



LinA: a faster approach to piecewise linear approximations using corridors and its application to mixed-integer optimization

Julien Codsi^{1,2} · Sandra Ulrich Ngueveu³ · Bernard Gendron⁴

Received: 17 November 2021 / Accepted: 23 October 2024 / Published online: 23 January 2025
© Springer-Verlag GmbH Germany, part of Springer Nature and Mathematical Optimization Society 2025

Abstract

In this paper, we address the problem of approximating and over/under-estimating univariate functions with piecewise linear (PWL) functions with the minimum number of linear segments given a bound on the allowed pointwise approximation error. Through a new geometric approach and building on the work of Ngueveu (Eur J Oper Res 275:1058–1071, 2019), we develop new algorithms that can solve the problem in quasi-logarithmic time on a very broad class of error types. Such algorithms find many applications, mostly related to solving certain classes of (mixed-integer) nonlinear and nonconvex programming (MINLP) problems by mixed-integer linear programming (MILP) techniques. An efficient implementation of our algorithms is available as a Julia package. Benchmarks are also provided to showcase how our method outperforms the state-of-the-art for this problem. Finally, we show how our algorithms can be used to efficiently solve certain classes of MINLP problems through a case study on multicommodity network design problems with congestion.

Keywords Piecewise linear functions · Approximation · Overestimation · Underestimation · Guaranteed tolerance · Piecewise linear regression with bounded error · MINLP · MILP · Julia package

✉ Julien Codsi
julien.codsi@umontreal.ca

Sandra Ulrich Ngueveu
ngueveu@laas.fr

Bernard Gendron
bernard.gendron@cirrelt.net

¹ Département d'informatique et de recherche opérationnelle, Université de Montréal, Montréal, Canada

² Princeton University, Princeton, NJ, USA

³ LAAS-CNRS, CNRS, INP, Université de Toulouse, Toulouse, France

⁴ CIRRELT and Département d'informatique et de recherche opérationnelle, Université de Montréal, Montréal, Canada

Mathematics Subject Classification 41A15 · 90C59 · 90C30 · 90C11 · 65D07

1 Introduction

The simplicity and ease of use of linear functions make them very attractive to many researchers and practitioners. However, when dealing with non-linear behaviors, using models that better encompass reality is often preferable. As a trade-off between simplicity and precision, piecewise linear functions (PWL) are used in a variety of fields, such as computer graphics, data science, and optimization. Throughout these fields, different criteria are used to judge the quality of a PWL approximation. Although our results are very general, this paper is motivated by the applications of PWL functions in Mixed integer nonlinear programming (MINLP), which guided the exact choices for the specifications of the problem tackled.

MINLP models are generally hard to solve. In addition to the presence of integer variables, one often has to handle nonlinear functions that are not necessarily convex. A widely used approach is to approximate the nonlinear functions with piecewise linear ones to derive mixed integer linear programming (MILP) models, thus benefiting from the optimization community's large-scale, continuous, and sustained effort on MILP for the past thirty years and beyond. Geißler et al. [6] were among the first to show that, in certain cases, MINLP models can be solved by applying purely techniques from MILP after approximating nonlinearities by piecewise linear functions. However, they did not focus on minimizing the number of linear segments. Piecewise linear approximation for solving MINLP is often performed in a preprocessing step using ad hoc methods. In general, the approximation error is not known beforehand, and the number of linear pieces is not minimized. This translates into an increased number of binary variables in the MILP model, which is problematic as MILP can scale exponentially in terms of the number of binary variables. In this paper, we address these issues and propose a geometric approach to compute a PWL function that minimizes the number of linear segments needed to approximate, over-estimate, or under-estimate a nonlinear continuous univariate function with a bounded pointwise approximation error.

The specificities of our problem, which separate it from the large majority of studies in the field of univariate piecewise linear approximation, are the following:

- Most contributions on piecewise linear approximations of nonlinear univariate functions focus on the minimization of an approximation error given a predefined number of linear segments (see for example [1, 2]). In contrast, in this paper, we are interested in minimizing the number of linear segments given a bounded approximation error. Using those algorithms combined with binary search could have been a valid approach to our problem. However, minimizing the error given the number of breakpoints is, to the best of our knowledge, computationally harder than the problem at hand. Moreover, this would imply solving this error minimization problem multiple times, which would be impractical.
- Because the ultimate goal is to solve MINLP problems using techniques from MILP, we are interested in pointwise errors, i.e., errors that can be expressed

as a function of the maximal difference between the nonlinear function and its approximation. This excludes most of the metrics classically used for piecewise linear regression in data mining or statistics, such as the sum of square deviations or the total sum of absolute deviations [5, 26, 27].

- Most piecewise linear regression studies, also known as segmented linear regression, curve fitting, or piecewise linear function fitting, consider a discrete set of points as an input instead of the continuous function we consider. Even in cases where the continuous function is known, the function is sampled, and the approximation is performed on the set of sample points. The algorithms proposed in the regression field do not ensure the predefined approximation error on the entire continuous domain and are therefore not directly applicable to our problem [2, 4, 12, 23].
- Finally, we are interested in exact methods providing optimal solutions or, at least, guarantees on the quality of the solutions produced, which excludes most heuristics from the literature.

To the best of our knowledge, only five papers address the specific problem we are interested in, computing piecewise linear functions that minimize the number of linear segments, given a bounded approximation error expressed in function of the difference with the nonlinear univariate function.

Rosen and Pardalos [19] first proposed to build *continuous* PWL interpolators that verify a specified “tolerance”, but only for concave quadratic programs and using equidistant breakpoints.

Rebennack and Kallrath [16] introduced two nonconvex optimization models and two heuristics for the computation of *continuous* PWL approximations that minimize the number of linear segments. The authors distribute breakpoints freely and allow shifts from the function at breakpoints, leading to up to an order of magnitude fewer breakpoints compared to the classical equidistant interpolation approach.

Rebennack and Krasko [17] proposed the first convex model for computing a *continuous* PWL approximation of a finite set of points that minimizes the number of pieces. The model proposed is then used to develop an exact algorithm for the piecewise linear approximation of continuous univariate functions. The convex formulation is based on the idea that computing the exact locations of the breakpoints is not needed during function fitting. Rather, it is sufficient to ensure that adjacent linear segments of the constructed function intersect within a certain range. The exact algorithm consists of solving a series of finite point fitting problems via the proposed convex models. By evaluating the computed *continuous* PWL function (also known in the literature as linear or first-order splines), the authors can identify points where the maximum difference with the original function is larger than desired. These points are added to the discretization, and new finite point fitting problems are solved. Computational results show that the resulting algorithm outperforms the ones from [16].

Kong and Maravelias [10] also proposed mixed-integer programs for computing a *continuous* PWL approximation of a finite set of points that is then used to develop an exact algorithm for the piecewise linear approximation of continuous univariate functions. However, the reported computational results show that this is more efficient

than approaches that utilize nonlinear constraints, but the proposed method may not be competitive in comparison to [16, 17].

Ngueveu [13] propose approximating a general univariate continuous function with a *non necessarily continuous* PWL function, adding an additional degree of freedom to obtain a breakpoint system with as many or less linear segments. The author presents models and algorithms to compute the PWL approximator/over-estimator/under-estimator with predefined absolute or relative error tolerance and with an additive worst-case guarantee on the number of linear segments needed. Evaluations on the instances from [16, 17] show a drastic reduction of the computing times in comparison to the state-of-the-art and sometimes a reduction of the number of linear segments. For solving MINLPs with an objective function that is separable in a sum of positive univariate nonlinear terms, [13] proposes a method based on upper and lower bounding the nonlinear terms using *non necessarily continuous* PWL functions with a predefined relative tolerance and the solution of a pair of mixed integer linear programs. Such an approach yields a performance guarantee when the nonlinearity is restricted to the objective function. To illustrate the efficiency of the method in comparison to state-of-the-art methods and general-purpose MINLP solvers, computational evaluation was performed on an energy optimization problem for hybrid electric vehicles.

The main contributions of this paper are the following: (I) the notion of a “corridor” that generalizes the classes of errors considered in piecewise linear approximation, (II) through a greedy algorithm, a reduction of the original problem to the maximal linear piece problem, (III) an algorithm to solve the problem in the general case, (IV) a logarithmic time algorithm for the special case of convex corridors through a strong characterization of the optimal solution, (V) clever speed-ups for a large class of corridors, (VI) an efficient and flexible Julia package, (VII) benchmarks on nonlinear functions from the literature that illustrate the major performance speed-ups of our method and (VIII) benchmark on the resolution of MINLP through our linearization + MILP solver approach

Most notably, the practical performance is drastically improved thanks to a new characterization of optimal solutions with intersections and tangents that are easy to compute. The piecewise linear functions computing times reported in [13] varied from dozen to hundreds of seconds. On the other hand, with our new algorithm, we computed PWL approximations in mere milliseconds on these instances while achieving fewer linear segments in the case of relative errors. Although the results from [13] have proven to be sufficient to outperform general-purpose MINLP solvers on problems containing a single univariate nonlinear function to approximate, they were incapable of tackling cases where there might be multiple nonlinear functions to approximate in a single instance within a reasonable time. Therefore, our results let us solve new classes of MINLP than what was possible with this kind of approach. For example, this is the case of the network design problem with congestion studied in this paper, where there is a different nonlinear congestion function for each node of a graph, and graphs can contain up to a hundred nodes. We showed that our algorithm allows us to compete with the state-of-the-art on this problem and significantly outperforms other linearization + MILP methods based on commonly used linearization algorithms.

2 Definitions

Here, we define terms that will be used throughout this paper.

Definition 1 (*PWL function*) A function $g : \mathbb{D} = [x_-, x_+] \rightarrow \mathbb{R}$ is a piecewise linear (PWL) function with n linear segments if it can be defined by Eq. (1) where $x_1 = x_-$ and $x_{n+1} = x_+$. The *domain length* of g is equal to $x_+ - x_-$.

$$g(x) = \begin{cases} g_i(x) = a_i x + b_i, & \forall i \in \{1, \dots, n-1\}, \forall x \in [x_i, x_{i+1}[\\ g_n(x) = a_n x + b_n, & \text{if } x \in [x_n, x_{n+1}] \end{cases} \quad (1)$$

Remark 1 It is important to note that the continuity property $g_{i-1}(x_i) = g_i(x_i) \forall i \in \{2, \dots, n\}$ is neither imposed nor forbidden, which implies that a PWL is non necessarily continuous.

Definition 2 (*Corridor*) Let $h, l : \mathbb{D} = [x_-, x_+] \rightarrow \mathbb{R}$ be two continuous functions verifying $h(x) > l(x), \forall x \in \mathbb{D}$. The surface area $\mathcal{C} \subset \mathbb{R}^2$ is called the *corridor* between h and l iff $\mathcal{C} = \{(x, y) \mid x \in \mathbb{D}, l(x) \leq y \leq h(x)\}$.

The *domain length* of \mathcal{C} is equal to $x_+ - x_-$.

Definition 3 (*Sub-corridor*) Let \mathcal{C}_1 and \mathcal{C}_2 be two corridors defined by functions $h_1, l_1 : \mathbb{D}_1 \rightarrow \mathbb{R}$ and $h_2, l_2 : \mathbb{D}_2 \rightarrow \mathbb{R}$, respectively. We call \mathcal{C}_2 a sub-corridor of \mathcal{C}_1 iff $\mathbb{D}_2 \subseteq \mathbb{D}_1, h_1(x) = h_2(x)$, and $l_1(x) = l_2(x), \forall x \in \mathbb{D}_2$.

Definition 4 (*Truncated-corridor*) Let \mathcal{C}_1 and \mathcal{C}_2 be two corridors defined by some function l and h respectively on the interval $[a, b]$ and $[c, d]$. We call \mathcal{C}_2 a truncated-corridor of \mathcal{C}_1 iff \mathcal{C}_2 is a sub-corridor of \mathcal{C}_1 and $a = c$.

Definition 5 (*Function within a corridor*) A function $g : \mathbb{D}_g \rightarrow \mathbb{R}$ is *within* a corridor \mathcal{C} iff $(x, g(x)) \in \mathcal{C}, \forall x \in \mathbb{D}_g$.

Definition 6 (*Induced sub-corridor*) Let $g : \mathbb{D}_g \rightarrow \mathbb{R}$ be a function within a corridor \mathcal{C}_1 . A corridor \mathcal{C}_2 is called an induced sub-corridor iff $\mathcal{C}_2 = \mathcal{C}_1 \cap (\mathbb{D}_g \times \mathbb{R})$, i.e \mathcal{C}_2 is the same as \mathcal{C}_1 but limited to the region where g is defined. We denote $\mathcal{C}_2 = \mathcal{C}_1(g)$

Definition 7 (*Fitting*) A function g fits a corridor \mathcal{C} iff g is within \mathcal{C} and $\mathcal{C}(g) = \mathcal{C}$, i.e the domain length of g is the same as the domain length of \mathcal{C} .

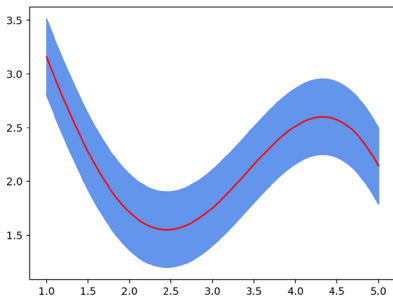
Definition 8 (*Maximal linear segment*) A maximal linear segment in a corridor \mathcal{C} is a linear segment within \mathcal{C} that induces a truncated corridor of maximal domain length.

3 Fitting a piecewise linear function through a corridor

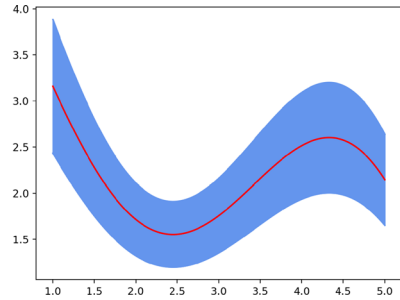
We use the notion of *corridors* (see Definition 2) to present a new geometric approach to the problem. Intuitively, a corridor is defined as the region between two functions that

Table 1 Equivalence of the piecewise linear approximation, overestimation, and underestimation of a function f with the corridor fitting problem for a corridor \mathcal{C} defined by h and l

	Absolute tolerance δ	Relative tolerance ε
Approximation	$h(x) = f(x) + \delta$ $l(x) = f(x) - \delta$	$h(x) = f(x) + \varepsilon f(x) $ $l(x) = f(x) - \varepsilon f(x) $
Overestimator	$h(x) = f(x) + \delta$ $l(x) = f(x)$	$h(x) = f(x) + \varepsilon f(x) $ $l(x) = f(x)$
Underestimator	$h(x) = f(x)$ $l(x) = f(x) - \delta$	$h(x) = f(x)$ $l(x) = f(x) - \varepsilon f(x) $



(a) absolute error tolerance



(b) relative error tolerance

Fig. 1 Corridors induced by an absolute and a relative error

do not intersect on a compact interval. We are interested in finding a PWL function g that *fits* (see Definition 7) a given corridor \mathcal{C} with a minimal number of linear segments. This problem is named the *corridor fitting problem*. To compute an approximation, an overestimation, or an underestimation of a function with a predefined pointwise error tolerance, we define a corridor surrounding the function so that the problem is equivalent to finding a piecewise linear function fitting the corridor. The corridor fitting problem, therefore, generalizes the problem of finding a piecewise linear function that overestimates, underestimates, or approximates an univariate function such that the number of linear segments is minimized given a bound on a pointwise error metric. As shown in Table 1, this includes corridors induced by absolute and relative pointwise errors, thus generalizing the work of [13].

To visualize this, Fig. 1 shows the corridors associated with the function $f(x) = \sqrt{x} \sin(-x) - x + 5$ defined on the interval $[1, 5]$ for an absolute error of 0.35 and for a relative error of 23%.

Such error classes find many applications, mostly related to solving (mixed-integer) nonlinear and nonconvex programming problems by (mixed-integer) linear programming techniques.

Remark 2 The absolute error case can be interpreted as a bound on the Chebyshev norm $\| \cdot \|_\infty$ between the PWL function and the original function.

3.1 Overview of the algorithm

Our solution is based on a greedy algorithm that creates the PWL function iteratively by choosing the linear segment with the maximal possible domain length at every step. This reduces the corridor fitting problem to a simpler problem, namely the *maximal linear segment problem* (see Definition 8). This procedure is detailed in Algorithm 1.

Algorithm 1 Computation of an optimal PWL function fitting a corridor \mathcal{C}

Require: Corridor \mathcal{C}
Ensure: PWL function g defined by the set of linear segments \mathcal{P}
 1: **while** $\mathcal{C} \neq \emptyset$ **do**
 2: $p^* \leftarrow$ Compute a maximum linear segment of \mathcal{C}
 3: $\mathcal{P} \leftarrow \mathcal{P} \cup p^*$
 4: $\mathcal{C} \leftarrow \mathcal{C} \setminus \mathcal{C}(p^*)$
 5: **end while**
 6: **return** \mathcal{P}

The correctness of the algorithm is proved in Sect. 3.2. We postponed the resolution of the maximal linear segment problem to Sect. 4. Section 5 mostly tackles clever ways to accelerate the Algorithm 1 for corridors with slightly more structure or when an almost-optimal approximation of the corridor fitting problem is sufficient. Finally, Sect. 6 is mainly constituted of computational results to show how our approach outperforms the state-of-the-art.

3.2 Optimality of a piecewise linearization with maximal linear segments

Proposition 1 and Corollary 1 extend the Theorem 3.3 from [13] (which only treated the special case of absolute errors) to the concept of corridors.

Proposition 1 *Given a corridor \mathcal{C} , if there exists a PWL function that fits \mathcal{C} , there exists an optimal solution of the corridor fitting problem on \mathcal{C} where the first segment is a maximal linear segment in \mathcal{C} .*

Proof Let \mathcal{C} be a corridor. Let g be any optimal PWL function that fits $\mathcal{C} = [x_-, x_+]$ and let $\{x_1, x_2, \dots, x_n\}$ be the break points of the linear segments of g . Note that $x_1 = x_-$ and $x_n = x_+$. Let $p : [x_-, x^*] \rightarrow \mathbb{R}$ be a maximal linear segment in \mathcal{C} . Let g^* be the PWL function defined by

$$g^*(x) = \begin{cases} p(x), & \forall x \in [x_-, x^*[\\ g(x), & \forall x \in [x^*, x_+]. \end{cases} \tag{2}$$

Note that, by definition, $x^* \geq x_2$. Therefore g^* has at most as many linear pieces as g which concludes the proof. □

Corollary 1 *Given a corridor \mathcal{C} , there exists an optimal solution to the corridor fitting problem such that each linear segment is a maximal linear segment on a truncated-corridor starting at the end of the previous segment (and for the whole corridor for the first segment).*

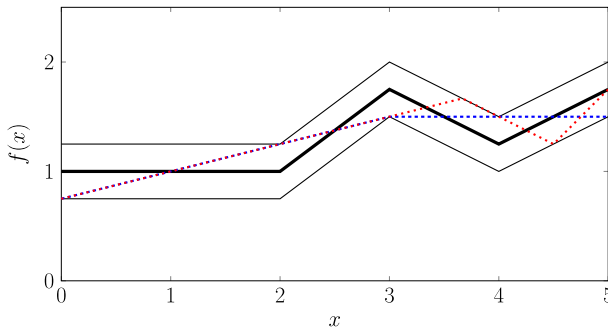


Fig. 2 Example of the non-optimality of the greedy algorithm in the continuous case (source: [16], p. 628)

Proof Proposition 1 can be applied iteratively on an optimal PWL function to ensure that each linear segment *induces* (see Definition 6) a *truncated-corridor* (see Definition 4) of maximal domain length. This works because this problem has an optimal substructure. In other words, if the optimal function is separated at the end of a segment, it will induce optimal solutions for the two *sub-corridors* (see Definition 3). \square

Corollary 1 proves that the corridor fitting problem can be solved with a greedy algorithm that computes a succession of maximal linear segment problems. This is a sharp contrast with what can be done with continuous PWL as in this case, the greedy algorithm is not guaranteed to give an optimal solution as shown in Fig. 2. Here, the blue curve is an optimal continuous PWL with 2 segments and the red curve is the continuous PWL with 3 segments obtained with a greedy algorithm.

3.3 Simultaneous under and over approximation

Given a function f and a maximal error, a natural extension of the problem at hand is to look for both an underestimator and overestimator PWL functions that share the same set of breakpoints. For absolute errors, this trivially reduces to the previous cases as one can obtain an optimal overestimate by shifting an underestimate. This is not the case for more general errors, even relative errors, as noted in [13]. However, the corollary 1 generalizes naturally to this case. Therefore, the problem reduces to finding maximal pairs of PWL functions, which can be done by maximizing both the underestimating segment and the overestimating one and truncating them so they have the same induced sub-corridor.

4 Maximal linear segment problem

In this section, we first tackle the maximal linear segment problem for convex corridors and then the general case. Even though the former is a special case of the latter, a dedicated algorithm was conceived for convex corridors because a strong character-

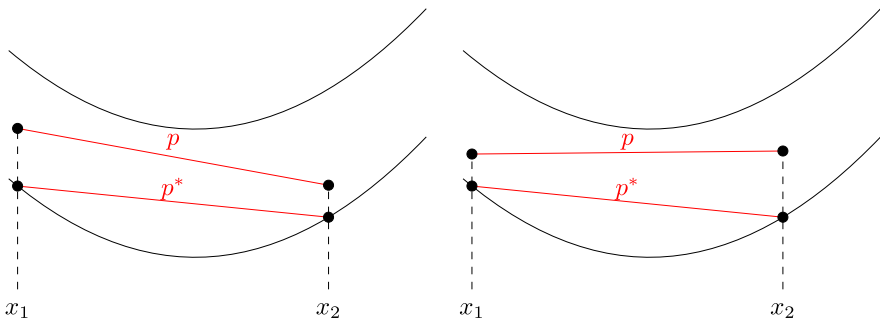


Fig. 3 Illustration of Lemma 1: two cases where a linear segment p^* intersecting the curve $\mathcal{L} = \{(x, l(x))\}$ at its breakpoint is derived from p with the same projected length $x_2 - x_1$

ization of the solution was found, which results in a major speedup over the general case.

4.1 Case of a convex or concave corridor

We consider a *convex corridor* $\check{\mathcal{C}}$ defined by functions $h, l : \mathbb{D} = [x_-, x_+] \rightarrow \mathbb{R}$ which are both continuously differentiable and convex. Note that all differentiable convex functions are C^1 , so the condition on the continuity of the derivative was only added for clarity. Notice that solving the maximal linear segment problem on a concave corridor $\hat{\mathcal{C}}$ defined by functions \hat{l}, \hat{h} reduces to the convex case as described above. Indeed, the corridor defined by the convex functions $-\hat{l}$ and $-\hat{h}$ is only a sign off and the solution to the original problem can easily be retrieved from a solution of this new instance by simply again multiplying by -1 . Lemmas 1 and 2 characterize a maximal linear segment in $\check{\mathcal{C}}$.

Lemma 1 *For any linear segment within $\check{\mathcal{C}}$, there exists a linear segment of equal domain length within $\check{\mathcal{C}}$ that intersects the curve $\mathcal{L} = \{(x, l(x))\}$ at its two endpoints.*

Proof Given any linear segment p within (see Definition 5) corridor $\check{\mathcal{C}}$ defined by endpoints (x_1, y_1) and (x_2, y_2) , we can define a linear piece of equal projected length p^* with its endpoints $(x_1, l(x_1))$ and $(x_2, l(x_2))$ on the curve \mathcal{L} . Because h, l are convex, this linear piece is within $\check{\mathcal{C}}$. Precisely, p^* must lie above \mathcal{L} as it is a line segment between two points on the graph of a convex function. Moreover, p^* must lie under $\mathcal{H} = \{(x, h(x))\}$ as $p^*(x) \leq p(x) \leq h(x)$ for every x . Therefore, p^* is within $\check{\mathcal{C}}$. \square

Fig. 3 illustrates the proof of Lemma 1.

Lemma 2 *A maximal linear segment in $\check{\mathcal{C}}$ either fits $\check{\mathcal{C}}$ or is tangent to the curve $\mathcal{H} = \{(x, h(x))\}$.*

Proof This assertion is proven by contradiction as follows. Let us assume that a segment p , defined by the points (x_1, y_1) and (x_2, y_2) , is a maximal linear segment in $\check{\mathcal{C}}$

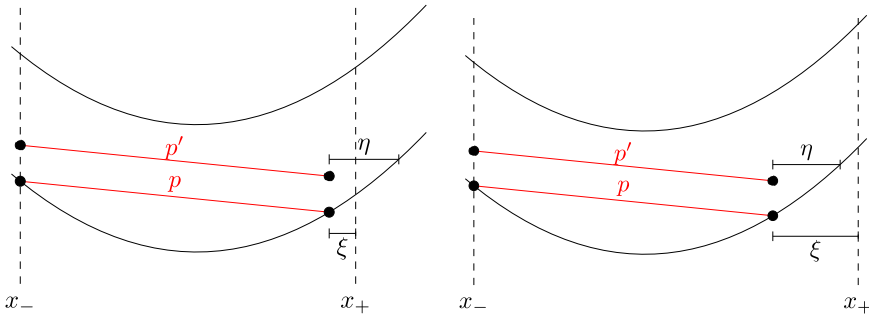


Fig. 4 Illustration of Lemma 2: case where $\min\{\xi, \eta\} = \xi$ and where $\min\{\xi, \eta\} = \eta$

that does not fit \check{C} and is not tangent to \mathcal{H} . Since p is not tangent to \mathcal{H} and clearly doesn't cross \mathcal{H} , there is no x such that $p(x) = h(x)$. Therefore, there exists $c \in \mathbb{R}$ such that $p' = p + c$ doesn't intersect \mathcal{H} , i.e., it is possible to do a vertical translation of p by c while still being within the corridor. For example, c could be taken as half the minimal vertical distance between p and \mathcal{H} . For the sake of simplicity, let us denote $y_1 + c$ by y'_1 , $y_2 + c$ by y'_2 , and the slope of p' by Δ . By construction, $(x_2, y'_2) \notin \mathcal{L}$. Also, since p and therefore p' does not fit \check{C} , there exists $\xi > 0$ such that $x_2 + \xi \leq x_+$. Finally, since none of the endpoints of p' lie on \mathcal{L} or \mathcal{H} , there exist $\eta > 0$ such that $\forall z \leq \eta, (x_2 + z, y'_2 + \Delta z) \notin \mathcal{L} \cup \mathcal{H}$ (i.e it is possible to extend p' without intersecting \mathcal{L} or \mathcal{H}). We can therefore construct the linear segment within \check{C} defined by (x_1, y'_1) and $(x_2 + \min\{\xi, \eta\}, y'_2 + \Delta \min\{\xi, \eta\})$, which has a domain length of $x_2 + \min\{\xi, \eta\} - x_1 > x_2 - x_1$, contradicting the optimality of p . \square

Fig. 4 illustrates the process in the proof of lemma 2.

Lemma 1 implies we can always choose $(x_-, l(x_-))$ as the starting endpoint of a maximal linear segment. Furthermore, Lemma 2 implies that an optimal segment will always be tangent to \mathcal{H} if it is impossible to fit the corridor with a single linear segment. By combining these two lemmas, we obtain a unique (in the non-degenerate case) characterization of the optimal linear segment (as illustrated in figure 5), which gives rise to an efficient algorithm.

Remark 3 As a consequence of this characterization, for any convex corridor, the PWL computed by algorithm 1 is continuous, so our algorithm also solves the continuous version of the problem in this case.

To find the desired linear segment, let us define $t(q, x)$ as the equation of the tangent to \mathcal{H} at q evaluated at point x . Formally,

$$t(q, x) = h'(q)(x - q) + h(q).$$

We are looking for a point q^* such that $t(q^*, x_-) = l(x_-)$ or equivalently, $t(q^*, x_-) - l(x_-) = 0$. Let

$$\Gamma(q) = t(q, x_-) - l(x_-).$$

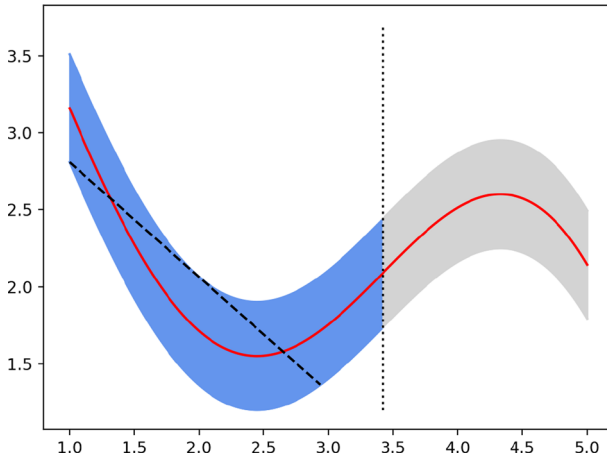


Fig. 5 Maximal linear piece on the convex sub-corridor derived from Fig. 1a

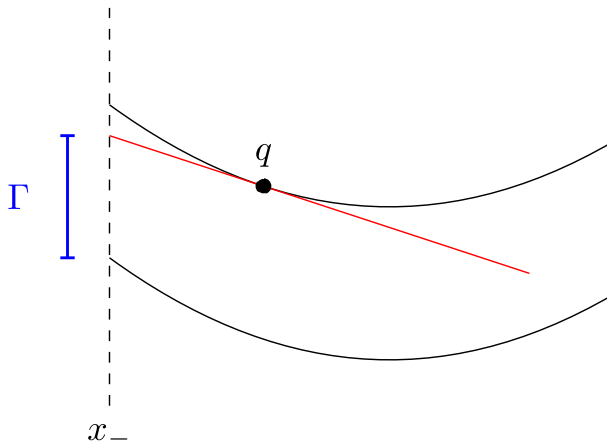


Fig. 6 Function $\Gamma(q)$

The geometric interpretation of $\Gamma(q)$ is illustrated in Fig. 6. Our interest in $\Gamma(q)$ is motivated by the fact that solving the maximal linear segment problem boils down to finding a zero of the decreasing function $\Gamma(q)$, which can be solved by binary search. We will prove this assertion in the following lemma.

Lemma 3 Γ is a decreasing function (strictly decreasing if h is strictly convex)

Proof Without loss of generality, we'll assume that $x_- = 0$. Since $\Gamma(q)$ is only a constant off $t(q, 0)$, it is sufficient to prove that $t(q, 0)$ is decreasing. Let $a, b \in \mathbb{R}$ such that $0 < a < b$. We will show that $t(a, 0) \geq t(b, 0)$.

$$\begin{aligned} t(b, 0) &= h'(b)(-b) + h(b) \\ &= h'(b)(-a) + h'(b)(a - b) + h(b) \end{aligned}$$

$$= h'(b)(-a) + t(b, a)$$

Since the tangent of a convex function is always below the curve, we have that $t(b, a) \geq h(a)$.

$$\begin{aligned} &\leq h'(b)(-a) + h(a) \\ \xrightarrow{\text{Convexity}} &\leq h'(a)(-a) + h(a) \\ &= t(a, 0) \end{aligned}$$

□

Once the optimal tangent point is found by finding the zero of Γ , we only need to find the second endpoint. This is done by computing where the line supporting the linear segment crosses \mathcal{L} again using binary search since there will be only one other crosspoint and since the difference between a linear function and a convex function is still convex.

The full procedure is summarized in Algorithm 2. As the only costly step of this algorithm is the binary search, our algorithm only needs $O(\log(1/\varepsilon))$ oracle calls to the function h, l and their derivatives where ε is the numerical precision. Since, in most cases, the time taken to evaluate h and l does not depend on its input, our algorithm can be seen as taking logarithmic time. The logarithmic time complexity of this approach is to be stressed, as it is the main reason behind the implementation's efficiency.

Algorithm 2 Solve the maximal linear piece problem on a convex corridor

Require: Convex corridor \mathcal{C}_v defined by convex functions $h, l : \mathbb{D} = [x_-, x_+] \rightarrow \mathbb{R}$

Ensure: Optimal linear piece defined by breakpoints (x_1, y_1) and (x_2, y_2)

```

1:  $(x_1, y_1) \leftarrow (x_-, l(x_-))$ 
2: if  $\Gamma(x_+) \geq 0$  then
3:    $(x_2, y_2) \leftarrow (x_+, h(x_+))$ 
4: else
5:    $q^* \leftarrow$  Solve  $\Gamma(q) = 0$  using binary search
6:    $p \leftarrow$  linear segment defined by  $(x_1, y_1)$  and  $q^*$ 
7:   if  $p(x_+) > l(x_+)$  then
8:      $(x_2, y_2) \leftarrow (x_+, l(x_+))$ 
9:   else
10:     $x^* \leftarrow$  Solve  $p(x) - l(x) = 0$  for  $x > x_-$  using binary search
11:     $(x_2, y_2) \leftarrow (x^*, l(x^*))$ 
12:   end if
13: end if
14: return  $(x_1, y_1), (x_2, y_2)$ 

```

As previously mentioned, Algorithm 2 applies to both the convex and the concave cases if we pre-process and post-process concave instances. Adding these processing steps gives us an algorithm which we will designate by 2'.

4.2 General case based on converting corridors into data ranges

An iterative procedure solves the maximal linear segment problem in the general case. The procedure builds a discretized variant of the problem, solves it using a state-of-the-art algorithm on the discretized version of the problem, and then retrieves a solution candidate for our continuous problem. Finally, it updates the discretization and repeats the last two steps until the solution obtained is feasible and optimal for the initial problem.

To obtain the discrete variant of the problem, the interval is discretized into a finite set of points \mathcal{X} . Any valid linear segment within a corridor \mathcal{C} has to at least verify Eq. (3) on a consecutive subset of \mathcal{X} . The discrete variant of the maximal linear segment problem consists of finding a linear segment that verifies Eq. (3) for a maximal number of consecutive points $x_i \in \mathcal{X}$ including the first one x_1 .

$$l(x_i) \leq mx_i + b \leq h(x_i) \quad (3)$$

This problem is known in the literature as data fitting or fitting a straight line between data ranges. For such discrete inputs, there exist publications on piecewise linear approximation with a minimum number of segments given a predefined bound on the absolute error in the fields of data reduction, pattern recognition or classification, and ECG waveform preprocessing [7, 21, 22].

A state-of-the-art algorithm for this problem is the one of O'Rourke [14]. It works by considering each Eq. (3) as a pair of constraints (4) in the m - b parameter space as illustrated in Fig. 7. These constraints define a polyhedron denoted P_k for the k first consecutive points of \mathcal{X} . If the resulting polyhedron P_k is empty, then no single line can verify Eq. (3) for the k first points of \mathcal{X} . Otherwise, any point inside or on the boundary of P_k represents a valid line. The resulting algorithm computes P_n with an $O(n)$ complexity (for n input points).

The algorithm stops if either $k = n$ if all points have been covered or $P_{k+1} = \emptyset$ if k is the maximal number of consecutive points of \mathcal{X} that can be covered and therefore $(m, b) \in P_{k-1}$.

$$\begin{cases} b \leq (-x_i)m + h(x_i) \\ b \geq (-x_i)m + l(x_i) \end{cases} \quad (4)$$

Let (m, b) and k be the straight line parameters and the number of intersected data ranges for the optimal solution of the discretized maximal linear segment problem. Verifying if the solution is feasible for the original non-discretized maximal linear segment problem consists in verifying if the linear segment f defined by $(x_1, mx_1 + b)$ and $(x_k, mx_k + b)$ fits within the truncated-corridor \mathcal{C}_k with domain $\mathbb{D}_k = [x_-, x_k]$. If the solution is not feasible, then at least one of the points that are on the linear segment but out of the truncated-corridor should be added to \mathcal{X} before the discretized problem is solved again. In our implementation, we compute all intersections between \mathcal{L} and f , i.e. we solve $l(x) - mx - b = 0$ with $x_1 \leq x \leq x_k$. Then for any two consecutive intersection points $(x_A, l(x_A))$ and $(x_B, l(x_B))$, we check if $](x_A, l(x_A)), (x_B, l(x_B))$

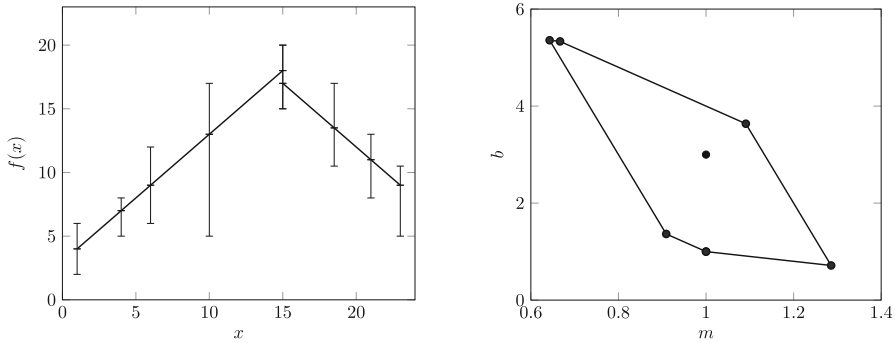


Fig. 7 Example from [14] with $n = 8$ data ranges and the polyhedron in the m - b space corresponding to the first five data ranges

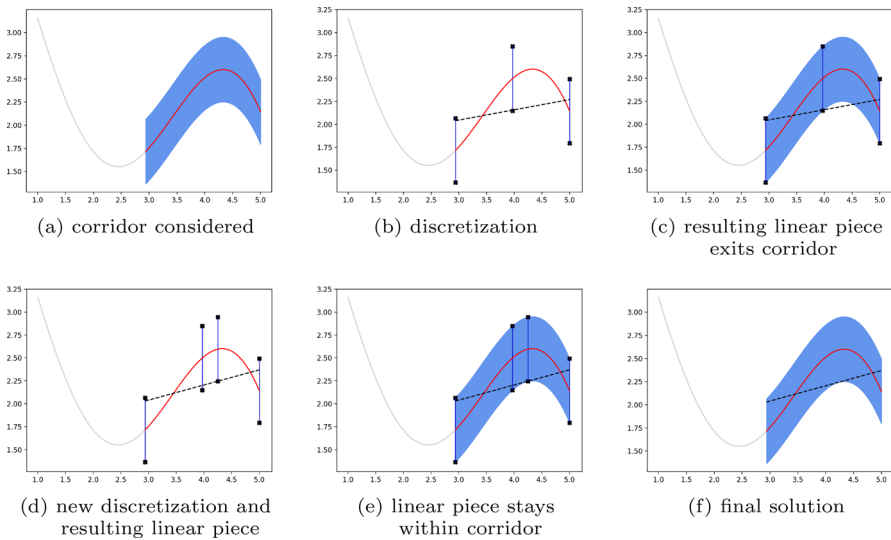


Fig. 8 Visualization of Algorithm 3

is out of the corridor. If this is the case, the middle-point of the interval $\frac{x_A+x_B}{2}$ is added to \mathcal{X} before the discretized problem can be solved again. The same procedure is repeated for possible intersections with \mathcal{H} . If no point was added during this process, then the solution to the discretized problem is feasible for the non-discretized problem. With such an implementation, only an LP solver is needed, and a drastic reduction in computing times can be achieved compared to state-of-the-art solutions.

When the solution of the discretized problem is feasible for the non-discretized problem, it is optimal within a numerical precision level of $x_{k+1} - x_k$. Let ε be a target numerical precision level. If $x_{k+1} - x_k > \varepsilon$, then the point $\frac{x_k+x_{k+1}}{2}$ is added to \mathcal{X} , before re-solving the discretized problem. Note that in the latter case, as a speed-up, O'Rourke's algorithm for solving the discretized problem at step $k + 1$ can be warm started with the polyhedron P_k avoiding the unnecessary overhead of recomputing P_k .

The resulting iterative procedure solving the maximal linear segment problem in the general case is described in Algorithm 3 and illustrated in Fig. 8. In an attempt to not obfuscate Algorithm 3, the warm start discussed above is not included but is straightforward to implement.

This algorithm always converges since, in the worst case, the corridor will be sampled every ε , in which case the discretized problem becomes equivalent to the continuous one (within the numerical precision) for which the algorithm from [14] always terminates. We mention that we did not observe this worst-case behavior in practice and that the algorithm seems to converge quite rapidly (while remaining significantly slower than algorithm 2') as illustrated by the benchmarks in Sect. 6.2.

The initial discretization of the corridor has a non-negligible impact on the run time of the algorithm. We could, therefore, try to estimate where the function varies the most to help the algorithm converge faster. However, we are not assuming differentiability in this section but rather looking for a general solution; therefore, we simply use equidistant points to initialize our algorithm.

Algorithm 3 Solve the general maximal linear segment problem

Require: Corridor \mathcal{C} defined by functions $h, l : \mathbb{D} = [x_-, x_+] \rightarrow \mathbb{R}$

Ensure: Optimal linear segment defined by endpoints (x_S, y_S) and (x_T, y_T)

```

1:  $\mathcal{X} \leftarrow$  discretize corridor  $\mathcal{C}$ 
2: FEASIBLE  $\leftarrow$  true
3:  $\{m, b, k\} \leftarrow$  O'Rourke's algorithm applied on  $\mathcal{X}$ 
4:  $\mathcal{I} \leftarrow$  Solve  $l(x) - mx - b = 0$ 
5:  $\mathcal{I} \leftarrow \mathcal{I} \cup$  Solve  $h(x) - mx - b = 0$ 
6: for all  $(x_A, x_B)$  a set of consecutive values in  $\mathcal{I}$  do
7:   if  $m \frac{x_A+x_B}{2} + b \notin [l(\frac{x_A+x_B}{2}), h(\frac{x_A+x_B}{2})]$  then
8:      $\mathcal{X} \leftarrow \mathcal{X} \cup \{\frac{x_A+x_B}{2}\}$ 
9:     FEASIBLE  $\leftarrow$  false
10:  end if
11: end for
12: if FEASIBLE then
13:   if  $x_{k+1} - x_k > \varepsilon$  then
14:      $\mathcal{X} \leftarrow \mathcal{X} \cup \{\frac{x_{k+1}+x_k}{2}\}$ 
15:     go to step 2
16:   else
17:     return Segment induced by  $x_-, x_k, m, b$ 
18:   end if
19: end if

```

5 Speed-ups for uniform corridors

In this section, we look at how the previous algorithms for the maximal linear segment problem can be combined to obtain faster convergence on the corridor fitting problem in the special case of *uniform corridor*. A corridor is said to be uniform if it is defined by C^1 functions having the same concavity on \mathbb{D} , i.e., at every $x \in \mathbb{D}$, the functions h and l are either both convex or both concave.

Remark 4 In the case of C^2 functions, the concavity condition can simply be stated as $l''(x)h''(x) \geq 0 \forall x \in \mathbb{D}$

This class of corridors includes approximations from relative and absolute pointwise errors. These types of errors are ubiquitous in the optimization community and cover most of the instances seen in practice.

5.1 A faster exact method

The corridor fitting problem can be solved on a uniform corridor using Algorithm 1 with Algorithm 3 as a subroutine, but a significant speed-up can be obtained by making use of the specific structure of uniform corridors. To do so, Algorithm 2' can be used on the convex and concave sub-corridors of \mathcal{C} , while Algorithm 3 is only used on sub-corridors that contain a change of concavity. The idea is to apply Algorithm 2' on \mathcal{C} until a change of concavity is reached. Then remove the last segment computed and use Algorithm 3 to compute the maximal linear piece that crosses the change of concavity. Algorithm 2' is then called again until the next change of concavity. This is summarized in Algorithm 4. The main advantage of this approach is that it utilizes the logarithmic time complexity of Algorithm 2' for most of the domain while only using the slower but more general Algorithm 3 when it is really needed.

Algorithm 4 Faster exact computation of an optimal PWL function fitting corridor \mathcal{C}

Require: Uniform corridor \mathcal{C}

Ensure: PWLfunction g defined by the set of linear segments \mathcal{P}

```

1: while  $\mathcal{C} \neq \emptyset$  do
2:    $\{\mathcal{C}_1, \dots, \mathcal{C}_k\} \leftarrow$  Partition into convex or concave sub-corridors of  $\mathcal{C}$ 
3:   while  $\mathcal{C}^c \neq \emptyset$  do
4:      $p^* \leftarrow$  Algorithm 2'( $\mathcal{C}^c$ )
5:     if  $\mathcal{C}^c(p^*) \neq \mathcal{C}^c$  or  $\mathcal{C}^c(p^*) = \mathcal{C}$  then
6:        $\mathcal{P} \leftarrow \mathcal{P} \cup p^*$ 
7:        $\mathcal{C} \leftarrow \mathcal{C} \setminus \mathcal{C}^c(p^*)$ 
8:     end if
9:      $\mathcal{C}^c \leftarrow \mathcal{C}^c \setminus \mathcal{C}^c(p^*)$ 
10:  end while
11:  if  $\mathcal{C} \neq \emptyset$  then
12:     $p^* \leftarrow$  Algorithm 3( $\mathcal{C}$ )
13:     $\mathcal{P} \leftarrow \mathcal{P} \cup p^*$ 
14:     $\mathcal{C} \leftarrow \mathcal{C} \setminus \mathcal{C}(p^*)$ 
15:  end if
16: end while

```

In our implementation, the positions of the changes of concavity can either be provided by the user at the same time as the corridor or left to be computed by a dedicated algorithm. We have implemented a dedicated algorithm that requires the corridor's bounding functions to be C^2 instead of C^1 . It finds the positions of changes of concavity by computing where the second derivatives of the bounding functions vanish.

5.2 A heuristic introducing a very limited number of additional segments

In practice, a *good* solution to the corridor fitting problem is often sufficient for one’s needs. Here we describe an algorithm that adds at most as many additional segments as the number of change of concavities in the corridor but, as a trade-off, converges faster.

Let \mathcal{C} be a uniform corridor we are looking to fit. \mathcal{C} can be partitioned into k sub-corridors $\mathcal{C}_1, \dots, \mathcal{C}_k$ such that each sub-corridor $\mathcal{C}_i, \forall i \in \{1..k\}$ is either convex or concave. A feasible solution to the corridor fitting problem on \mathcal{C} is obtained by concatenating optimal solutions for all the sub-corridors’ corridor fitting problems. On each sub-corridor, we can use Algorithm 2’ to compute the maximal linear segments. This is summarized in Algorithm 5.

Algorithm 5 Heuristic computation of a PWL function fitting a uniform corridor \mathcal{C}

Require: Uniform corridor \mathcal{C}

Ensure: PWL function g defined by the set of linear pieces \mathcal{P}

```

1:  $\{\mathcal{C}_1, \dots, \mathcal{C}_k\} \leftarrow$  PARTITION INTO CONVEX OR CONCAVE SUB-CORRIDORS OF( $\mathcal{C}$ )
2: for  $i \in \{1..k\}$  do
3:   while  $\mathcal{C}_v \neq \emptyset$  do
4:      $p^* \leftarrow$  Algorithm 2’ ( $\mathcal{C}_v$ )
5:      $\mathcal{P} \leftarrow \mathcal{P} \cup p^*$ 
6:      $\mathcal{C}_v \leftarrow \mathcal{C}_v \setminus \mathcal{C}(p^*)$ 
7:   end while
8: end for
    
```

Here, similarly as in algorithm 4, the changes of concavity can be either provided by the user or computed by looking where the second derivatives are zero.

The advantage of this approach is that it is possible to compute the linearization in each sub-corridor in parallel. Also, the implementation is simpler and quicker as it only relies on Algorithm 2’, which has a logarithmic time complexity for every segment. The drawback is that the number of segments might not be optimal as there are segments that are not maximal at the junction of sub-corridors. However, the number of additional segments is tightly bounded as stated in Lemma 4.

Lemma 4 *Let \bar{n} be the number of linear segments from Algorithm 5 and let n^* be the optimal number of linear segments, then $n^* \leq \bar{n} \leq n^* + k - 1$.*

Proof Let \mathcal{C} be a uniform corridor over $\mathbb{D} = [x_-, x_+]$. Let B_1, B_2, \dots, B_{k-1} represent the changes of concavity of \mathcal{C} , $B_0 = x_-$ and $B_k = x_+$. \mathbb{D} can be partitioned as $\bigcup_{i=1}^{k-1} [B_{i-1}, B_i] \cup [B_{k-1}, B_k]$ such that the corridor is either convex or concave on each interval. Let g^* be the optimal PWL function fitting corridor \mathcal{C} and having as breakpoints x_0, \dots, x_{n^*} and f be the PWL function obtained by Algorithm 5. Let S_i be the minimal x_j such that $x_j \geq B_i$. Finally, let ν be the counting function for the number of segments in a piecewise linear function (so that we have $\nu(g^*) = n^*$). To make the notation less cumbersome, $\nu(g^*)$ will not count a segment made of a single point so that we can work with closed intervals. Since $[B_{i-1}, B_i]$ is either convex or

concave, $v(f|_{[B_{i-1}, B_i]})$ is optimal over this restricted domain and we therefore have

$$v(f|_{[B_{i-1}, B_i]}) \leq v(g^*|_{[B_{i-1}, B_i]}), \quad \forall i \in \{1, \dots, k\}.$$

Since f and g^* share the same number of segments before the first change in concavity, we have equality for $i = 1$. Let us analyze the case where $i \neq 1$. The minimality of S_{i-1} implies that the interval $[B_{i-1}, S_{i-1}]$ is covered by a single segment in g^* . Therefore,

$$v(f|_{[B_{i-1}, B_i]}) \leq v(g^*|_{[B_{i-1}, B_i]}) \leq v(g^*|_{[S_{i-1}, B_i]}) + 1, \quad \forall i \in \{1, \dots, k\}.$$

Noting that $v(f|_{[B_{i-1}, B_i]}) + v(f|_{[B_i, B_{i+1}]}) = v(f|_{[B_{i-1}, B_{i+1}]})$ as there is no segment overlapping between two sub-corridors by the definition of the algorithm, we have that

$$\begin{aligned} v(f) &= \sum_{i=1}^k v(f|_{[B_{i-1}, B_i]}) \\ &= v(f|_{[B_0, B_1]}) + \sum_{i=2}^k v(f|_{[B_{i-1}, B_i]}) \\ &= v(f|_{[x_-, B_1]}) + \sum_{i=2}^k v(f|_{[B_{i-1}, B_i]}) \\ &\leq v(f|_{[x_-, B_1]}) + \sum_{i=2}^k v(g^*|_{[S_{i-1}, B_i]}) + 1 \end{aligned}$$

Since the segment of g^* fitting $[B_i, S_{i+1}]$ is also contributing to $g^*|_{[S_i, B_i]}$,

$$\begin{aligned} &= v(f|_{[x_-, S_1]}) + \sum_{i=2}^k v(g^*|_{[S_i, S_{i+1}]} + 1 \\ &= k - 1 + v(f|_{[x_-, S_1]}) + \sum_{i=2}^k v(g^*|_{[S_i, S_{i+1}]}) \\ &= k - 1 + \sum_{i=1}^k v(g^*|_{[S_i, S_{i+1}]}) \\ &= k - 1 + n^* \end{aligned}$$

□

In addition to providing a guarantee on the quality of PWL obtained, Lemma 4 gives an easy way to compute lower bounds for the number of linear segments needed to solve the corridor fitting problem optimally.

To illustrate the differences between both algorithms presented in this section, Fig. 9 compares the solution obtained by the Algorithms 5 and 4 on the corridor derived from

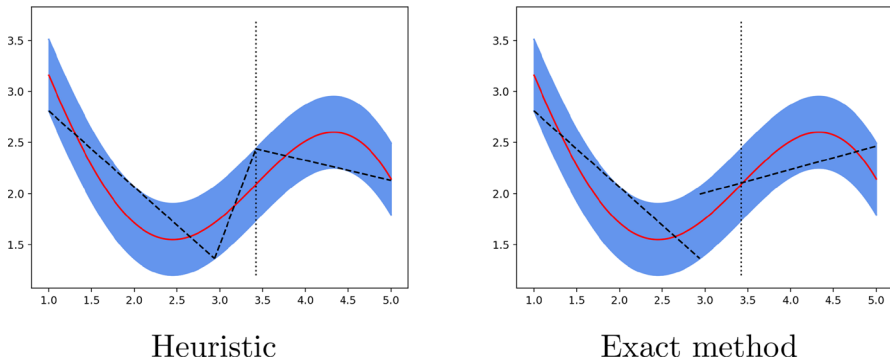


Fig. 9 Comparison between both approaches

an absolute tolerance of 0.3 of $\sqrt{x} \sin x + x$ on the interval $[1, 5]$. The vertical dotted line marks the concavity change at $x \approx 3.42$.

Remark 5 As a consequence of Lemma 1, the PWL computed on uniform corridors are continuous on each convex or concave sub-corridor, and the only possible discontinuities occur in the junctions of sub-corridors. Moreover, the PWL are convex (resp. concave) functions on these convex (resp. concave) sub-corridors.

6 Computational evaluation

6.1 Implementation

The package proposed is named LINA [3] and is compatible with Julia 1.0.1 and onward. It is available at the following address: <https://github.com/LICO-labs/LinA.jl> under the MIT License. It accepts as input the expression of a function or a native Julia function, the type of error requested (relative or absolute), and optionally the type of over-/under-/approximation expected (by default approximation) and the algorithm desired (by default heuristic). It is also possible to provide directly the two functions defining the corridor to fit or even to define custom pointwise error types. Simultaneous approximations (see Sect. 3.3) are also implemented.

One of the interesting features of the implementation is the use of symbolic differentiation through the package CALCULUS.JL (<https://github.com/JuliaMath/Calculus.jl>) when function expressions are provided. When a symbolic expression is hard to obtain, the user can replace the expression with a native Julia function. In this case, the LINA will seamlessly use automatic differentiation through the ForwardDiff.jl package [18] instead. This is achieved through the extensive use of multiple dispatch, which gives a very modular code and, in addition, allows the addition of custom error types easily (for the exact procedure, see the documentation). Another perk is the use of functors¹ to make the syntax as natural as possible as shown in Fig. 10.

¹ See <https://docs.julialang.org/en/v1/manual/methods/>.

```

julia> using LinA

julia> f = Linearize(:(x^2+1),-3,3,Absolute(0.1))
7-element Vector{LinA.LinearPiece}:
-5.105572809000085 x -5.416718427000255 from -3.0 to -2.1055728090000847
-3.316718427000254 x -1.6501552810007603 from -2.1055728090000847 to -1.2111456180001692
-1.5278640450004222 x + 0.5164078649987369 from -1.2111456180001692 to -0.31671842700025316
0.26099033699940966 x + 1.0829710109982338 from -0.31671842700025316 to 0.577708763999663
2.049844718999242 x + 0.04953415699772967 from 0.577708763999663 to 1.4721359549995794
3.838699100999076 x -2.5839026970027774 from 1.4721359549995794 to 2.3665631459994967
5.627553482998908 x -6.817339551003287 from 2.3665631459994967 to 3.0

julia> f[1]
-5.105572809000085 x -5.416718427000255 from -3.0 to -2.1055728090000847

julia> f(2)
5.093495504995374

julia> f[1].xMax
-2.1055728090000847

```

Fig. 10 Simple example of the LINA syntax

The current implementation of LINA relies on the root finding functions available in the package `ROOTS.JL` (<https://github.com/JuliaMath/Roots.jl>). In particular, it uses its binary search function. The function `FIND_ZEROS` is used to identify the location of changes of concavity for C^2 uniform corridor by computing the points where the second derivative of the input function vanishes but also in other contexts, such as finding the intersection between a line and the corridor in Algorithm 3 (used in the exact method). Also, the `POLYHEDRA.JL` package [11] is used to compute the intersections of polyhedrons for Algorithm 3.

6.2 Benchmarks in approximations of nonlinear functions

To assess the efficiency of LINA we perform a computational study on thirteen non-linear continuous functions from the literature provided in Table 2. We then compare the results obtained by LINA with the ones reported by [10, 13, 16, 17].

Even though comparing continuous approximations with our non necessarily continuous ones is not ideal in the general case, we compared our algorithms with all the exact algorithms available in the literature on solving the problem of minimizing the number of linear pieces with a bound on the pointwise approximation error. In addition, some of our benchmarks were on convex or concave functions for which our algorithm gives a continuous approximation, as stated in a remark in Sect. 4.1. Moreover, it is possible to transform a PWL from LinA to a potentially non-optimal continuous one by adding a small segment at each discontinuity of which there are few (at most as many as the number of changes in concavity as mentioned in a remark at the end of Sect. 5). Therefore, LinA can be used to derive good upper bounds on the continuous version of the problem.

All of our benchmarks ran on a consumer's desktop. More precisely, Table 3 shows the characteristics of the machines used by the different authors.

To illustrate the usefulness of these methods over non-optimal linearisation approaches in the reduction of segment obtained, we used two widely used non-optimal linearization algorithms as a baseline. More precisely, we used the "naive"

Table 2 Benchmark: univariate nonlinear functions and tolerance types used

Reference	Function	Domain	Error	Tol.*
(A-I)	x^2	$[-3.5; 3.5]$	Absolute	a
(A-II)	$\ln(x)$	$[1; 32]$	Absolute	a
(A-III)	$\sin(x)$	$[0; 2\pi]$	Absolute	a
(A-IV)	$\tanh(x)$	$[-5; 5]$	Absolute	a
(A-V)	$\frac{\sin(x)}{x}$	$[1; 12]$	Absolute	a
(A-VI)	$2x^2 + x^3$	$[-2.5; 2.5]$	Absolute	a
(A-VII)	$e^{-x} \sin(x)$	$[-4; 4]$	Absolute	a
(A-VIII)	$e^{-100(x-2)^2}$	$[0; 3]$	Absolute	a
(A-IX)	$1.03e^{-100(x-1.2)^2} + e^{-100(x-2)^2}$	$[0; 3]$	Absolute	a
(A-X)	Maranas and Floudas (1994)	$[0, 2\pi]$	Absolute	a
(R-I)	$-0.005x^3 + 0.5x^2 - 0.8x + 10.0$	$[1; 60]$	Relative	a, o, u
(R-II)	$0.001x^3 - 0.024x^2 + 1.92x + 5.91$	$[1; 60]$	Relative	a, o, u
(R-III)	$0.000002x^5 - 0.0000274x^4 + 0.00151450x^3 - 0.02453270x^2 + 1.92434870x + 5.90568630$	$[1; 60]$	Relative	a, o, u

*Tolerance: (a)pproximation, (o)verfitting, (u)nderestimation

Table 3 Parameter settings per publication

[16]	Computer	Intel(R) i7 single core 2.93 GHz, 12.0 GB RAM 64-bit Windows 7
	single node ?	yes
	Passmark cpu score	5366
	Software	GAMS 23.6, LindoGlobal 23.6.5
	Accuracy	10^{-5}
	Time limit	1800s per iteration
[17]	Computer	Intel 3.5 GHz, 32 GB of RAM
	single node ?	not specified
	Passmark cpu score	between 8687 and 28594
	Software	GAMS 24.8, LindoGlobal (for GO), CPLEX (for MILP)
	Accuracy	10^{-3}
	Time limit	24 h per instance
[13]	Computer	Intel(R) Xeon(R) single core CPU E3- 1271 v3, 32 GB RAM (heuristic) Neos server (exact)
	single node ?	yes
	Passmark cpu score	10086 (heuristic); 6878, 7641, 21149 (exact)
	Software	GAMS 23.6, GAMS 24.9.2 r64480 single core, LindoGlobal (for GO), CPLEX 12.6 (for MILP)
	Accuracy	10^{-5}
	Time limit	1800s per iteration
[10]	Computer	not specified
	single node ?	not specified
	Passmark cpu score	not specified
	Software	GAMS (25.0.1), CPLEX (12.8.0.0), ANTIGONE v1.1, BARON 17.10.16, SCIP 4.0
	Accuracy	not specified
	Time limit	1200s
Ours	Computer	Intel(R) Core(TM) i7-7700 CPU 3.60GHz, 16 GB of RAM
	single node ?	yes
	Passmark cpu score	8625
	Software	LINA
	Accuracy	10^{-5}
	Time limit	1800s

algorithm based on interpolation of equidistant function samples and the sandwich algorithm with the maximum error rule (see [20]) on each convex and concave part of the domain. Since the speed of these linearisations was not our focus, we omitted to report the time taken by our implementations as much improvement could easily be made.

6.2.1 Absolute errors

Table 4 reports the number of linear pieces for the piecewise linear approximation with four different absolute tolerance values, which were previously used in the literature: $\delta = 0.1, 0.05, 0.01,$ and 0.005 . Some papers did not report results for every instance. For the instances not available, we left the corresponding entry of the table blank.

In this table, p is the number of convex or concave subintervals over the domain, δ is the absolute tolerance value used, and n^* is the optimal number of linear pieces. For our heuristic and for algorithms that did not always converge within the time limit, we give n_- (resp. n_+), which is the lower (resp. upper) bound returned by each algorithm. The symbol “-” means that the lower bound matches the upper bound (and so the algorithm converged). For our heuristic, Lemma 4 implies a lower bound of $n_- = n_+ - p + 1$, where n_+ is the number of linear pieces of the feasible solution returned by Algorithm 5.

Results in Table 4 suggest that [10] may not be competitive with the other solution methods. Rebennack and Krasko [17] improved the results from [16], but there still remain instances for which the optimal PWL approximation could not be found, contrary to [13] and LINA. One interesting remark is that the optimal number of segments of a continuous PWL function is at least as large as the number of segments of an optimal general PWL function. Because of this simple fact, our algorithms managed to improve some lower bounds from the literature on the continuous version of the problem. For example, for function (A-X) with $\delta = 0.005$, our exact algorithm shows that 65 is a valid lower bound for the continuous problem while the best bound from the literature was 50. Note that in this case, the lower bound provided by our heuristic was equal to 64, also better than the best ones from the literature.

Table 5 reports the computing times to obtain those results. Rebennack and Kallrath [16] reported only the order of magnitude of the computing time for instances that could be solved optimally. In their scale *frac* means “ ≤ 1 ”, *few* means “ ≥ 1 and ≤ 10 ” seconds. The instances with t.o.^{1800s/it} (resp. t.o.^{1200s}) are the ones for which the 1800 (resp. 1200) seconds time limit was reached during one of the iterations (resp. in total).

Results in Table 5 show that a major speed-up is achieved: approximations that required hours of calculations in [16, 17] or hundred of seconds in [13] are done in one-tenth of a second with LINA.

6.2.2 Relative errors

Similarly, Tables 6 and 7 report the number of segments and computing times for the piecewise linear overestimation, underestimation and approximation of the three non-linear functions that represent energy conversion functions for hybrid electric vehicles in [13], with three different relative tolerance values: $\varepsilon = 0.01, 0.001,$ and 0.0001 .

Table 4 Number of linear segments obtained with absolute tolerance δ

Function	δ	Continuous approximation				Non necessarily continuous approximation			
		Naive [16]		[17]		Sandwich		LINA heuristic	
Reference	p	n_+	n_-	n_+	n_-	n_+	n_-	LINA exact n^*	LINA heuristic n_+
(A-I)	1	12	-	8	-	10	-	8	8
		0.050	16	12	-	18	-	12	12
		0.010	35	25	-	34	-	25	25
		0.005	50	35	-	66	-	35	35
(A-II)	1	22	-	3	-	6	-	3	3
		0.050	35	4	-	8	-	4	4
		0.010	71	9	-	14	-	9	9
		0.005	79	13	-	20	-	13	13
(A-III)	2	7	-	5	-	8	-	5	6
		0.050	10	5	-	12	-	5	6
		0.010	23	13	-	20	-	13	14
		0.005	32	17	-	28	-	17	18
(A-IV)	2	5	-	3	-	8	-	4	4
		0.050	14	5	-	8	-	5	6
		0.010	30	9	-	14	-	9	10
		0.005	44	12	-	22	-	13	14
(A-V)	4	6	-	3	-	10	-	3	5
		0.050	9	5	-	10	-	4	6
		0.010	20	9 [†]	-	18	-	8	10
		0.005	27	15 [†]	-	26	-	12	15

Table 4 continued

Function	δ	p	Continuous approximation				Non necessarily continuous approximation							
			Naive [16]		[17]		Sandwich		LINA exact		LINA heuristic			
Reference			n_+	n_-	n_+	n_-	n_+	n_-	n_+	n_-	n_+	n_-	n_+	n_-
(A-VI)	0.100	2	24	-	11	-	11	17	11	11	12	11	12	11
	0.050		35	-	15	-	15	28	15	15	16	15	16	15
	0.010		77	15	34 [†]	31	34 [†]	53	34	34	35	34	35	34
(A-VII)	0.005		109	15	47 [†]	40	47 [†]	67	47	47	48	47	48	47
	0.100	3	75	4	14 [†]	-	14	26	14	14	16	14	16	14
	0.050		105	4	19 [†]	-	19	29	19	19	21	19	21	19
(A-VIII)	0.010		232	4	43 [†]	34	43 [†]	66	43	43	45	43	45	43
	0.005		264	4	61 [†]	35	61 [†]	93	61	61	63	61	63	61
	0.100	3	18	4	4 [†]	-	4	8	4	4	6	4	6	4
(A-IX)	0.050		58	4	6 [†]	-	5	12	5	5	6	5	6	5
	0.010		94	4	11 [†]	-	11	18	11	10	12	10	12	10
	0.005	5	158	4	14 [†]	-	14	24	14	14	16	14	16	14
(A-X)	0.100		42	7	7 [†]	-	7	16	7	7	11	7	11	7
	0.050		58	7	11 [†]	-	9	22	9	7	11	7	11	7
	0.010		103	7	21 [†]	-	21	34	21	19	23	19	23	19
(A-X)	0.005		173	7	28 [†]	-	27	50	27	27	31	27	31	27
	0.100	7	32	3	16 [†]	-	16	36	15	13	19	13	19	13
	0.050		45	3	22 [†]	-	21	38	20	18	24	18	24	18
(A-X)	0.010		102	3	45 [†]	42	45 [†]	82	45	45	51	45	51	45
	0.005		145	3	66 [†]	50	66 [†]	108	65	64	70	64	70	64

Bold indicates optimal number of linear segments

†: Best solution found before timeout

Table 5 Cpu times from [13, 16, 17] and LINA for the piecewise linear approximation of nonlinear functions with absolute tolerance δ

Function Reference	p	δ	Continuous approximation		Non necessarily continuous approximation		LINA		
			[16]	[17]	[10]	[13]	Exact	Heuristic	Exact
(A-I)	1	0.1	Few sec.		350s			0.003s	1.5 ms
		0.05	Few sec.		t.o. ^{1200s}			0.003s	1.7 ms
		0.01	Hours				19s	0.054s	4.3 ms
		0.005	Hours				4s	0.59s	4.2 ms
		0.1	Frac. sec.	0.1s	26s			0.006s	1.3 ms
(A-II)	1	0.05	Few sec.	0.1s	t.o. ^{1200s}			0.003s	1.6 ms
		0.01	Sec	9.8s			221s	0.002s	1.9 ms
		0.005	Sec	128.2s			172s	0.002s	1.6 ms
		0.1	Few sec.					0.12s	2.0 ms
		0.05	Few sec.					0.056s	2.1 ms
(A-III)	2	0.01	Sec				57s	0.09s	2.8 ms
		0.005	Few min				68s	0.058s	2.4 ms
		0.1	Frac sec.					0.10s	2.6 ms
		0.05	Few sec					0.039s	2.6 ms
		0.01	Few sec				161s	0.085s	2.9 ms
(A-IV)	2	0.005	Few min				128s	0.040s	2.8 ms
		0.1	Frac sec.	0.7s				0.40s	6.6 ms
		0.05	Sec.	6.5s				0.37s	2.6 ms
		0.01	Sec	49.7s			143s	0.38s	5.7 ms
		0.005	Few min	35.8s			181s	0.34s	5.7 ms
(A-V)	4	0.1	Frac sec.					0.10s	2.6 ms
		0.05	Few sec					0.039s	2.6 ms
		0.01	Few sec				161s	0.085s	2.9 ms
		0.005	Few min				128s	0.040s	2.8 ms
		0.1	Frac sec.	0.7s				0.40s	6.6 ms

Table 5 continued

Function Reference	p	δ	Continuous approximation		Non necessarily continuous approximation		
			[16]	[17]	[13]	LinA	
				Exact	Heuristic	Exact	Heuristic
(A-VI)	2	0.1	Min	24.4s	115s	0.087s	4.0 ms
		0.05	Few days	107.7s	88s	0.10s	4.2 ms
		0.01	t.o., 1800s/it	6819.1s	164s	0.047s	4.9 ms
(A-VII)	3	0.005	t.o., 1800s/it	35,787.4s	195s	0.052s	6.5 ms
		0.1	t.o., 1800s/it	311.5s	226s	0.10s	4.1 ms
		0.05	t.o., 1800s/it	14,514.6s	287s	0.15s	4.4 ms
(A-VIII)	3	0.01	t.o., 1800s/it	35,411s	268s	0.17s	10 ms
		0.005	t.o., 1800s/it	70,313.1s	869s	0.13s	5.7 ms
		0.1	Sec	1.7s	74s	0.16s	4.0 ms
(A-IX)	5	0.05	t.o., 1800s/it	5.3s	83s	0.096s	4.3 ms
		0.01	t.o., 1800s/it	59.6s	138s	0.066s	4.4 ms
		0.005	t.o., 1800s/it	247.2s	1466s	0.087s	6.4 ms
(A-X)	5	0.1	Few days	1.9s	77s	0.23s	13 ms
		0.05	t.o., 1800s/it	12s	64s	0.18s	12 ms
		0.01	t.o., 1800s/it	13,873.4s	114s	0.20s	11 ms
(A-X)	5	0.005	t.o., 1800s/it	42,068.4s	784s	0.19s	13 ms
		0.1	t.o., 1800s/it	3085.5s		0.59s	328 ms
		0.05	t.o., 1800s/it	10,735.3s		0.60s	334 ms
(A-X)	5	0.01	t.o., 1800s/it	82,831.3s		0.57s	317 ms
		0.005	t.o., 1800s/it	90,028.4s		0.52s	317 ms

We mention that the naive algorithm generally does not work for overestimation and underestimation, as interpolators are not constrained to stay over or under a function. Therefore, we only report the result of the naive algorithm for the approximation. Similarly, there is no standard way to generalize the sandwich algorithm to relative errors, so this baseline is not included in this comparison.

Results in Tables 6 and 7 show that we were able to find PWL functions with fewer segments than [13] and much fewer segments than the naive algorithm. As in the case of absolute errors, we can observe a drastic improvement in the computation times over the state-of-the-art.

Such major performance speed-up in the computation of the optimal piecewise linear approximation opens up new possibilities, such as the application of LINA to solve with minimal efforts some MINLP instances. For example, this offers a natural way to tackle instances of linearly constrained MINLPs with a nonlinear but separable objective function. This is the case of the congested multicommodity network design problem used as an illustrative case study in the next subsection.

6.3 Application on MINLPs: the case of the multicommodity network design problem with congestion

To illustrate how LINA can be used as an effective pre-processing to derive high-quality bounds with minimal effort for linearly constrained MINLPs, we consider the case of the Congested Multicommodity Network Design (cMCND) problem where there is a different non-linear congestion function for each node of a graph. The problem was introduced by Paraskevopoulos et al. [15] to explicitly take into account the congestion occurring at nodes of networks used to represent a wide range of planning and operation management problems in transportation, telecommunications, logistics, and production. The resulting MINLP is linearly constrained and its nonlinear objective function is separable as a sum of positive univariate nonlinear terms. The state-of-the-art solution method for this problem is a mixed-integer second-order cone reformulation solved with a dedicated solver.

The LINA+MILP method consists of replacing each nonlinear term of the MINLP with its PWL overestimation computed with LINA, then solving the resulting MILP with a black box MILP solver. This approach is well suited for the use of relative errors since all the approximated terms are summed together, and all the terms in the objective function are positive. This implies that a relative error of ε on each nonlinear function propagates to the whole function, guaranteeing that the solution found is within ε of the optimal one. This fact guided our choice of problem to tackle in this benchmark as it makes our results easier to compare to the state-of-the-art.

Many different approaches exist to model piecewise linear functions in MILP each requiring a different quantity of additional binary variables and constraints. Various representations have been studied, for example, by [8, 9, 25], including ones where the number of binary variables increases only logarithmically with the number of linear pieces. Vielma et al. [24] showed that most models for continuous PWL could be either directly applied to discontinuous PWL functions or adapted for it via duplication of breakpoints or a combination of models. In practice, most modern commercial solvers

Table 6 Results from [13] and LINA for under/overestimation with relative tolerance ϵ

Function Reference	ϵ	Underestimation				Overestimation							
		[13] n_+	Time	LINA exact n^*	Time	LINA heuristic n_+	Time	[13] n_+	Time	LINA exact n^*	Time	LINA heuristic n_+	Time
(R-I)	0.01	6	8s	5	104 ms	6	6.57 ms	6	8s	5	106 ms	6	6.64 ms
	0.001	19	19s	18	90.0 ms	19	9.67 ms	19	17s	18	99.9 ms	19	9.55 ms
	0.0001	56	50s	56	61.3 ms	57	19.0 ms	56	52s	56	64.5 ms	57	18.8 ms
(R-II)	0.01	10	11s	9	80.0 ms	10	5.17 ms	10	10s	9	67.3 ms	10	5.14 ms
	0.001	27	27s	26	66.6 ms	27	9.31 ms	27	25s	26	73.8 ms	27	9.37ms
	0.0001	82	74s	81	71.8 ms	82	22.4 ms	82	73s	81	80.5 ms	82	22.5ms
(R-III)	0.01	14	16s	14	125 ms	14	6.31 ms	14	14s	14	98.7 ms	14	6.24 ms
	0.001	43	42s	42	111 ms	43	13.6 ms	43	41s	42	118 ms	43	13.4 ms
	0.0001	134	114s	133	76.8 ms	133	36.3 ms	133	109s	133	76.8 ms	133	34.7 ms

Bold indicates optimal number of linear segments

Table 7 Results from the naive algorithm and LINA for the approximation with relative tolerance ε

Function Reference	ε	Approximation				
		Naive n_+	LinA-exact n^*	Time	LinA-heuristic n_+ Time	
(R-I)	0.01	7	4	6.70 ms	5	2.72 ms
	0.001	41	12	9.05 ms	13	5.63 ms
	0.0001	82	39	16.2 ms	43	14.2 ms
(R-II)	0.01	10	6	4.89 ms	7	3.10 ms
	0.001	40	19	8.10 ms	20	7.03 ms
	0.0001	92	58	17.6 ms	58	18.8 ms
(R-III)	0.01	60	9	5.88 ms	10	4.87 ms
	0.001	88	30	11.1 ms	31	11.3 ms
	0.0001	437	94	27.3 ms	95	31.7 ms

such as CPLEX and GUROBI offer the possibility to handle piecewise linear functions automatically. We make use of that feature and let the solver handle the PWL functions. Exact details on the resulting MILP reformulation are provided in the appendix.

In addition to comparing our algorithms to the state-of-the-art, we compared the impact of using an optimal PWL compared to non-optimal ones. To do so, we also solved the instances using linearizations obtained by the naive and sandwich algorithms. These also serve as a good way to study the potential trade-off between the number of segments and the continuity of the PWL approximation, as both the naive and the sandwich algorithms PWL approximations are always continuous for this problem.

We mention that the running times of the other piecewise linearization techniques used in the benchmarks of Sect. 6.2 made them completely impractical for this use case as cMCND instances contained several dozen non-linear functions and so the linearization time would overshadow the time taken by the solver.

The computational evaluation of all methods was done on the 258 instances from the cMCNDlib benchmark generated by [15], partitioned into 43 groups of 6 instances with, using their notation, different *incret* (either 6, 7, or 8) and *versions* (either 3 or 8). These instances include either 20, 25, 30, or 100 nodes, from 10 to 400 commodities, and from 100 to 700 arcs.

The authors of [15] kindly gave us access to their implementation, allowing us to conduct a fairer comparison between each approach. We, however, mention that we did not obtain the parameters files from every instance. Therefore, some of the entries of Table 8 are left blank. Overall, the algorithm of [15] could be tested on 181 instances, whereas the LinA+MILP method and the naive+MILP method could both be tested on all 258. As we compared our approach to the state of the art using relative errors for which the sandwich algorithm is not well suited, we split our comparison into the two types of error. The computing times from all methods and errors are displayed in Tables 8 and 9.

Table 8 Time taken (in seconds) on instances of cMCND problem with a relative error of 1%

Instance	Version 3, increment 6		Version 3, increment 7		Version 3, increment 8		Version 8, increment 6		Version 8, increment 7		Version 8, increment 8							
	Naive	LinA	Naive	LinA	Naive	LinA	Naive	LinA	Naive	LinA	Naive	LinA						
c33	1.03	1.3	0.81	0.98	1.2	0.95	1.0	1.28	5.41	6.53	1.64	1.89						
c35	2.05	2.44	2.16	1.73	2.36	1.64	2.64	3.14	11.8	9.05	2.0	4.36						
c36	14.6	3.2	3.66	24.0	2.92	5.41	12.2	3.98	5.53	12.4	4.48	2.92	17.2	15.1	12.1	14.2	4.31	8.66
c37	2940.0	3200.0	2460.0	2600.0	2790.0	2040.0	3200.0	2400.0	2590.0	t.o	3140.0	t.o	2370.0	t.o	3040.0	2450.0	2610.0	
c38	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	
c39	763.0	1090.0	1180.0	1400.0	1230.0	1350.0	1520.0	654.0	1450.0	2670.0	1180.0	1340.0	1510.0	1810.0	2060.0	1050.0	1480.0	1290.0
c40	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o
c41	1.95	0.66	0.97	2.3	0.89	0.98	3.95	1.48	1.28	3.03	1.11	1.51	3.69	2.05	2.7	3.48	1.94	3.33
c42	5.72	2.31	2.19	16.2	2.2	2.09	7.91	3.34	2.64	6.44	2.52	4.86	10.2	7.84	4.7	9.39	3.14	6.44
c43	3.92	1.36	1.26	8.53	1.64	2.28	4.91	1.81	1.14	2.84	2.28	2.84	2.28	13.0	6.99	4.08	8.0	
c44	4.47	2.56	2.09	5.34	2.11	3.05	9.45	3.73	2.19	10.9	3.74	6.89	7.72	7.67	6.69	6.25	2.63	5.66
c45	723.0	629.0	536.0	762.0	690.0	514.0	383.0	313.0	243.0	t.o	1790.0	2470.0	1760.0	t.o	t.o	775.0	523.0	624.0
c46	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o
c47	743.0	719.0	475.0	759.0	334.0	218.0	678.0	85.4	80.7	1880.0	2230.0	1380.0	1110.0	1490.0	729.0	405.0	179.0	134.0
c48	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o
c49	35.0	11.5	7.13	40.4	20.8	13.3	57.2	15.5	7.69	109.0	28.2	39.5	327.0	184.0	98.3	164.0	15.4	31.9
c50	500.0	659.0	616.0	486.0	280.0	272.0	509.0	377.0	557.0	2000.0	1010.0	679.0	645.0	585.0	781.0	710.0	705.0	472.0
c51	45.8	57.5	50.4	214.0	82.2	92.0	218.0	57.0	56.2	275.0	136.0	64.1	153.0	141.0	137.0	88.1	61.0	62.3
c52	1690.0	815.0	1010.0	2590.0	1490.0	1720.0	2020.0	1870.0	1370.0	t.o	2760.0	3520.0	2880.0	t.o	t.o	2640.0	t.o	t.o

Table 8 continued

Instance	Version 3, increment 6		Version 3, increment 7		Version 3, increment 8		Version 8, increment 6		Version 8, increment 7		Version 8, increment 8			
	[15]	Naive	LinA	Naive	LinA	[15]	Naive	LinA	[15]	Naive	LinA	[15]	Naive	LinA
c53	t.o	t.o	t.o	2580.0	1630.0	2550.0	1080.0	1050.0	t.o	t.o	t.o	t.o	t.o	t.o
c54	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o
c55	t.o	t.o	1440.0	3550.0	3120.0	1410.0	352.0	771.0	t.o	t.o	t.o	t.o	t.o	1770.0
c56	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o
c57	66.1	14.4	16.3	14.5	8.27	13.1	55.6	8.5	8.69	51.0	45.3	44.6	111.0	89.0
c58	12.3	13.8	11.2	12.2	12.0	9.34	12.2	13.2	8.66	59.1	17.5	14.4	36.1	17.5
c59	243.0	69.9	43.1	86.1	35.9	46.3	28.8	36.3	15.0	142.0	126.0	167.0	216.0	77.7
c60	90.8	63.5	19.9	68.1	40.9	20.8	41.2	24.1	13.6	230.0	111.0	110.0	56.5	134.0
c61	t.o	t.o	t.o	t.o	t.o	t.o	t.o	2740.0	t.o	t.o	t.o	t.o	t.o	t.o
c62	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o
c63	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o
c64	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o
c25_100_10_F_L_5	0.98	0.92	0.95	0.84	0.95	0.84	1.0	1.05	3.86	1.63	2.92	3.73	1.05	0.81
c25_100_10_F_T_5	1.44	1.5	0.91	0.7	0.91	0.7	1.19	1.08	1.75	1.55	1.67	1.58	1.47	1.05
c25_100_10_V_L_5	0.17	0.17	0.17	0.17	0.17	0.17	0.34	0.33	0.17	0.14	0.39	0.58	0.56	0.78
c25_100_30_F_L_5	11.2	8.89	5.17	6.34	5.17	6.34	5.23	4.84	42.2	30.9	8.41	11.9	6.36	4.25
c25_100_30_F_T_5	2.53	2.89	7.06	6.53	7.06	6.53	4.5	4.81	15.5	14.9	877.0	312.0	16.6	21.3
c25_100_30_V_T_5	0.89	1.27	3.05	1.7	3.05	1.7	0.95	0.53	2.52	3.09	9.88	8.52	2.2	2.0
c100_400_10_F_L_10	125.0	246.0	265.0	158.0	152.0	99.3	172.0	348.0	126.0	199.0	362.0	203.0	372.0	351.0
c100_400_10_F_T_10	2150.0	1020.0	1360.0	1410.0	1360.0	1410.0	1560.0	1050.0	1550.0	771.0	1450.0	804.0	758.0	278.0
c100_400_10_V_L_10	5.61	10.8	6.16	5.11	14.0	9.86	11.7	14.2	13.1	5.2	11.9	4.47	4.77	8.88
c100_400_30_F_L_10	366.0	445.0	302.0	234.0	302.0	234.0	1260.0	423.0	839.0	400.0	1230.0	521.0	1080.0	724.0
c100_400_30_V_T_10	193.0	148.0	71.7	37.1	71.7	37.1	54.3	39.6	220.0	385.0	3560.0	1180.0	62.8	55.4

Table 9 Time taken (in seconds) on instances from the cMCND problem with an absolute error of 10 per linearization

Instance	Version 3, increment 6		Version 3, increment 7		Version 3, increment 8		Version 8, increment 6		Version 8, increment 7		Version 8, increment 8	
	Sandwich	LinA	Sandwich	LinA	Sandwich	LinA	Sandwich	LinA	Sandwich	LinA	Sandwich	LinA
c33	1.36	0.71	2.1	1.33	1.8	0.88	6.96	3.51	22.42	11.63	7.65	3.58
c35	19.39	5.11	12.71	7.45	9.37	7.75	20.32	21.23	15.26	8.89	16.03	12.74
c36	29.13	20.83	42.32	25.3	21.37	31.19	30.62	23.76	51.61	37.98	57.27	29.74
c37	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0
c38	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0
c39	t.0	2068.26	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0
c40	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0
c41	1.45	0.85	3.47	1.86	2.38	3.09	3.84	4.81	10.05	9.44	25.18	18.61
c42	27.38	13.01	25.38	19.75	10.8	32.58	20.08	22.24	21.37	13.86	43.13	25.34
c43	8.24	7.79	36.44	12.41	4.95	4.15	27.09	15.18	14.11	18.21	7.74	27.52
c44	32.16	10.2	37.89	17.58	17.66	34.39	18.04	19.05	17.88	13.42	11.44	24.43
c45	2588.9	t.0	t.0	1890.72	333.33	503.41	t.0	t.0	t.0	t.0	1770.74	2569.04
c46	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0
c47	t.0	1895.88	2583.68	1539.18	471.53	363.96	t.0	t.0	t.0	1698.95	2001.35	966.98
c48	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0
c49	64.86	67.08	166.63	317.07	97.81	184.29	125.84	546.17	284.2	236.7	87.07	310.55
c50	2644.3	t.0	2653.53	3459.43	2559.74	t.0	t.0	t.0	t.0	t.0	1716.01	t.0
c51	99.97	85.28	1018.95	921.37	366.03	366.86	327.57	622.87	3111.17	1437.46	789.67	210.27
c52	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0
c53	t.0	t.0	t.0	t.0	1934.39	t.0	t.0	t.0	t.0	t.0	t.0	t.0
c54	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0	t.0

Table 9 continued

Instance	Version 3, increment 6		Version 3, increment 7		Version 3, increment 8		Version 8, increment 6		Version 8, increment 7		Version 8, increment 8	
	Sandwich	LinA	Sandwich	LinA	Sandwich	LinA	Sandwich	LinA	Sandwich	LinA	Sandwich	LinA
c55	t.o	t.o	t.o	t.o	925.94	2522.21	t.o	t.o	t.o	t.o	t.o	t.o
c56	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o
c57	150.45	38.87	91.76	34.58	15.0	10.36	287.68	138.46	498.4	370.4	40.48	19.95
c58	77.91	39.66	54.01	43.18	33.67	37.03	317.06	97.8	104.3	67.57	45.38	34.86
c59	239.91	404.1	59.84	98.86	132.67	852.77	225.09	347.76	748.88	321.61	180.88	257.34
c60	244.22	138.37	558.86	93.09	584.51	50.66	472.36	508.05	545.55	280.31	470.78	495.5
c61	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o
c62	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o
c63	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o
c64	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o	t.o
c25_100_10_F_L_5	6.22	3.59	5.11	2.48	1.58	1.32	9.63	7.93	11.56	8.12	4.03	5.29
c25_100_10_F_T_5	6.99	6.01	2.92	2.9	2.56	1.43	9.05	7.3	7.28	5.53	1.99	1.8
c25_100_10_V_L_5	0.18	0.16	0.18	0.08	1.85	0.33	0.2	0.14	0.26	0.21	1.29	0.79
c25_100_30_F_L_5	24.37	36.95	24.1	18.55	8.79	13.03	t.o	t.o	31.76	39.25	19.36	22.95
c25_100_30_F_T_5	8.26	12.76	107.79	22.95	13.48	16.84	381.55	104.46	t.o	t.o	t.o	t.o
c25_100_30_V_T_5	1.53	0.76	10.84	3.66	3.41	2.35	22.62	19.43	25.57	10.41	27.64	12.29
c100_400_10_F_L_10	110.44	115.18	302.97	359.91	349.87	209.67	217.98	173.83	361.44	158.72	399.77	299.56
c100_400_10_F_T_10	2375.77	1821.23	1193.71	1395.55	1546.76	967.21	2436.55	1052.03	1731.57	2119.05	2975.51	1211.23
c100_400_10_V_L_10	12.26	7.99	12.19	12.05	15.76	10.04	19.06	8.77	9.23	18.79	11.55	14.31
c100_400_30_F_L_10	2352.83	1371.32	972.88	895.84	2458.75	1022.69	t.o	1219.93	t.o	1548.08	t.o	1728.01
c100_400_30_V_T_10	t.o	t.o	424.33	217.01	322.34	164.99	t.o	t.o	t.o	t.o	3187.83	497.86

6.3.1 Relative error

These computations were performed using CPLEX 20.1 on the computer described in Table 3 with a 1% relative overestimation tolerance in the piecewise linearization phase, i.e. $\varepsilon = 0.01$ and with a time limit of an hour. As mentioned in the previous subsection this error propagates to the whole objective function. We mention that there is a trade-off between the linear approximation's precision and the solver's precision, as it might be faster to do a more precise linearization and stop the solver when at a larger gap. However, for the sake of simplicity, we used the default CPLEX tolerance of 0.01%. We note that the naive algorithm provides a valid PWL overestimation in the case of the cMCND because the non-linear congestion functions are concatenations of convex functions for which interpolators stay over the nonlinear function.

Comparison of LINA+MILP and [15]: For this analysis, we focus on the 181 instances on which [15] was tested. Of those, 115 instances were solved optimally by at least one of the two methods. Among these 115 instances, LINA+MILP was faster than [15] on 96, and [15] was faster than LINA+MILP on 19 instances. Finally, over the 106 instances solved optimally by both algorithms, the average time taken by LINA+MILP was 334 sec whereas [15] took 469 sec ($\approx 29\%$ improvement).

Comparison of LINA+MILP and naive+MILP: For this analysis we consider all 258 instances. Out of those, 184 were solved optimally by at least one of the two methods. Among these 184 instances, LINA+MILP was faster than naive+MILP on 112 instances, and naive+MILP was faster than LINA+MILP on 72 instances. Finally, over the 180 instances solved optimally by both algorithms, the average time taken by LINA+MILP was 331 sec whereas naive+MILP took 393 sec ($\approx 16\%$ improvement). We note that on average, each linearization made by LinA saved around 2 segments over the naive approach on the congestion functions, which pales in comparison with the hundreds of segments saved in the previous section. This is explained by the fact that the linearized congestion functions were quite regular, and because the function spanned several orders of magnitude, so the allowed errors were quite large for some parts of the domain. Therefore, equidistant breakpoints did not create too many additional segments. Yet, we saw a clear speed-up by using LINA over the widely used approach.

6.3.2 Absolute error

This section aims to compare the impact of using a non-naive but not optimal linearization compared to the optimal one offered by LinA. As mentioned in a previous section, the sandwich algorithm does not have a canonical generalization to relative error, therefore we compare LINA+MILP with sandwich+MILP with an absolute error of 10 per linearization. These computations were performed using CPLEX 12.8 on a cluster using an AMD EPYC 7453 CPU @2.75Ghz with 256 GB of RAM. Here again, for reproducibility purposes, we limit the number of threads to one for each instance. We note that since different setups were used and absolute errors do not propagate as well on sums, these results are not to be compared with the ones obtained in the previous section.

Comparison of LINA+MILP and sandwich+MILP: Across the 258 instances, 154 were solved optimally by at least one method. Among those, LINA+MILP was faster

on 106 instances, while sandwich+MILP was faster on 48. On the 142 instances solved by both methods, the average time for sandwich+MILP was 410 sec while it was 330 sec for LINA+MILP ($\approx 19.5\%$ improvement). Here again, LINA compares favorably to a widely used linearization method.

6.3.3 Summary

The full results (with CPLEX logs, solutions found, etc) can be found at <https://github.com/LICO-labs/cMCND-Computational-Results>. These results show that with minimal implementation efforts and without trying to take advantage of the structure of the problem, the straightforward LINA+MILP approach is competitive with the state-of-the-art method for solving the multicommodity network design problem with congestion.

During our benchmarks, we did not observe a trade-off between the number of segments and the continuity of the approximation. This might be partly explained by our algorithm only introducing a few discontinuities in the PWL approximation (at most, one per change in concavity) and that adding even very few segments slows down the solver significantly. Therefore, it seems that adding segments (and thus binary variables in the model) to go from almost continuous to continuous is not worth it, but this would require further investigation.

7 Conclusion and future work

In conclusion, the proposed algorithm solves the corridor fitting problem, outperforms what already exists to compute PWL approximations with predefined pointwise maximal error, and does not require any non-linear optimization solver in contrast to the recent contributions of [13, 16, 17]. Instances from the literature are solved in mere milliseconds.

We also demonstrated how an approach based on the linearization of MINLPs using this algorithm can obtain state-of-the-art results in a simple and almost *black box* manner. As a next step, an exhaustive study of the advantages of utilizing LinA across a broader range of MINLP classes should be conducted to characterize the benefits and limitations of our approach. Further work could also be done on this approach by exploring other types of errors when approximating a MINLP with a MILP. Another interesting perspective is to generalize the linearizations to higher dimensions by approximating functions with hyper-planes.

For the Julia package, a further improvement would be to utilize the specific structure of the polyhedrons in the coefficient space while performing Algorithm 3 to obtain a sensible speedup while calculating the intersection as proposed in [14].

Appendix A Solving the multicommodity network design problem with congestion using LinA+MILP

A.1 Methodology

The solution method consists of identifying, for a given value ε , a PWL overestimator $\bar{f}^\varepsilon(x)$ that verifies Eq. (5).

$$f(x) \leq \bar{f}^\varepsilon(x) \leq f(x) + \varepsilon |f(x)|, \forall x \in \mathbb{D} \tag{5}$$

Let (\bar{P}^ε) be the MILP resulting from the replacement of all the non-linear functions f by \bar{f}^ε . Let c^* be the optimal cost of the initial MINLP and let $c(\bar{P}^\varepsilon)$ denote the optimal costs of \bar{P}^ε .

Solving (\bar{P}^ε) yields an upper bound for the MINLP that verifies $c^* \leq c(\bar{P}^\varepsilon) \leq (1 + \varepsilon)c^*$. Thus, we can then extract a lower bound as $\frac{c(\bar{P}^\varepsilon)}{(1+\varepsilon)} \leq c^*$. Moreover, the upper bound can be improved by recomputing the cost of the solution of \bar{P}^ε using the original cost function f .

A.2 Problem definition

Let A be the set of arcs, P be the set of commodities, and N be the set of nodes. Each node $i \in N$ has a fixed cost E_i , an initial capacity C_i^0 , an upgraded capacity C_i^δ , a delay cost D_i , a free flow classification delay F_i and a demand of commodity p denoted D_i^p . Each arc $(i, j) \in A$ has a fixed opening cost O_{ij} and a capacity U_{ij} . Each commodity $p \in P$ has a quantity to be shipped W_p and a unit routing cost over arc (i, j) denoted D_{ij}^p . The mathematical formulation of the CMND requires the following decision variables:

- continuous variables $x_{ij}^p \geq 0$ that specify the amount of flow of commodity $p \in P$ on arc $(i, j) \in A$
- binary variable y_{ij} equal to 1 if arc $(i, j) \in A$ is used and 0 otherwise
- binary variable z_i equal to 1 if node $i \in N$ is upgraded and 0 otherwise

The resulting mathematical formulation proposed by [15] is:

$$\min \sum_{(i,j) \in A} O_{ij}y_{ij} + \sum_{(i,j) \in A} \sum_{p \in P} D_{ij}^p x_{ij}^p + \sum_{i \in N} E_i z_i + \sum_{i \in N} g_i(x, z) \tag{6}$$

$$s.t. \sum_{j \in N_i^+} x_{ij}^p - \sum_{j \in N_i^-} x_{ji}^p = D_i^p, \quad i \in N, p \in P \tag{7}$$

$$x_{ij}^p \leq W^p y_{ij}, \quad (i, j) \in A, p \in P \tag{8}$$

$$\sum_{p \in P} x_{ij}^p \leq U_{ij} y_{ij}, \quad (i, j) \in A \tag{9}$$

$$\sum_{j \in N^-} \sum_{p \in P} x_{ji}^p \leq C_i^0 + C_i^\delta z_i, \quad i \in N \tag{10}$$

$$x_{ij}^p \geq 0, \quad (i, j) \in A, p \in P \tag{11}$$

$$y_{ij} \in \{0, 1\}, \quad (i, j) \in A \tag{12}$$

$$z_i \in \{0, 1\}, \quad i \in N \tag{13}$$

The objective function (6) minimizes the total cost of design, routing and capacity augmentation, and cost congestion. The function $g_i(x, z)$ that appears in the last term of the objective function models congestion and is expressed with Eq. (14) where α and β are calibration parameters. Flow conservation constraints (7) ensure that the demands are satisfied for each node. Constraints (8) make sure that no flow of any commodity exists on an arc that is not selected. Constraints (9) limit the amount of flow on an arc to the capacity of the arc. Finally, constraints (10) limit the total inflow of node i by its initial or extended capacity. The remaining constraints give domain definitions of variables.

$$g_i(x, z) = D_i \left(F_i \sum_{j \in N^-} \sum_{p \in P} x_{ji}^p + \beta \frac{\left(\sum_{j \in N^-} \sum_{p \in P} x_{ji}^p \right)^{\alpha+1}}{\left(C_i^0 + C_i^\delta z_i \right)^\alpha} \right) \tag{14}$$

This formulation is not suitable for a direct application of the LINA+MILP methodology because it contains bivariate nonlinear terms $g_i(x, z)$. Hereafter, we propose a reformulation containing only univariate nonlinear terms. The idea is simply to compute explicitly the quantity of flow where upgrading becomes optimal and to define a univariate function switching from the upgraded and non-upgraded cost at that point.

Let us introduce a new variable v_i defined as follows: $v_i = \sum_{j \in N^-} \sum_{p \in P} x_{ji}^p, \forall i \in N$. Under the new variable definition, the congestion functions can be expressed with Eq. (15). Let us define $|N|$ functions $h_i(v_i, z_i)$ as $h_i(v_i, z_i) = g_i(v_i, z_i) + E_i z_i$ with $i \in N$. These multivariate functions can be reduced to univariate functions as defined in Lemma 5.

$$g_i(v_i, z_i) = D_i \left(F_i v_i + \beta \frac{(v_i)^{\alpha+1}}{\left((C_i^0 + C_i^\delta z_i)^\alpha \right)} \right), \quad i \in N \tag{15}$$

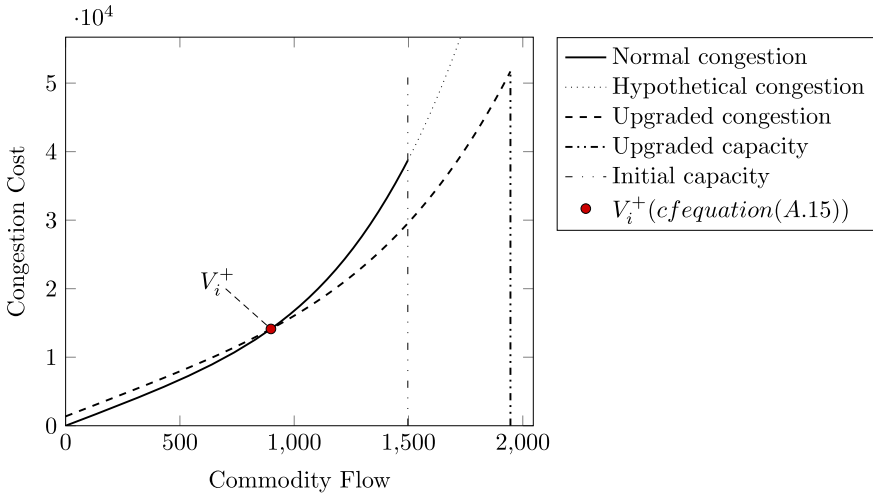


Fig. 11 Example of congestion function $f_i(v_i)$

Lemma 5 For any $i \in N$, there exists a scalar $V_i^+ \in]0, C_i^0 + C_i^+]$ such that $h_i(v_i, z_i)$ can be reduced to a univariate function $f_i(v_i)$ that verifies Eq. (16).

$$f_i(v_i) = \begin{cases} f_i^<(v_i) = D_i \left(F_i v_i + \beta \frac{(v_i)^{\alpha+1}}{(C_i^0)^\alpha} \right), & \text{if } v_i \leq V_i^+ \\ f_i^>(v_i) = E_i + D_i \left(F_i v_i + \beta \frac{(v_i)^{\alpha+1}}{(C_i^0 + C_i^+)^\alpha} \right), & \text{otherwise} \end{cases} \quad (16)$$

Proof From (16), we derive Eq. (17).

$$(f_i^<(v_i) - f_i^>(v_i))' = (\alpha + 1)v_i^\alpha D_i \beta \left(\frac{1}{(C_i^0)^\alpha} - \frac{1}{(C_i^0 + C_i^+)^\alpha} \right) \quad (17)$$

Since $\alpha > 0, \beta > 0, D_i > 0, C_i^0 > 0, C_i^+ > 0$ and $v_i \geq 0$, then $(f_i^<(v_i) - f_i^>(v_i))' \geq 0$, meaning that the difference $f_i^<(v_i) - f_i^>(v_i)$ is non decreasing as illustrated on figure 11. Let V_i^+ be the value that verifies $f_i^<(V_i^+) = f_i^>(V_i^+)$ and can be computed with Eq. (18).

$$V_i^+ = \exp \left(\frac{\ln E_i - \ln D_i - \ln \beta - \ln \left(\frac{1}{C_i^0} - \frac{1}{C_i^0 + C_i^+} \right)}{\alpha + 1} \right) \quad (18)$$

Therefore $\forall v_i \leq V_i^+$, so it is cost-effective not to upgrade the capacity at node i , which leads to function $h_i(v_i, z_i) = f_i^<(v_i)$. On the other hand, $\forall v_i \geq V_i^+$, so it is cost-effective to upgrade the node capacity of i , which leads to the function $h_i(v_i, z_i) = f_i^>(v_i)$. Taking into account both cases leads to function $h_i(v_i, z_i) = f_i(v_i)$ from Eq. (16). □

The resulting linearly constrained non-convex MINLP formulation of the cMND with a sum of univariate nonlinear terms in the objective function is the following:

$$\min \sum_{(i,j) \in A} O_{ij}y_{ij} + \sum_{(i,j) \in A} \sum_{p \in P} D_{ij}^p x_{ij}^p + \sum_{i \in N} f_i(v_i) \tag{19}$$

s.t. (7) – (9) (20)

$$\sum_{j \in N^-} \sum_{p \in P} x_{ji}^p = v_i, \quad \forall i \in N \tag{21}$$

(11) – (12) (22)

$$v_i \in [0, C_i^0 + C_i^\delta], \quad \forall i \in N. \tag{23}$$

Solving this formulation with the LINA+MILP methodology from section A.1 requires replacing each nonlinear function f_i with the piecewise linear functions \bar{f}_i^ϵ verifying Eq. (5). In practice, most modern commercial solvers such as CPLEX and GUROBI offer the possibility to handle piecewise linear functions automatically. We make use of that feature and let the solver handle the PWL functions.

Remark 6 Each nonlinear function $f_i(v_i)$ is not differentiable at $v_i = V_i^+$ so Algorithm 5 cannot apply directly. However, the functions are C^1 on each of the subdomains $\mathbb{D}_1 = [0, V_i^+]$ and $\mathbb{D}_2 = [V_i^+, C_i^0 + C_i^\delta]$. We can, therefore, linearize on both subdomains and concatenate the resulting piecewise linear function. We could also use Algorithm 3 to find the segment that crosses V_i^+ .

Acknowledgements We would like to thank Jean Laprés-Chartrand for his fruitful discussions and help during the algorithms’ implementation. The authors would like to thank the anonymous reviewers for the detailed and constructive feedback that greatly improved this work.

Funding This research benefited from the support of the FMJH Program PGMO, from the support of EDF-Thales-Orange. This research also benefited from the support of the program “Soutien à la Mobilité Internationale (SMI) de Toulouse INP”. We acknowledge the support of the Natural Sciences and Engineering Council of Canada (NSERC) through the Undergraduate Student Research Awards.

Data availability statement We made the full set of instances originally from [15] available on GitHub at the following address: <https://github.com/LICO-labs/cMCND-Computational-Results>

Code Availability The full code was reviewed and is available on GitHub at the following address: <https://github.com/LICO-labs/LinA.jl> under the MIT License.

Declarations

Conflict of interest The authors declare no conflict of interest.

References

- Ahmadi, H., Martí, J.R., Moshref, A.: Piecewise linear approximation of generators cost functions using max-affine functions. In: 2013 IEEE Power Energy Society General Meeting, pp. 1–5 (2013). <https://doi.org/10.1109/PESMG.2013.6672353>
- Camponogara, E., Nazari, L.F.: Models and algorithms for optimal piecewise-linear function approximation. *Math. Probl. Eng.* (2015). <https://doi.org/10.1155/2015/876862>
- Codsi, J., ngueveu: Lico-labs/lina.jl: Release for doi (2024). <https://doi.org/10.5281/zenodo.14052032>
- Douglas, D.H., Peucker, T.K.: Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Class. Cartogr. Reflect. Infl. Artic. Cartogr.* **10**, 112–122 (1973)
- Ertel, J.E., Fowlkes, E.B.: Some algorithms for linear spline and piecewise multiple linear regression. *J. Am. Stat. Assoc.* **71**(355), 640–648 (1976)
- Geißler, B., Martin, A., Morsi, A., Schewe, L.: Using piecewise linear functions for solving minlps. In: Lee, J., Leyffer, S. (eds.) *Mixed Integer Nonlinear Programming*, pp. 287–314. Springer, New York (2012)
- Gritzali, F., Papakonstantinou, G.: A fast piecewise linear approximation algorithm. *Signal Process.* **5**(3), 221–227 (1983)
- Huchette, J., Vielma, J.P.: Nonconvex piecewise linear functions: Advanced formulations and simple modeling tools. *Oper. Res.* **71**(5), 1835–1856 (2019)
- Hwang, F., Huang, Y.: An effective logarithmic formulation for piecewise linearization requiring no inequality constraint. *Comput. Optim. Appl.* **79**, 601–631 (2021)
- Kong, L., Maravelias, C.: On the derivation of continuous piecewise linear approximating functions. *INFORMS J. Comput.* **32**(3), 1–16 (2020)
- Legat, B., Deits, R., Evans, O., Goretkin, G., Koolen, T., Huchette, J., Oyama, D., Forets, M., guberger, Schwarz, R., Saba, E., Coleman, C.: Juliapolyhedra/polyhedra.jl: v0.5.1 (2019). <https://doi.org/10.5281/zenodo.1214290>
- Luwel, K.A., Beem, L., Onghena, P., Onghena, L.: Susing segmented linear regression models with unknown change points to analyze strategy shifts in cognitive tasks. *Behav. Res. Methods Instrum. Comput.* **33**, 470–478 (2001)
- Ngueveu, S.U.: Piecewise linear bounding of univariate nonlinear functions and resulting mixed integer linear programming-based solution methods. *Eur. J. Oper. Res.* **275**, 1058–1071 (2019)
- O'Rourke, J.: An on-line algorithm for fitting straight lines between data ranges. *Commun. ACM* **24**(9), 574–578 (1981). <https://doi.org/10.1145/358746.358758>
- Paraskevopoulos, D.C., Gürel, S., Bektaş, T.: The congested multicommodity network design problem. *Transp. Res. Part E Logist. Transp. Rev.* **85**, 166–187 (2016)
- Rebennack, S., Kallrath, J.: Continuous piecewise linear delta-approximations for univariate functions: computing minimal breakpoint systems. *J. Optim. Theory Appl.* **167**(2), 617–643 (2015)
- Rebennack, S., Krasko, V.: Piecewise linear function fitting via mixed-integer linear programming. *INFORMS J. Comput.* (2019) **to appear**
- Revels, J., Lubin, M., Papamarkou, T.: Forward-mode automatic differentiation in Julia. [arXiv:1607.07892](https://arxiv.org/abs/1607.07892) [cs.MS] (2016)
- Rosen, J.B., Pardalos, P.M.: Global minimization of large-scale constrained concave quadratic problems by separable programming. *Math. Program.* **34**(2), 163–174 (1986)
- Rote, G.: The convergence rate of the sandwich algorithm for approximating convex functions. *Computing* **48**(3), 337–361 (1992). <https://doi.org/10.1007/BF02238642>
- Tomek, I.: Piecewise-linear approximation with a bound on absolute error. *Comput. Biomed. Res.* **7**(1), 64–70 (1974)

22. Tomek, I.: Two algorithms for piecewise-linear continuous approximation of functions of one variable. *IEEE Trans. Comput.* **23**(4), 445–448 (1974)
23. 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>
24. Vielma, J.P., Ahmed, S., Nemhauser, G.L.: Mixed-integer models for nonseparable piecewise-linear optimization: unifying framework and extensions. *Oper. Res.* **58**, 303–315 (2010)
25. Vielma, J.P., Nemhauser, G.L.: Modeling disjunctive constraints with a logarithmic number of binary variables and constraints. *Math. Program.* **128**(1), 49–72 (2011)
26. Watson, G.A.: Choice of norms for data fitting and function approximation. *Acta Numer.* **7**, 337–377 (1998). <https://doi.org/10.1017/S0962492900002853>
27. Yang, L., Liu, S., Tsoka, S., Papageorgiou, L.G.: Mathematical programming for piecewise linear regression analysis. *Expert Syst. Appl.* **44**, 156–167 (2016). <https://doi.org/10.1016/j.eswa.2015.08.034>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.