7	**1**	3.1	2	0.3	4	0.004	**1**	6.0
	0.05	**2**	8.1	**2**	25.7	6	18.7	9	0.006	**2**	13.4
	0.03	**4**	12.9	**4**	34.2	10	12.7	12	0.008	**4**	39.0
	0.01	**11**	102.7	**11**	227.9	31	54.6	30	0.005	-	TO
	0.001	100	749.2	**95**	2258.7	350	652.6	238	0.008	-	TO
N3	1.0	**3**	11.0	**3**	11.1	5	1.0	12	0.004	-	TO
	0.5	**4**	11.5	**4**	22.2	8	13.1	16	0.007	-	TO
	0.25	7	21.6	**6**	28.6	16	30.0	22	0.007	8	342.6
	0.1	20	64.5	**16**	136.1	44	74.6	68	0.008	-	TO
	0.05	35	110.1	**29**	398.5	85	141.9	120	0.013	-	TO
N4	0.5	**2**	10.1	**2**	10.2	**2**	1.8	3	0.004	**2**	13.3
	0.25	**3**	10.0	4	34.4	4	1.0	8	0.007	4	19.6
	0.1	7	37.1	**6**	125.5	9	25.8	21	0.007	**6**	66.9
	0.05	14	128.7	**11**	363.4	23	14.4	27	0.006	12	7246.5
	0.03	**20**	178.4	**20**	459.5	40	161.4	44	0.006	-	TO
N5	1.0	2	11.0	**1**	3.9	6	7.7	20	0.006	**1**	6.3
	0.5	8	23.7	5	54.0	6	1.3	42	0.007	**4**	86.9
	0.25	12	44.4	14	115.8	21	30.8	72	0.007	**5**	2091.2
	0.1	**26**	119.0	**26**	463.7	96	73.0	168	0.01	-	TO
	0.05	57	189.6	**51**	653.8	274	305.5	340	0.012	-	TO
N6	1.0	**3**	15.1	**3**	27.9	6	22.8	9	0.267	**3**	23.3
	0.5	5	19.6	5	41.4	9	15.6	9	0.004	**4**	127.2
	0.25	9	32.1	8	97.6	12	22.9	16	0.005	**6**	305.5
	0.1	**15**	94.4	16	471.2	40	202.8	49	0.006	-	TO
	0.05	34	273.7	**28**	1088.1	87	174.1	81	0.006	-	TO
N7	1.0	**1**	8.9	**1**	15.2	2	0.8	4	0.003	**1**	2.9
	0.5	**2**	23.7	**2**	115.7	4	66.6	4	0.003	**2**	9.5
	0.25	**4**	123.8	**4**	634.4	6	4.4	9	0.004	**4**	35.5
	0.1	7	266.5	7	2680.1	84	231.5	16	0.004	**5**	377.7
	0.05	**10**	1278.7	11	76234.0	86	57.8	36	0.004	-	TO

Table 2: Comparison with state of the art heuristics

L. Kong and C. T. Maravelias. On the derivation of continuous piecewise linear approximating functions. INFORMS Journal on Computing, 32(3):531–546, 2020. doi: 10.1287/ijoc.2019.0949.

S. U. Ngueveu. Piecewise linear bounding of univariate nonlinear functions and resulting mixed integer linear programming-based solution methods. European Journal of Operational Research, 275(3):1058–1071, 2019. doi: 10.1016/j.ejor.2018.11.021.

S. Rebennack and J. Kallrath. Continuous piecewise linear delta-approximations for bivariate and multivariate functions. J Optim Theory Appl, 167:102–117, 2015. doi: 10.1007/s10957-014-0688-2.

S. Rebennack and V. Krasko. Piecewise linear function fitting via mixed-integer linear programming. INFORMS Journal on Computing, 32(2):507–530, 2020. doi: 10.1287/ijoc.2019.0890.

J. P. Vielma, S. Ahmed, and G. Nemhauser. Mixed-integer models for non-separable piecewise-linear optimization: Unifying framework and extensions. Operations Research, 58(2), 2010. doi: 10.1287/opre.1090.0721.