IISE Transactions

ISSN: (Print) (Online) Journal homepage: www.tandfonline.com/journals/uiie21

Efficient continuous piecewise linear regression for linearising univariate non-linear functions

John Alasdair Warwicker & Steffen Rebennack

IISE TRANSACTIONS
https://doi.org/10.1080/24725854.2023.2299809

Taylor & Francis
Taylor & Francis Group

OPEN ACCESS Check for updates

Efficient continuous piecewise linear regression for linearising univariate non-linear functions

John Alasdair Warwicker and Steffen Rebennack

Stochastic Optimization, Institute of Operations Research, Karlsruhe Institute of Technology, Karlsruhe, Germany

ABSTRACT
Due to their flexibility and ability to incorporate non-linear relationships, Mixed-Integer Non-Linear Programming (MINLP) approaches for optimization are commonly presented as a solution tool for real-world problems. Within this context, piecewise linear (PWL) approximations of non-linear continuous functions are useful, as opposed to non-linear machine learning-based approaches, since they enable the application of Mixed-Integer Linear Programming techniques in the MINLP framework, as well as retaining important features of the approximated non-linear functions, such as convexity. In this work, we extend upon fast algorithmic approaches for modeling discrete data using PWL regression by tuning them to allow the modeling of continuous functions. We show that if the input function is convex, then the convexity of the resulting PWL function is guaranteed. An analysis of the runtime of the presented algorithm shows which function characteristics affect the efficiency of the model, and which classes of functions can be modeled very quickly. Experimental results show that the presented approach is significantly faster than five existing approaches for modeling non-linear functions from the literature, at least 11 times faster on the tested functions, and up to a maximum speedup of more than 328,000. The presented approach also solves six benchmark problems for the first time.

ARTICLE HISTORY
Received 14 June 2023
Accepted 19 December 2023

KEYWORDS
Piecewise linear function; continuous function; function fitting; piecewise linear regression; convex function

1. Introduction

Many real-world optimization problems are today formulated and solved as large-scale Mixed-Integer Non-Linear Programming (MINLP) problems, including problems from the fields of scheduling (Yue and You, 2013) and energy (Geißler *et al.*, 2012). Despite a number of recent advancements in solution techniques and software (see e.g., Lee and Leyffer (2011)), MINLP problems still require vast computational resources to solve. Hence, fast, approximate solutions are often sought.

One such approach is to utilize advancements in the computation and solution of Mixed-Integer Linear Programming (MILP) problems. This is achieved by approximating the non-linearities present in the MINLP through Piecewise Linear (PWL) functions. This allows complex non-linear and non-convex programming problems to be (approximately) solved quickly using standard MILP techniques (Feijoo and Meyer, 1988; Geißler *et al.*, 2012; Rebennack and Kallrath, 2015a,2015b; Rebennack, 2016a). Geißler *et al.* (2012) showed it is possible to solve MINLP problems using only MILP techniques when the non-linearities are approximated by PWL regression functions as opposed to polynomial regression functions. As an example, Gunnerud and Foss (2010) modeled the non-linearities within a real-time optimization model of process systems from the field of engineering, allowing MILP techniques to be used. They also found error bounds on the optimal solution to the original MINLP model.

PWL functions are known in the field of statistics as splines of order 2, with a continuity requirement at the knots (breakpoints). As opposed to standard parametric polynomial regression models, fitting data with a PWL function can be advantageous due to the removal of the complicating non-linearity present in the data, and providing analytical insights into the underlying pattern, while maintaining accurate models. Such approaches have been used in the fields of healthcare (Wagner *et al.*, 2002), energy and power systems (Guan *et al.*, 2018), network flow studies (Muriel and Munshi, 2004) and engineering (Gunnerud and Foss, 2010), amongst others. Furthermore, PWL functions are commonly used in the field of data envelopment analysis to fit frontiers to data, since their adaptability in incorporating convexity constraints allows the retention of important features of the data set (Charnes *et al.*, 1978). Such approaches extend to applications in energy fields (Hwangbo *et al.*, 2018; Ding, 2019).

CONTACT John Alasdair Warwicker john.warwicker@kit.edu
Supplemental data for this article can be accessed online at https://doi.org/10.1080/24725854.2023.2299809.

Alongside using PWL functions, a recent trend for solving the univariate curve fitting problem is to utilize non-parametric and semi-parametric approaches from statistical learning (Hastie *et al.*, 2009). Non-parametric methods, such as kernel smoothing, approximate the data using a weighted moving average approach, where the kernel (i.e., the window function) is used to define the weights. A further example is smoothing splines, which provide a trade-off in the quality of the fit as well as its smoothness. These splines typically use cubic polynomials, where the smoothness requirement is explicit at the intersection of adjacent segments (known as knots, or breakpoints) (Wang, 2011).

One disadvantage of these approaches is an assumption of a sufficiently large data set, since typically neighboring data points are used in the calculation of the regression function value. For smaller data sets, there is a higher likelihood of overfitting, and hence, parametric approaches are typically favored. Furthermore, whereas non-parameteric approaches for fitting functions to data typically lead to more accurate approximations, the linearity present in PWL functions is better suited for approximating pre-existing functions in the context of mathematical programming models, due to their effectiveness at retaining shape-based features of the approximated function. In this context, non-parametric models do not produce solutions satisfying the MINLP constraints. Hence, in the remainder of this article, we consider the problem of finding the best (shape-constrained) PWL regression function for the context of solving MINLP problems.

Although the linearity significantly simplifies the problem, there are still difficulties involved in finding optimal PWL functions, such as ensuring the continuity of the connected linear segments, and calculating optimal breakpoint locations (i.e., where the linear segments intersect). Recently, the PWL regression problem has been approached from an optimization perspective (Toriello and Vielma, 2012; Goldberg *et al.*, 2014; Goldberg *et al.*, 2021). Kallrath and Rebennack (2014) presented non-convex Non-Linear Programming (NLP) approaches for computing tight over- and underestimators for fitting PWL functions to continuous functions. Rebennack and Kallrath (2015b) further presented three non-convex NLP models, alongside two heuristics, to fit PWL functions within a given error tolerance. Exact MILP approaches have been presented to optimally fit PWL functions to discrete data, which implicitly model the breakpoint locations (where adjacent linear segments of the PWL function intersect) (Kong and Maravelias, 2020; Rebennack and Krasko, 2020); an experimental analysis suggested the formulation from Rebennack and Krasko (2020) is faster. Similar approaches have been utilized for machine learning problems, including clustering and clusterwise linear regression (Warwicker and Rebennack, 2023a). These models become Mixed-Integer Quadratically-Constrained Programs (MIQCPs) if the aim is to minimize the sum of the squares of the residuals (i.e., the L_2 metric). The approach by Rebennack and Krasko (2020) was also extended to model continuous functions with continuous PWL functions, which uses an adaptively refined discretization of the function into a series of data points, using the refinement strategy presented by Rebennack and Kallrath (2015b).

Traditionally, however, such problems were solved exactly using incremental algorithms from Computer Science, taking advantage of the inherent nature of the problems to solve them quickly. The advantage of such methods is that due to the proven polynomial runtimes, even large problems can be solved very quickly. In particular, Imai and Iri (1986) presented a fast algorithm for PWL regression, where the goal is to fit a PWL function with minimal breakpoints such that the maximal error (i.e., using the L_∞ metric) is within a given tolerance (known as the min-B problem) - a small extension by Hakimi and Schmeichel (1991) ensured the linear runtime of the algorithm. This was the basis of the $\mathcal{O}(n^2)$ implementation for fitting a PWL function with a given number of breakpoints with minimal error (known as the min-ξ problem) presented by Wang *et al.* (1993) (an $\mathcal{O}(n \log n)$ algorithm for this problem was later presented by Goodrich (1995)). These methods, however, cannot be simply extended to deal with outliers, or for other distance metrics. We discuss these methods in more detail in Section A of the appendix.

Recently, similar algorithmic approaches have been presented for the min-B problem, which use optimal greedy strategies that seek to iteratively maximize the length of each linear segment (Ngueveu, 2019; Codsi *et al.*, 2021). However, the resulting PWL regression functions are not guaranteed to be continuous (i.e., they are Non-Necessarily Continuous (NNC)) if the function being modeled is non-convex. Since removing the continuity requirement results in a significantly easier problem (Chen and Wang 2013), we focus our discussion on continuous PWL regression.

We summarize the most important works for univariate PWL fitting in Table 1.

In this article, we present exact algorithms for optimally fitting PWL regression functions to continuous data. Our main contributions are as follows:

1. We embed the linear time algorithm presented by Imai and Iri (1986) within the framework for fitting continuous functions presented by Rebennack and Kallrath (2015b), and prove it finds globally optimal continuous PWL regression functions in finite time.
2. We prove that if the input function is convex (concave), then the resulting PWL regression function will also be convex (concave).
3. Regarding its performance, we show that our presented algorithm is very fast by identifying function characteristics that affect the runtime of the presented algorithm the most.
4. We present experimental results that show the presented model is much faster than state-of-the-art models for approximating continuous functions with continuous PWL regression functions.
5. We show that the presented approach solves six benchmark problems for the first time.

Table 1. State-of-the-art approaches for univariate PWL function fitting. ⋆ represents the fastest shown available approaches for fitting data points with the L_∞ metric (Imai and Iri, 1986), with outliers (Warwicker and Rebennack, 2023b), and with higher-order metrics (Rebennack and Krasko, 2020); fitting continuous functions with non-necessarily continuous PWL functions (Codsi *et al.*, 2021), and continuous PWL functions (this paper).

Approach	*Model Type*	*Optimality*	*Input*
Imai and Iri (1986)*	Algorithmic	Global (L_∞)	Data points
Hakimi and Schmeichel (1991)	Algorithmic	Global (L_∞)	Data points
Toriello and Vielma (2012)	(Convex) MILP / MIQCP	Global Convex	Data points
Goldberg *et al.* (2014), Goldberg *et al.* (2021)	(Nonconvex) MINLP	Global	Data points
Kallrath and Rebennack (2014)	(Nonconvex) NLP	Global (L_∞)	Continuous function
Rebennack and Kallrath (2015b)	(Nonconvex) NLP	Global (L_∞)	Continuous function
Ngueveu (2019)	Algorithmic + MILP	Global NNC (L_∞)	Continuous function
Rebennack and Krasko (2020)*	(Convex) MILP / MIQCP	Global	Data points /
	Algorithmic (MILP)	Global	Continuous function
Kong and Maravelias (2020)	(Convex) MILP / MIQCP	Global	Data points /
	Algorithmic (MILP)	Global	Continuous function
Codsi *et al.* (2021)*	Algorithmic	Global NNC (L_∞)	Continuous function
Warwicker and Rebennack (2023b)*	(Decomposed) MILP	Global (L_∞) (Outliers)	Data points
This paper*	Algorithmic	Global (L_∞)	Continuous function

The rest of this article is structured as follows. In Section 2, we discuss the implementation of algorithms for function fitting from the literature. In Section 3, we discuss how these algorithms can be used to model continuous functions, including convex ones. In Section 4, we discuss how runtime bounds on the algorithm to model continuous functions can be found under mild assumptions. In Section 5, we present a detailed experimental analysis of our model and compare its performance with the state-of-the-art. We conclude with a discussion on the presented algorithms and present some ideas for future work.

2. Efficient algorithms for PWL regression for discrete univariate data

Throughout this article, the we refer to the set $\{1,\dots,n\}$ as $[n]$.

Suppose we are given a set of n ordered data points of the form $(x_i, y_i) \in \mathbb{R}^2, \forall i \in [n]$. The goal of PWL regression is to find a function $f : [x_1, x_n] \to \mathbb{R}$, whose graph is a connected polygonal line that best approximates the data points, according to a given distance metric. This polygonal line is made up of a number of connected *linear segments* which intersect at *breakpoints*. Note that compared with previous works which consider non-necessarily continuous functions (Ngueveu, 2019; Codsi *et al.*, 2021), this article considers a PWL function as a *continuous* PWL function.

The simplest metric to consider is the *maximum absolute difference* metric (or L_∞ metric), whereby the quality of the fit of the PWL function is measured by the maximum of the absolute (vertical) distances to each data point (i.e., the residuals). Regarding this problem, polynomial time algorithms have been presented for the following variants:

1. **Min-B Problem** - For a given error tolerance $\xi > 0$, find an approximating PWL function within the given accuracy with minimal number of breakpoints.
2. **Min-ξ Problem** - For a given number of breakpoints $B > 0$, find an approximating PWL function with the lowest error bound.

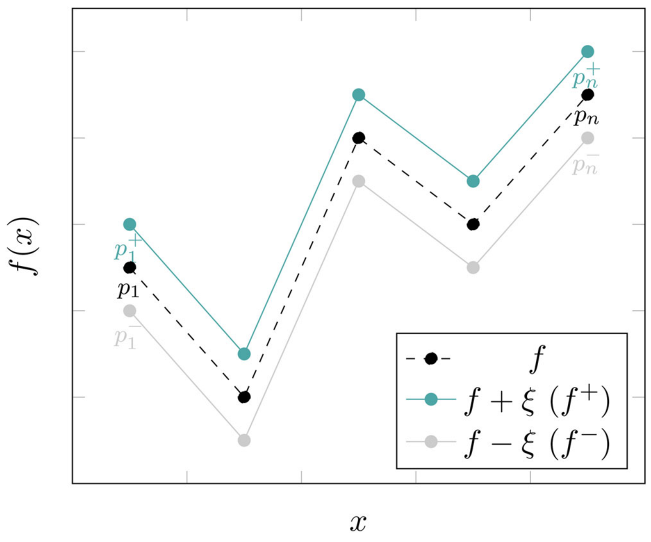


Figure 1. The *tunnel* formed by translating the polygonal function *f* above and below by ξ.

Since we mostly consider algorithms for the min-B problem, we discuss that in detail in the following subsection. We discuss algorithms for the min-ξ problem in Section A of the appendix.

2.1. Min-B problem - A $\mathcal{O}(n)$ time algorithm

The first linear time algorithm for the min-B problem was presented by Imai and Iri (1986), taking inspiration from the algorithm presented by Aggarwal *et al.* (1989) that finds a minimal vertex polygon that is nested between two convex polygons. For a given set of points $p_i = (x_i, y_i) \in \mathbb{R}^2$ ($\forall i \in [n]$) and a given error tolerance $\xi > 0$, the resulting PWL function must be contained within a *tunnel*, which is formed by translating the data points above and below by ξ (that is, consisting of the points $p_i^+ = (x_i, y_i + \xi)$ and $p_i^- = (x_i, y_i - \xi)$ to create the upper and lower sections $f^+ := \{p_i^+ | i \in [n]\}$ and $f^- := \{p_i^- | i \in [n]\}$ respectively); see Figure 1. The tunnel forms a (closed) polygon.

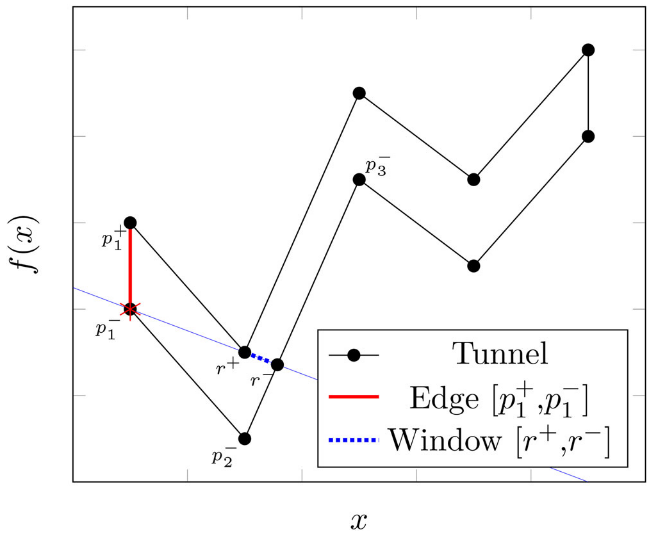


Figure 2. The window visible from the initial edge of the polygon. The starred point (at p_1^-) is the first calculated point of the approximating PWL function.

Hakimi and Schmeichel (1991) further noted that if the resulting PWL does not extend beyond the points on the upper and lower section of the given tunnel (formed by the translation), an optimal PWL function is still guaranteed, even if it exceeds the given error bound between data points. In either case, the linear time algorithm works the same.

Let an *edge* of the tunnel be defined as a line connecting some point on the upper section of the tunnel to some point on the lower section of the tunnel (see e.g., the line $[p_1^+, p_1^-]$ in Figure 2). For a given edge, the *visibility polygon* is the set of points in the tunnel that can be reached in a straight line from any point on the edge without exiting the tunnel (such points are *visible* from the edge), and the *invisibility polygon* is the remaining set of points in the tunnel (i.e., those that cannot be reached in a straight line from the edge without exiting the tunnel). The border between the two sets (i.e., where the two polygons touch) is defined as the *window* from the given edge to the end line $[p_n^+, p_n^-]$ (see e.g., the line $[r^+, r^-]$ in Figure 2). That is, the set of points between the edge and its window (inclusive) is exactly the visibility polygon. Each breakpoint on the approximated PWL function is found on the intersection of a given edge with the line connecting it to its window (see e.g., the starred point in Figure 2).

Algorithm 1 describes the process of the linear time algorithm. Starting from the initial window (the line $[p_1^+, p_1^-]$), if the exit $[p_n^+, p_n^-]$ of the polygon is not visible (see line 6), the algorithm computes the next window (line 7). Then, the remaining polygon (used as a basis for the next steps) is updated and a point on the approximated PWL function is calculated (lines 8-10). If the exit is visible, the final two points on the PWL function are computed (lines 11-12). For a more detailed pseudocode, which includes the calculations of the windows and intersecting points, see Imai and Iri (1986) and Hakimi and Schmeichel (1991). Notably, the breakpoints are only calculated when the convex hulls formed from the set of data points in the upper and lower sections, calculated from the previous breakpoint (or starting data point) up until the current point, intersect.

We note that interpolating between any two pairs of data points to find extreme values for the gradients will provide bounds on the gradient of the segments of the PWL functions found by Algorithm 1, as well as extreme values for the intercepts (see e.g., Rebennack and Krasko (2020)). That is, breakpoints are only calculated at the intersection of the two convex hulls, which avoids arbitrarily large gradient values. These extreme bounds are hereafter referred to as $[\underline{C}, \bar{C}]$ for the gradient and $[\underline{D}, \bar{D}]$ for the intercept.

Algorithm 1: Idea behind the $\mathcal{O}(n)$ algorithm for the min-B problem (Imai and Iri, 1986)

```
Input: Function $f$ defined by $p_1 = (x_1, y_1), ..., p_n = (x_n, y_n)$, error bound $\xi > 0$
Output: Approximated function $q_1, ..., q_m$
1  for $j = 1$ to $n$ do
2  $p_j^+ \leftarrow (x_j, y_j + \xi)$;  $p_j^- \leftarrow (x_j, y_j - \xi)$;
3  $P_1 \leftarrow$ polygon $p_1^- ... p_n^- p_n^+ ... p_1^+$;
4  $e_1 \leftarrow [p_1^+, p_1^-]$;
5  $i \leftarrow 1$;
6  while $[p_n^+, p_n^-]$ is not visible from $e_i$ in polygon $P_i$ do
7      $e_{i+1} \leftarrow$ window from $e_i$ to $[p_n^+, p_n^-]$ in polygon $P_i$;
8      $P_{i+1} \leftarrow$ polygon invisible from $e_i$ containing $[p_n^+, p_n^-]$ in
           polygon $P_i$;
9      $q_i \leftarrow$ point of intersection of line containing $e_{i+1}$ and
           the edge $e_i$;
10     $i++$;
11 $m \leftarrow i + 1$;
12 find a point $q_{m-1}$ on edge $e_i$ and a point $q_m$ on edge
   $[p_n^+, p_n^-]$ which are visible from each other in polygon $P_{m-1}$;
13 Return $q_1, ..., q_m$;
```

This algorithm could be extended to fit NNC PWL functions based on the ideas presented by Ngueveu (2019) and Codsi *et al.* (2021). The idea would be to take the points of the approximating PWL function as the furthest-right point of each computed window. Then, a new window would be formed by the vertical extension from the current point, and the process would reset (i.e., in each iteration, the segment of maximum length is found). For our applications in the remainder of this article, we consider Algorithm 1 as we are interested in fitting continuous PWL regression functions.

In Figure 3, we present an application of Algorithm 1 to the *DebrisFlow* data set, which is commonly used in applications of PWL regression (McCoy *et al.*, 2016; Krasko and Rebennack, 2017). After a translation of the data onto $[0, 1] \times [0, 1]$, we set the error bound to $\xi = 0.1$ (i.e., an accuracy level of 10%). We present the upper and lower bounds of the data, as well as the calculated windows (dotted lines) and the final resulting PWL fuction (as a dashed line), which has four breakpoints (the starred points).

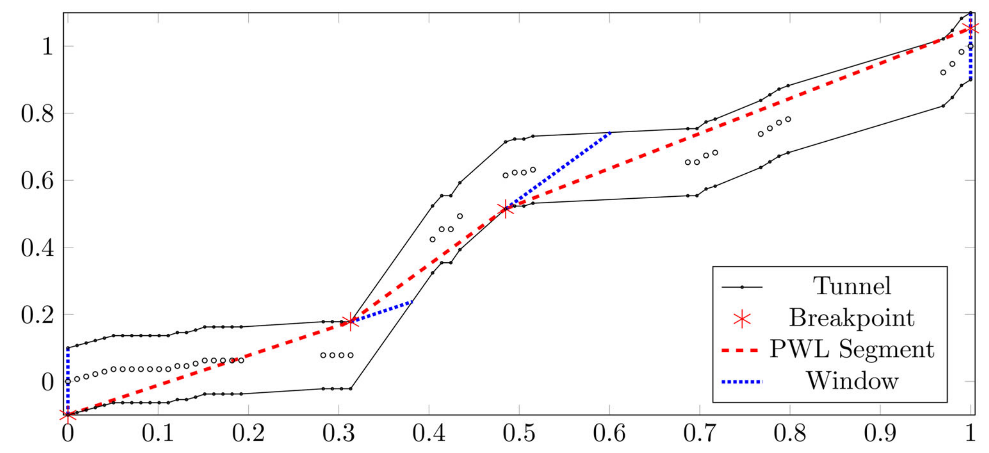


Figure 3. An example of Algorithm 1 on the (translated) *DebrisFlow* data set (McCoy *et al.*, 2016; Krasko and Rebennack, 2017), with $\xi = 0.1$.

3. Applications to fitting continuous univariate functions

Algorithm 1 fits a minimal-breakpoint, ξ-optimal PWL function $f : [x_1, x_n] \to \mathbb{R}$ to a given discrete data set, defined by a set of n tuples $p_i = (x_i, y_i) \in \mathbb{R}^2$, for $i \in [n]$.

This methodology can easily be extended to approaches fitting PWL functions to continuous functions using the adaptive discretization method presented by Rebennack and Kallrath (2015b), which is an iterative approach with the goal of fitting a PWL function to within some error, minimizing the number of breakpoints used. Their presented method iteratively refines a set of discrete data points that are used to approximate the continuous function, and fits the set of data points with a PWL function in each iteration. Rebennack and Krasko (2020) used this framework, where their presented MILP model is used in each iteration to calculate the PWL function. A further approach by Kazda and Li (2021) used a similar adaptive discretization method hybridized with a difference-of-convex approach. Since these models solve the min-ξ problem, we adapt the algorithm to utilize min-B algorithms (in particular, Algorithm 1).

3.1. Fitting continuous univariate functions

Let $f(x) : [\underline{X}, \bar{X}] \to \mathbb{R}$ be a given (univariate) continuous function. A PWL function approximating $f(x)$ is a continuous function $p(x) : [\underline{X}, \bar{X}] \to \mathbb{R}$ with $B \geq 2$ breakpoints such that some distance function E between $f(\cdot)$ and $p(\cdot)$ is optimized. For a given $f(\cdot)$ and $p(\cdot)$, and for a given set of breakpoints $\{X_1^{\text{break}}, \dots, X_B^{\text{break}}\}$, this can be calculated by solving the following problem:

$$\begin{aligned} E(p(x)) :&= \max_{x \in [\underline{X}, \bar{X}]} |p(x) - f(x)| \\ &= \max_{b \in \{1,\dots,B-1\}} \max_{x_s \in \left[X_b^{\text{break}}, X_{b+1}^{\text{break}}\right]} |p(x_s) - f(x_s)| \end{aligned} \tag{1}$$

This problem decomposes into $B-1$ box-constrained, global, univariate optimization problems.

For a given accuracy $\varepsilon > 0$, the problem of fitting a minimal-breakpoint PWL function $p(\cdot)$ to a function $f(\cdot)$ involves solving the following problem, where the value of E is calculated as above in (1):

$$\begin{aligned} B^\star :&= \min B \\ \text{s.t.} \quad & E(p(\cdot)) < \varepsilon \\ & p(\cdot) \text{ is a } B\text{-breakpoint PWL function on } [\underline{X}, \bar{X}] \\ & \text{with slope} \in [\underline{C}, \bar{C}] \text{ and intercept} \in [\underline{D}, \bar{D}]. \end{aligned} \tag{2}$$

Consider some *a priori* discretization of the continuous function $f(\cdot)$, in which a given number of data points are used as a discretization of the domain $[\underline{X}, \bar{X}]$. Then, Algorithm 1 can be used to fit a ξ-optimal PWL function for this discretized data set. Depending on the accuracy E between the resulting PWL function $p(\cdot)$ and the continuous function $f(\cdot)$, we can refine the discretization until E reaches the desired accuracy.

Definition 1. *A finite set $\mathcal{I} = \{X_i \mid i \in [n]\}$ is a discretization of the nonempty compact interval $[\underline{X}, \bar{X}] \subset \mathbb{R}$ if*

$$\underline{X} = X_1 < \dots < X_i < X_{i+1} < \dots < X_n = \bar{X}.$$

A discretization $\mathcal{I}$ of $[\underline{X}, \bar{X}]$ is a refinement of the discretization $\hat{\mathcal{I}}$ if $\hat{\mathcal{I}} \subset \mathcal{I}$.

For a given discretization $\mathcal{I}$ of $[\underline{X}, \bar{X}]$, let $p^{\mathcal{I}}(x)$ be the minimal breakpoint, ξ-optimal PWL function (possibly found by Algorithm 1). The maximum difference between the continuous function $f(x)$ and $p^{\mathcal{I}}(x)$ (denoted $E(p^{\mathcal{I}}(x))$ for the PWL function $p^{\mathcal{I}}(x)$) is likely to be greater than ξ, but never smaller.

Suppose we have that our error tolerance on the calculation of $E(p^{\mathcal{I}}(x))$ is some small constant $\alpha > 0$. In order to ensure that we achieve an optimal PWL function within some given accuracy $\varepsilon > 0$, we can calculate a ξ-optimal PWL function in each iteration (using Algorithm 1) where we set $\xi = \varepsilon - \alpha$. Then, if for our chosen accuracy ε, the following holds:

$$E(p^{\mathcal{I}}(x)) - \xi \leq \alpha, \tag{3}$$

then $p^{\mathcal{I}}(x)$ is an ε-optimal PWL function for $f(x)$ using B breakpoints minimizing the maximum difference.

We summarize the min-B problem for continuous functions in Algorithm 2.

Algorithm 2: Modeling a continuous function with a minimal breakpoint PWL function with maximum error ε

```
Input: Continuous function $f(x) : [\underline{X}, \bar{X}] \to \mathbb{R}$, global tolerance $\alpha > 0$, maximum error tolerance $\varepsilon > \alpha > 0$
Output: Minimal breakpoint PWL function $p(x)$, with $B^\star$ breakpoints and maximum global error $E \le \varepsilon$
1 Initialise: Choose some initial discretization $\mathcal{I} = \mathcal{I}_{\text{init}}$ with $|\mathcal{I}_{\text{init}}| \ge 2$;
2 PWLR: Use Algorithm 1 to compute a minimal breakpoint PWL function $p^{\mathcal{I}}(x)$ (with e.g., $B$ breakpoints) for $\mathcal{I}$ with maximum error $\xi = \varepsilon - \alpha$;
3 Evaluate: Using $p^{\mathcal{I}}(x)$, solve the global optimization problem (1) to $(\alpha/2)$-optimality to obtain $E(p^{\mathcal{I}}(x))$;
4 Check Optimality: if $E(p^{\mathcal{I}}(x)) - \xi \le \alpha/2$ then
5     $p(x) = p^{\mathcal{I}}(x), E = E(p(x)), B^\star = B$, STOP
6 else
7     Refine discretization $\mathcal{I}$ and GOTO 2
```

Typically, we set $\alpha = \varepsilon/5$, which gives a good trade-off between the time required to solve the global optimization problem (1) in Step 3, and the overall runtime of the algorithm.

Remark 1. We now discuss the correctness of the solution with regards to problem (2) for a given continuous function f. For an optimal solution to Algorithm 2 (i.e., with final discretization, say $\mathcal{I}_{\text{opt}}$), let the ε-optimal PWL approximation of f that is found (say p_{opt}) have B_{opt} breakpoints. We know that B_{opt} is minimal for the data set $\mathcal{I}_{\text{opt}}$ from Algorithm 1, see e.g., Hakimi and Schmeichel (1991).

Suppose there exists an ε-optimal PWL function (say $p^\star$) for the given continuous function f with $B^\star$ breakpoints, where $B^\star < B_{\text{opt}}$. Clearly, $p^\star$ would also be ε-optimal for the data set $\mathcal{I}_{\text{opt}}$ (since $\mathcal{I}_{\text{opt}}$ only consists of points from f). However, this contradicts the optimality of Algorithm 1. Hence, we know that Algorithm 2 outputs an ε-optimal PWL function for the continuous function f with minimal breakpoints. $\square$

Step 1 of Algorithm 2 requires an initial discretization of the continuous function. A natural choice is the equidistant placement of data points along the continuous function; that is,

$$\mathcal{I}_{\text{init}} = \left\{ X_i := \underline{X} + (i-1) \cdot \frac{\bar{X} - \underline{X}}{|\mathcal{I}_{\text{init}}| - 1} \quad | \quad \forall i \in [|\mathcal{I}_{\text{init}}|] \right\}.$$

The initial choice of the size of the discretization ($|\mathcal{I}_{\text{init}}|$) is also important. For a given accuracy ξ, Algorithm 1 should be able to calculate the given PWL function such that $B < |\mathcal{I}_{\text{init}}|$ (otherwise the initial computational effort is wasted). This is dependent on the continuous function. We also note that heuristic approaches to finding well-fitting initial discretizations could lead to improvements in the efficiency of the overall algorithm - we leave this problem for future work.

Step 2 requires the implementation of Algorithm 1 on the data set $\mathcal{I}$, to compute a minimal breakpoint PWL function $p^{\mathcal{I}}(x)$ with maximum error ξ. By setting $\xi = \varepsilon - \alpha$, we ensure that the optimality check in step 4 is successful when $E(p^{\mathcal{I}}(x)) - \xi \le \alpha$.

Step 3 requires the evaluation of the PWL function using global optimization techniques. If the function $f(\cdot)$ is differentiable, then gradient descent techniques can be employed. This is due to the fact each linear segment $b \in [B]$ of the PWL function is differentiable (with gradient given by $c_b \in [\underline{C}, \bar{C}]$), and the sum (or difference) of two differentiable functions is always differentiable. Furthermore, if the function $f(\cdot)$ is convex (or concave) over a given segment, then this step can be solved quickly if the changes in convexity of the function are known (Nesterov and Nemirovskii, 1994); in particular, in time $\mathcal{O}(1/\alpha)$ if solved to an accuracy of α. We prove this formally in Section 4.

Step 4 determines whether or not the given PWL approximation is optimal (for the chosen accuracy ε). If not, the set of data points $\mathcal{I}$ needs to be refined (line 7 of Algorithm 2), which crucially affects the computational performance. By adding too many data points, the number of iterations is reduced, yet the complexity of the PWL regression is increased (since it runs in $\mathcal{O}(|\mathcal{I}|)$ time). We use the refinement strategy suggested by Rebennack and Krasko (2020), which utilizes the solution of the global optimization problem (1). For each of the B – 1 box-constrained, global, univariate problems ($b \in [B-1]$), the term

$$\max_{x_s \in \left[X_b^{\text{break}}, X_{b+1}^{\text{break}}\right]} |p(x_s) - f(x_s)|$$

is known from solving (1). Therefore, since this is an absolute value problem, the solution to (1) includes calculating the terms

$$E_b^+ := \max_{x_s \in \left[X_b^{\text{break}}, X_{b+1}^{\text{break}}\right]} p(x_s) - f(x_s);$$

$$E_b^- := \min_{x_s \in \left[X_b^{\text{break}}, X_{b+1}^{\text{break}}\right]} p(x_s) - f(x_s).$$

Let x_b^+ and x_b^- be the arguments of the optimal solutions to the above two equations, respectively. In the refinement step on line 7 of Algorithm 2, we add x_b^+ and x_b^- to $\mathcal{I}$ if the following hold (respectively):

$$E_b^+ \ge \frac{5\xi}{6}; \quad -E_b^- \ge \frac{5\xi}{6}. \tag{4}$$

Hence, we are adding at most $2(B-1)$ new data points in each iteration, where B is the number of breakpoints in the current PWL function.

We establish the finiteness of Algorithm 2 in Theorem 1, which is based on Rebennack and Krasko (2020, Theorem 3). Since Algorithm 2 makes use of Algorithm 1 (for the min-B problem), the final steps of the proof of Theorem 1 differ to those presented by Rebennack and Krasko (2020), since the accuracy of the modeled PWL function in each iteration is guaranteed.

Theorem 1. *Let $f(x) : [\underline{X}, \bar{X}] \to \mathbb{R}$ be a continuous function on $[\underline{X}, \bar{X}] \neq \emptyset$, and let bounds on the slope $[\underline{C}, \bar{C}]$ and intercept $[\underline{D}, \bar{D}]$ be given. Then Algorithm 2 computes a continuous ε-optimal PWL function approximation of f, $p(x) : [\underline{X}, \bar{X}] \to \mathbb{R}$ with slope $\in [\underline{C}, \bar{C}]$ and intercept $\in [\underline{D}, \bar{D}]$, in a finite number of iterations if the refinement strategy admits an arbitrarily small distance between consecutive data points.*

Proof. As a first statement, we state that the refinement strategy allows discretizations $\mathcal{I}$ with arbitrarily small distances between consecutive data points. That is, for every constant $\hat{h} > 0$, there exists a discretization $\hat{\mathcal{I}}$ with $\max_{i \in [|\hat{\mathcal{I}}|-1]} \{X_{i+1} - X_i\} \leq \hat{h}$, where X_i is defined as in Definition 1 for the given $\hat{\mathcal{I}}$. Such a discretization will always be constructed by Algorithm 2, if it does not converge before.

As a second statement, for a fixed $\mathcal{I}$, the difference between $f(\cdot)$ and $p^{\mathcal{I}}(\cdot)$ in the interval $[X_i, X_{i+1}]$ is bounded by

$$|f(x) - p^{\mathcal{I}}(x)| \leq \xi + \bar{C} \cdot (X_{i+1} - X_i) + \sup\{|f(u) - f(v)| : X_i \leq u, v \leq X_{i+1}\}.$$

The right-hand side term ξ is given by the solution to Algorithm 1. $\bar{C}(X_{i+1} - X_i)$ is the extreme value which any PWL function can take (the upper bound on the slope $\bar{C}$ is ensured by Algorithm 1), and the right-most term is the modulus of continuity. Because $f(\cdot)$ is continuous over the compact interval $[\underline{X}, \bar{X}]$, it is also uniformly continuous on $[\underline{X}, \bar{X}]$ according to the Heine–Cantor theorem.

Combined, the first and second statements imply that for every $\alpha > 0$, there exists an $\hat{h} > 0$ and a finite discretization $\hat{\mathcal{I}}$ such that

$$\bar{C} \cdot (X_{i+1} - X_i) + \sup\{|f(u) - f(v)| : X_i \leq u, v \leq X_{i+1}\} \leq \alpha, \quad \forall i \in \hat{\mathcal{I}} \setminus \bar{X}. \tag{5}$$

For $\hat{\mathcal{I}}$, we obtain: $E(p^{\hat{\mathcal{I}}}(x)) = |f(x) - p^{\hat{\mathcal{I}}}(x)| \leq \xi + \alpha$. Hence, by setting $\xi = \varepsilon - \alpha$ in Algorithm 2, we have that: $|f(x) - p^{\hat{\mathcal{I}}}(x)| \leq \varepsilon$, and hence $p^{\hat{\mathcal{I}}}(x)$ is an ε-optimal minimal breakpoint continuous PWL function that is computed in finitely many iterations.

However, note further that since for a given $\hat{\mathcal{I}}$, we calculate $E(p^{\hat{\mathcal{I}}}(x))$ to within an accuracy of $\alpha/2$. Since the first and second statements imply that (5) also holds for an arbitrary accuracy $\alpha/2$, then if $E(p^{\hat{\mathcal{I}}}(x)) = |f(x) - p^{\hat{\mathcal{I}}}(x)| \leq \xi + \alpha/2$ holds, we are guaranteed that $p^{\hat{\mathcal{I}}}(x)$ is an ε-optimal minimal breakpoint continuous PWL function (which is computed in finitely many iterations). □

Note that the discussed refinement strategy above (i.e., (4)) allows arbitrarily small distances between data points where computed PWL functions violate the fitting criteria. As such, the presented refinement strategy is valid and leads to a finitely convergent algorithm for fitting continuous PWL functions to continuous univariate functions. Furthermore, the bounds on the slope and intercept can be calculated in the same way presented by Rebennack and Krasko (2020) (i.e., by considering extreme slopes interpolated between the data points).

As a further point, we note that since applying Algorithm 2 provides an ε-optimal PWL approximation of a given continuous function f, an approximating polygon can be constructed that fully contains f by translating the PWL function above and below by ε.

3.2. Fitting convex functions

If the input function is convex, it is important that the resulting PWL function is also convex for optimization purposes in order to preserve the favorable property of convex functions, e.g., for problem minimization (Horst *et al.*, 2000). Furthermore, convex PWL functions have the special property whereby the function value is given by the maximum value among all of the linear segments (Rebennack, 2016b, Lohmann and Rebennack, 2017). That is, the function value y_i at data point $i \in [I]$ satisfies

$$y_i = \max_{b \in [B-1]} c_b X_i + d_b \quad \forall i \in [I].$$

This is a desirable property that can be exploited when approximating non-linear functions.

We now discuss how Algorithm 2 performs when the function being modeled is convex. We restrict our discussion in this section to convex functions; however, we note that analogous results apply also for concave functions.

Theorem 2 shows that if the function f is convex, then the PWL function found by Algorithm 2 is also guaranteed to be convex.

Theorem 2. *Let $f(x) : [\underline{X}, \bar{X}]$ be a continuous, convex function on $[\underline{X}, \bar{X}] \neq \emptyset$. Then, the ε-optimal PWL function $p(x) : [\underline{X}, \bar{X}] \to \mathbb{R}$ approximating f that is outputted by Algorithm 2 will also be convex on $[\underline{X}, \bar{X}]$.*

Proof. For a convex function f, let $p := \{p_j \mid j \in [n]\}$ denote the set of data points used to approximate the function. The function formed by connecting adjacent data points in p with a straight line maintains the same convex relationship as the function f. If further data points from f are included in the set, the same convex relationship will hold. Hence, if we are able to show for such a set of data points that Algorithm 1 outputs a convex PWL function, this will prove the theorem statement and guarantee the convexity of the resulting PWL function found by Algorithm 2.

Consider the PWL function found by Algorithm 1 for such a data set. Algorithm 1 only adds a new breakpoint to the PWL function when the convex hulls of $p^+ := \{p_j^+ \mid j \in [n]\}$ and $p^- := \{p_j^- \mid j \in [n]\}$ intersect. These convex hulls are updated iteratively in each iteration $i \in [n]$. We define the *upper* (and *lower*) convex hulls (of a convex hull) to be those data points on the hull that lie above (or below) the straight line connecting its leftmost and rightmost points respectively. Note that the leftmost and rightmost points are part of both the upper and lower convex hulls. In particular, in iteration i $(3 \leq i \leq n)$, the *lower* convex hull of p^+ will contain all points of p^+ (i.e., $p_1^+, \ldots, p_i^+$), while the *upper* convex hull of p^- will consist of a straight line connecting p_1^- and p_i^-. The calculation of the convex hulls restarts whenever a breakpoint is calculated; however, the lower convex hull of p^+ will consist of all the data points that are currently being considered, while the upper convex hull of p^- will consist of the first and last data point. Therefore, any intersection will occur when the upper convex hull of p^- overlaps into the lower convex hull of p^+ (and not the other way around).

There are two *separating lines* formed by the convex hulls, which are calculated as the longest straight lines that connect points on different hulls (i.e., upper to lower, and lower to upper) that do not fall inside either hull - see Imai and Iri (1986) for a formal definition. At some point between the two end points, these lines intersect.

Recall that the initial window consists of the line connecting the points p_1^+ and p_1^-. When the two convex hulls intersect, the breakpoint is calculated as the intersection of the current window and the relevant (upper or lower) separating line. Since the nature of the overlap of the two convex hulls will always be the same, the nature of the relevant supporting line will always be the same. In particular, the breakpoint will always be found on the rightmost point of the window, exactly where it intersects with the lower tunnel. Any line connecting the breakpoints that coincide with the lower tunnel will be convex by definition.

The above only holds while the final window is not visible from the current window. For the penultimate breakpoint, since the previous breakpoint will be on the lower tunnel, the convexity of the PWL function will not be affected up until this breakpoint. The final breakpoint is found on the window $[p_n^-, p_n^+]$, and therefore the convexity will be maintained since the slope of the PWL function connecting the final two breakpoints is guaranteed to increase in comparison to the previous slope, due to the convex nature of the data points. □

Corollary 1. *Let $f(x) : [\underline{X}, \bar{X}]$ be a continuous, convex function on $[\underline{X}, \bar{X}] \neq \emptyset$. Then, there exists an ε-optimal convex PWL function $p : [\underline{X}, \bar{X}] \to \mathbb{R}$ approximating f.*

Note that for a given convex function *f*, there may also be ε-optimal PWL functions approximating *f* that are not convex.

Figure 4 shows an example of Algorithm 1 applied to a data set taken from the convex function x^2 on the interval $[-3.5, 3.5]$ (with $\xi = 1$). From this, it can be noticed that from a given window, the next window is exactly the intersection of the points in the tunnel that are visible and not visible from the rightmost point of the window (i.e., the calculated breakpoint). Therefore, the interior windows coincide with the segments of the calculated PWL function.

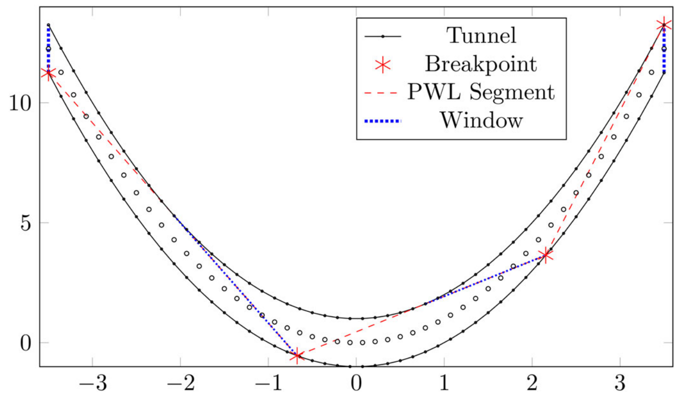


Figure 4. Algorithm 1 applied to the convex function x^2, with $\xi = 1$.

4. Runtime bounds for fitting continuous functions

Algorithm 2 makes use of the linear time algorithm (Algorithm 1) presented by Imai and Iri (1986) and refined by Hakimi and Schmeichel (1991). It is further possible that there are classes of functions, or requirements on the input, where the runtime of Algorithm 2 can be bounded. The efficient piecewise linearization of such function classes would allow faster solution times when using MILP approaches to solve complicated MINLP problems in which these functions are inherent.

Algorithm 2 receives as input a continuous function $f : [\underline{X}, \bar{X}] \to \mathbb{R}$, alongside a domain it is defined over. That is, we are able to evaluate the function *f* at any point over its given domain. In order to provide guaranteed runtime bounds for Algorithm 2, we require bounds on the following actions:

1. The runtime required to solve the global optimization problem (1).
2. The number of iterations of Algorithm 2.

In order to bound the runtime of solving the global optimization problem (1), we require assumptions on two further aspects of the function *f*. First, that the function *f* is Lipschitz continuous, and second, that we know the *change-in-convexity* points (i.e., the points at which the convexity of the given function changes). In order for the bound presented in Lemma 1 to hold, we assume such points are given as input if *f* is non-convex; otherwise, there are none.

Lemma 1. *Let $f(x) : [x_0, x_n] \to \mathbb{R}$ be a Lipschitz-continuous function with $K \geq 0$ change-in-convexity points. If the K change-in-convexity points of f are known, then each iteration of Algorithm 2 applied to the function f takes time at most*

$$\mathcal{O}\left(n + \frac{B+K}{\alpha}\right)$$

if the global optimization problem (1) is solved to $(\alpha/2)$-optimality, where B is the number of breakpoints found by Algorithm 1 for the given discretization $\mathcal{I}$ (with $|\mathcal{I}| = n$ data points).

Proof. Consider an iteration of Algorithm 2 (i.e., steps 2-7). The linear runtime of step 2 was proved by Imai and Iri (1986) and Hakimi and Schmeichel (1991). The refinement of the discretization in step 7 considers the addition of at most $2(B-1)$ new data points. Since $B \leq n$ (typically, $B \ll n$), this step will also take time at most $\mathcal{O}(n)$.

We can bound the runtime of the global optimization problem in step 3 if we know that each of the $B - 1$ box-constrained problems in (1) considers the difference between a linear function and a convex function (or a concave function; for brevity, we refer only to convex functions in the following text). Note that in the case of a concave function, we can consider the term $-(p(x_s) - f(x_s))$ in the objective of the optimization problem (1) instead, which will be convex.

In particular, if we know that each segment of f is convex, the optimization problem (1) reduces to solving two problems (for E_b^+ and E_b^- as defined in Section 3.1). One requires solving a global convex optimization problem (since the difference between a convex function and linear function is convex) to $(\alpha/2)$-optimality, which can be done in $\mathcal{O}(1/\alpha)$ time for Lipschitz-continuous functions (see e.g., Nesterov and Nemirovskii (1994)). The second is the maximization of a convex function over an interval, which is trivial.

In order to know whether or not a given segment of f is convex or concave (without a complicated mathematical analysis), we need to know the change-in-convexity points of f (which can be given as input to Algorithm 2). Such points can then be used to separate the $B - 1$ box-constrained problems in (1) into a total of $K + B - 1$ convex optimization problems, which can each be solved in $\mathcal{O}(1/\alpha)$ time. Therefore, step 3 of Algorithm 2 can be solved in time $\mathcal{O}((B+K)/\alpha)$. Hence, each iteration of Algorithm 2 will take time at most

$$\mathcal{O}\left(n + \frac{B+K}{\alpha}\right).$$

□

For the majority of functions, we expect that $K \le B \le n$. Therefore, a more conservative bound for the runtime found by Lemma 1 is $\mathcal{O}(n/\alpha)$.

In order to bound the total number of iterations required by Algorithm 2 to find an optimal PWL approximation of the given function, we can find a bound on the number of data points required for an optimal PWL approximation by a provably worse algorithm. Since we know that at least one new data point is added in each iteration if the optimal approximation has not been found, this number will also be a bound on the number of iterations required. Lemma 2 presents this bound as a function of the term $\sqrt{\max_{[x_0,x_n]}|f''(x)|}$, which we assume we have access to; this is only necessary to present the given result, and is not required as input for Algorithm 2 in order to find the approximating PWL function.

Lemma 2. *Let $f(x) : [x_0, x_n] \to \mathbb{R}$ be a twice-differentiable function, where $\sqrt{\max_{[x_0,x_n]}|f''(x)|} = \mathcal{O}(g)$ holds for some function g, and let $D_n = x_n - x_0$ be the length of the interval over which f is defined. Then, Algorithm 2 requires at most*

$$\mathcal{O}\left(\frac{D_n \cdot g}{\sqrt{\xi}}\right)$$

iterations before a ξ-optimal PWL function approximating f is found.

Proof. The maximum absolute difference between the approximated PWL function and the continuous function f (found by the global optimization problem in step 3) decreases in every iteration of Algorithm 2. The final set of data points leading to the optimal PWL function will include the data points from the initial discretization, the data points on the continuous function that led to the maximum error from previous iterations, and possibly some more.

The discretization of the continuous function f in step 1 of Algorithm 2 begins with a simple linear interpolation (i.e., taking equally distributed data points from the continuous function). It is possible that the number of data points in the initial discretization is enough such that an optimal PWL function can be found in a single iteration by Algorithm 1 in step 2. Let N_{init} be the minimal number of data points needed such that the initialized discretization produces an optimal PWL function with regards to the stated error bound in the first iteration.

For our given choice of $|I_{\text{init}}| \ll N_{\text{init}}$, we know that the number of data points in the final set leading to an optimal PWL function will be fewer than N_{init}, by the design of the refinement in line 7 of Algorithm 2 (i.e., that points on f leading to the maximum error from previous iterations are included in the data set for future iterations).

Therefore, we require a bound on the number of (equally spaced) data points such that an approximation using these data points is optimal (i.e., a bound on N_{init}). This in turn will give a bound on the number of iterations required to find an optimal approximating PWL function, since we know that at least one data point is added in each iteration before the optimal is found.

Consider a number of equally spaced data points taken from the continuous function f. A linear interpolation between the data points will be a worse approximation of f than the solution found by Algorithm 1. Hence, a bound on the number of data points required such that the linear interpolation leads to an optimal PWL function will also bound the number of iterations required by Algorithm 2 (by reusing the same logic as above).

If the given function f is twice-differentiable, then the error of interpolating between the points x_i and x_j (for any $x_i < x_j \in [x_0, x_n]$), denoted by R_T, is bounded by

$$|R_T| \le \frac{(x_j - x_i)^2}{8} \cdot \max_{x_i \le x \le x_j} |f''(x)|$$

by Rolle's theorem (see e.g., Meijering (2002)).

Since we assume an equal distribution of the n data points, we have that $h_n = x_{i+1} - x_i$ is constant for any consecutive data points. In particular, let $D_n = x_n - x_0$ be the length of the interval $[x_0, x_n]$ over which f is defined. Then, for a given number of data points n, we have that $h_n = D_n/n$ and we are searching for n such that

$$|R_T| \le \frac{(D_n/n)^2}{8} \cdot \max_{x_i \le x \le x_j} |f''(x)| \le \xi - \alpha$$

holds. That is,

$$n^2 \geq \frac{D_n^2}{8(\xi - \alpha)} \cdot \max_{x_i \leq x \leq x_j} |f''(x)|$$
$$\Rightarrow n \geq \frac{D_n}{\sqrt{8(\xi - \alpha)}} \cdot \sqrt{\max_{x_i \leq x \leq x_j} |f''(x)|} = \hat{n}.$$

This value of $\hat{n}$ is an upper bound on the number of data points required such that Algorithm 1 can find an optimal PWL approximation of f in step 2 of Algorithm 2. Furthermore, since at least one new data point is added in each iteration (if the optimal solution has not been found), $\hat{n}$ is an upper bound on the number of iterations required to find an optimal PWL function.

Suppose $\sqrt{\max_{x_i \leq x \leq x_j} |f''(x)|} = \mathcal{O}(g)$ for some function g. Since we set $\alpha < \xi$, we have the following bound on the number of iterations:

$$\hat{n} = \mathcal{O}\left(\frac{D_n \cdot g}{\sqrt{\xi}}\right).$$

Typically, we do not expect g to be too large (for example, for a quadratically constrained function f, we will have $g = \mathcal{O}(1)$).

Overall, we bound the runtime of Algorithm 2 in Theorem 3.

Theorem 3. *Let $f(x) : [x_0, x_n] \to \mathbb{R}$ be a continuous, twice-differentiable function with K known change-in-convexity points, where $\sqrt{\max_{[x_0, x_n]} |f''(x)|} = \mathcal{O}(g)$ holds for some function g. Let $D_n = x_n - x_0$ be the length of the interval over which f is defined. Then, the runtime required by Algorithm 2 to find a ξ-optimal continuous PWL approximation of f is bounded by*

$$\mathcal{O}\left(\frac{D_n \cdot g}{\sqrt{\xi}} \cdot \left(n^\star + \frac{B^\star + K}{\xi}\right)\right),$$

where $B^\star$ is the number of breakpoints in the optimal PWL function for the given discretization of $n^\star$ data points.

Proof. The number of data points $n^\star$ in the final set of data points is an upper bound on the value of n from Lemma 1, while $B^\star$ is an upper bound on the number of breakpoints. Hence, by Lemma 1, the runtime of each iteration of Algorithm 2 is bounded from above by $\mathcal{O}(n^\star + (B^\star + K)/\alpha) = \mathcal{O}(n^\star + (B^\star + K)/\xi)$ if the K change-in-convexity points are known, since we have $\alpha = \Theta(\xi)$. By Lemma 2, the total number of iterations before Algorithm 2 finds an optimal PWL function is bounded from above by $\mathcal{O}(D_n \cdot g/\sqrt{\xi})$. Therefore, the overall runtime of Algorithm 2 is found by multiplying the two terms. □

Theorem 3 suggests that the main factors that determine the efficiency of Algorithm 2 are the length of the domain interval, the desired accuracy, and how many data points are required to fit the model well. Regarding the latter point, this further suggests that complicated functions (i.e., those with multiple inflection points and local optima) would take longer to model due to the increased complexity of the required PWL function, whereas simple functions (i.e., quadratically constrained functions, and convex functions which require no extra input) can be modeled very quickly.

5. Computational results

In order to assess the effectiveness of Algorithm 2, we implemented the algorithm in C++. The global optimization problems are solved using the locally biased variant of the DIRECT algorithm (Gablonsky and Kelley, 2001) which is a global solver embedded within C++ using the NLopt optimization software (Johnson, 2021) with standard solver settings. The experiments in this section were run on an Intel 3.00 GHz machine with 16 GB of RAM.

5.1. Continuous functions

Firstly, we present computational results for 15 different univariate, continuous functions taken from the literature (Rebennack and Kallrath, 2015b; Rebennack and Krasko, 2020). The functions and their domains are listed in Section B of the appendix. In Figures 5 and 6, we present the functions alongside an ε-optimal, minimal breakpoint PWL which models the function (with $\varepsilon = 0.1$).

We consider four values for ε (0.1, 0.05, 0.01 and 0.005). We compare Algorithm 2 with the runtimes for the MILP models presented by Kong and Maravelias (2020) Rebennack and Krasko (2020), and the decomposition approach presented by Warwicker and Rebennack (2023b)[1] (we implemented these models ourselves with the suggested settings), as well as presenting the results from Rebennack and Kallrath (2015b).

Firstly, in Table 2, we present the optimal number of breakpoints for the resulting PWL function found by Algorithm 2 for each of the 40 tested instances across the first 10 functions. We compare this with the state-of-the-art results presented by Rebennack and Kallrath (2015b) and Rebennack and Krasko (2020). In particular, we note that Algorithm 2 is able to solve six of these benchmark instances for the first time.

In Table 3, $B^\star$ shows the minimum number of breakpoints required for the PWL function for each instance. The next four columns show the computational results of this article. We present the number of iterations of Algorithm 2, the overall runtime (as well as the overall time taken by Algorithm 1 (Local) and the global optimizer (Global) for each run). The next three columns present comparative runtime results from the literature, whereas the final column presents the speedup of Algorithm 2 compared with the fastest of the three compared algorithms.

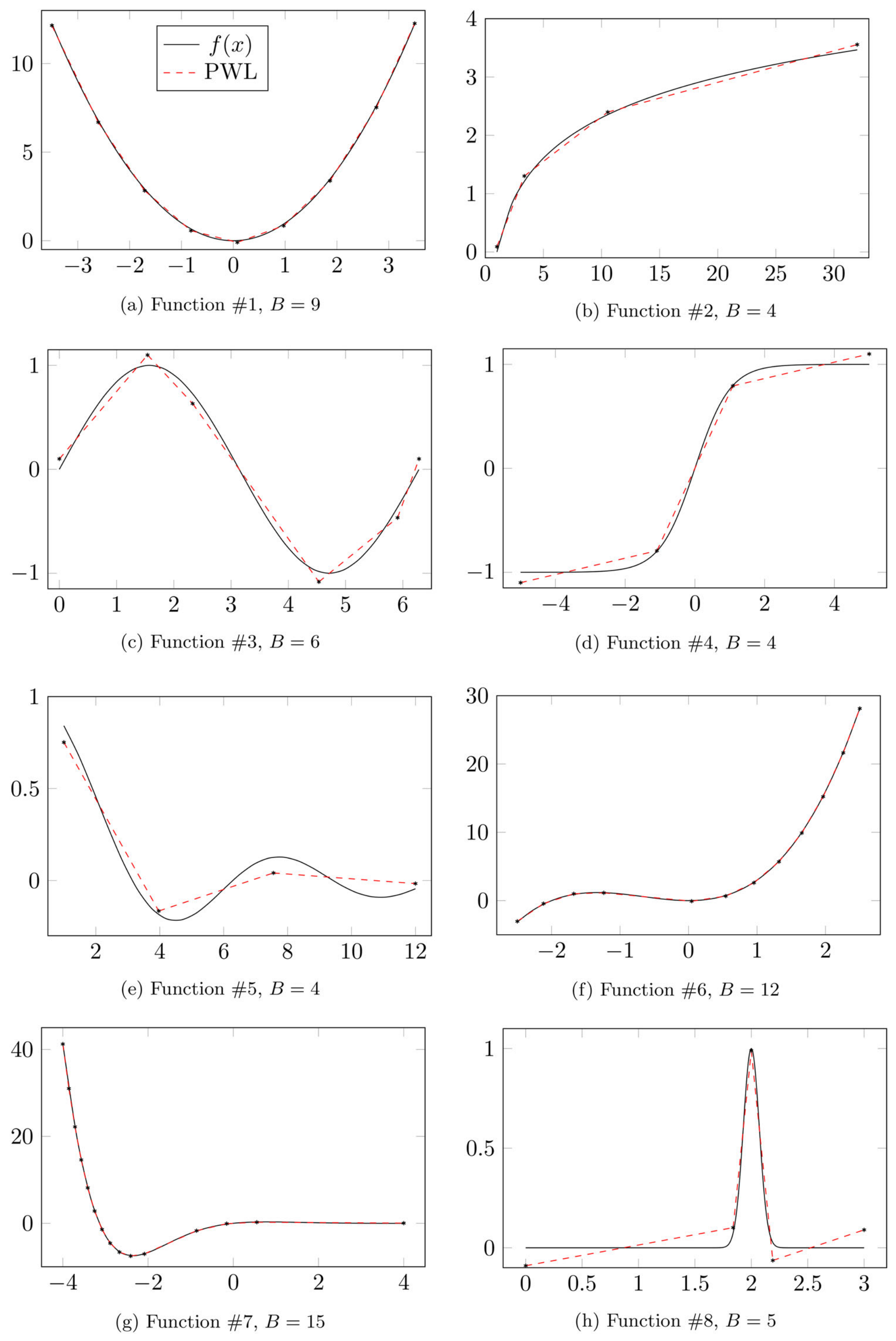
From Table 3, we see that Algorithm 2 is significantly faster than the state-of-the-art algorithms in each of the 40 instances; it is at least 11 times faster on each instance, up to a maximum speedup of over 328,000. In particular, it is able to find an optimal solution in less than 1 second for all but one case when $\xi \geq 0.01$. Since the optimization performed by Algorithm 1 is incredibly fast compared to the MILP approaches, we note that the results are

[1]Note that the advantage of the approach from Warwicker and Rebennack (2023b) is for outlier detection; nevertheless, we present a comparison with the presented decomposition approach without any outlier detection embedded.

constrained only by the global optimizer, which is also very fast.

There are nine instances where none of the three compared algorithms are able to find the optimal PWL function (either due to exceeding the time limit or available memory). However, Algorithm 2 is able to find the optimal PWL function in all of these cases within 10.58 seconds. For instances where both algorithms are able to find a solution within the given time limit, Algorithm 2 is on average more than 36,000 times faster than the approach from Rebennack and Krasko (2020), more than 79,000 times faster than the approach from Kong and Maravelias (2020), and more than 6700 times faster than the approach from Warwicker and Rebennack (2023b).

Figure 7 shows the fraction of instances that are solved within a given time limit for the four compared approaches. For any time budget (above 0.001 seconds), Algorithm 2 is able to solve a greater fraction of instances within the time limit, and is able to solve all instances to optimality within



(a) Function #1, $B = 9$

(b) Function #2, $B = 4$

(c) Function #3, $B = 6$

(d) Function #4, $B = 4$

(e) Function #5, $B = 4$

(f) Function #6, $B = 12$

(g) Function #7, $B = 15$

(h) Function #8, $B = 5$

Figure 5. Univariate continuous functions and ε-optimal ($\varepsilon = 0.1$) minimal breakpoint PWL functions. Breakpoint locations are marked as black dots.

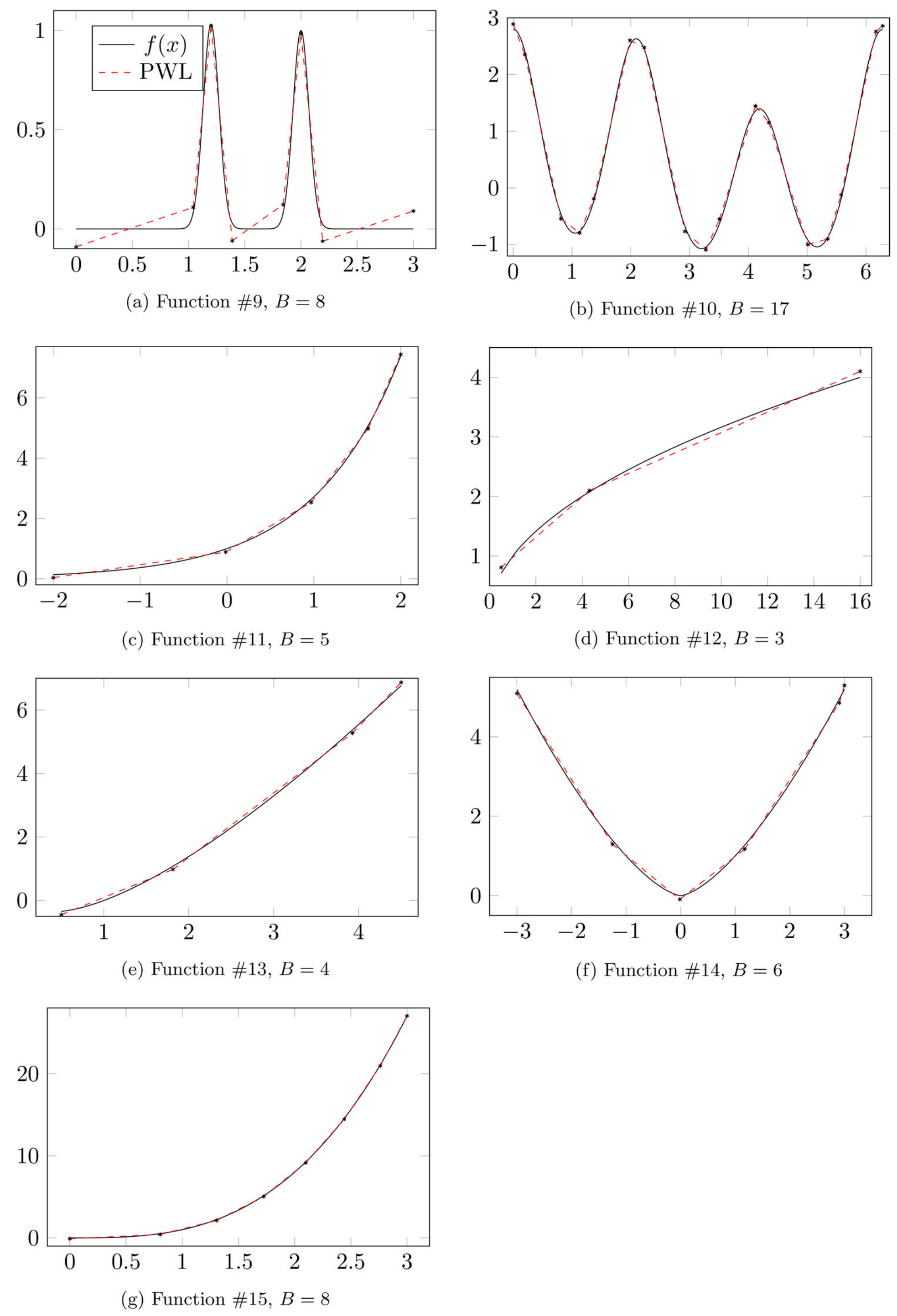


Figure 6. Univariate continuous functions and ε-optimal ($\varepsilon = 0.1$) minimal breakpoint PWL functions (cont.). Breakpoint locations are marked as black dots.

10.58 seconds. Within this time budget, the other three approaches are only able to solve up to 27.5% of the given instances. Therefore, we can clearly see the significant improvement presented by Algorithm 2 for fitting PWL regression functions to univariate continuous functions.

5.2. Convex functions

Table 5 in Section C of the appendix presents a comparison of Algorithm 2 with the approach for fitting convex PWL functions presented by Toriello and Vielma (2012). For this comparison, we consider the convex and concave functions 1, 2 and 11-15 from Figures 5 and 6 (whose functions and domains are listed in Section B of the appendix).

Again, we see that Algorithm 2 is significantly faster than the state-of-the-art approach. There are seven instances where the approach from Toriello and Vielma (2012) is unable to find the optimal solution within the given time limit, whereas Algorithm 2 solves all of these instances within 5.56 seconds. In particular, Algorithm 2 is at least

Table 2. Optimal number of breakpoints required for the given accuracy. MF94 refers to the function from Maranas and Floudas (1994). RK15 and RK20 refer respectively to the results from (Rebennack and Kallrath, 2015b; Rebennack and Krasko, 2020). Boldface indicates instances that are solved for the first time in this article.

Data Set	*Accuracy*	*B**	*RK20*	*RK15*	*Data Set*	*Accuracy*	*B**	*RK20*	*RK15*
(1) x^2	0.1	9	9	9	(6) $2x^2 + x^3$	0.1	12	12	12
$x \in [-3.5, 3.5]$	0.05	13	13	13	$x \in [-2.5, 2.5]$	0.05	16	16	16
	0.01	26	26	26		0.01	**35**	$[32, \infty]$	[16,35]
	0.005	36	36	36		0.005	**48**	$[41, \infty]$	[16,48]
(2) $\ln(x)$	0.1	4	4	4	(7) $e^{-x}\sin(x)$	0.1	15	15	[5,15]
$x \in [1, 32]$	0.05	5	5	5	$x \in [-4, 4]$	0.05	20	20	[5,20]
	0.01	10	10	10		0.01	**44**	$[35, \infty]$	[5,44]
	0.005	14	14	14		0.005	**62**	$[36, \infty]$	[5,62]
(3) $\sin(x)$	0.1	6	6	6	(8) $e^{-100(x-2)^2}$	0.1	5	5	5
$x \in [0, 2\pi]$	0.05	6	6	6	$x \in [0, 3]$	0.05	6	6	[5,7]
	0.01	14	14	14		0.01	12	12	[5,12]
	0.005	18	18	18		0.005	15	15	[5,15]
(4) $\tanh(x)$	0.1	4	4	4	(9) $1.03e^{-100(x-1.2)^2}$	0.1	8	8	8
$x \in [-5, 5]$	0.05	6	6	6	$+e^{-100(x-2)^2}$	0.05	10	10	[8,12]
	0.01	10	10	10	$x \in [0, 3]$	0.01	22	22	[8,22]
	0.005	14	14	14		0.005	28	28	[8,29]
(5) $\sin(x)/x$	0.1	4	4	4	(10) MF94	0.1	17	17	[4,17]
$x \in [1, 12]$	0.05	6	6	6	$x \in [0, 2\pi]$	0.05	22	22	[4,23]
	0.01	10	10	10		0.01	**46**	$[43, \infty]$	[4,46]
	0.005	13	13	13		0.005	**67**	$[51, \infty]$	[4,67]

Table 3. Experimental results. MF94 refers to the function from Maranas and Floudas (1994). RK20, KM20, and WR22 refer respectively to the results from Rebennack and Krasko (2020), Kong and Maravelias (2020), Warwicker and Rebennack (2023b). † indicates that the optimal solution could not be found within 86,400 seconds (i.e., 1 day). ⋆ indicates the approach failed due to exceeding the memory limit (16GB). Speedup represents the ratio of the fastest approach to Algorithm 2; a value > 1 indicates that Algorithm 2 is fastest.

				Time (s)						
Data Set	*Accuracy*	*B**	*Iter*	*Local*	*Global*	*Overall*	*RK20*	*KM20*	*WR22*	*Speedup*
(1) x^2	0.1	9	4	0.001	0.002	**0.013**	14.60	25.68	10.38	798
$x \in [-3.5, 3.5]$	0.05	13	7	0.003	0.033	**0.080**	167.59	2019.83	159.98	2000
	0.01	26	7	0.002	0.246	**0.263**	†	†	⋆	$> 328,517$
	0.005	36	8	0.012	5.51	**5.56**	†	†	⋆	$> 15,540$
(2) $\ln(x)$	0.1	4	4	–	0.002	**0.005**	0.193	0.336	0.909	39
$x \in [1, 32]$	0.05	5	4	–	0.001	**0.008**	0.629	1.11	2.23	79
	0.01	10	6	–	0.094	**0.121**	51.53	230.63	115.96	426
	0.005	14	7	–	1.96	**1.99**	988.48	4957.97	208.96	105
(3) $\sin(x)$	0.1	6	3	–	–	**0.010**	0.138	0.344	0.720	14
$x \in [0, 2\pi]$	0.05	6	3	–	0.008	**0.017**	0.775	1.75	2.32	46
	0.01	14	5	–	0.118	**0.133**	239.97	9881.03	325.08	1804
	0.005	18	7	–	2.46	**2.49**	†	†	⋆	$> 34,699$
(4) $\tanh(x)$	0.1	4	3	–	–	**0.011**	0.198	0.274	0.607	18
$x \in [-5, 5]$	0.05	6	3	–	0.007	**0.014**	0.147	2.20	1.76	11
	0.01	10	4	–	0.076	**0.084**	15.61	90.83	187.52	186
	0.005	14	5	–	1.36	**1.38**	286.88	689.77	532.05	208
(5) $\sin(x)/x$	0.1	4	4	–	–	**0.001**	0.103	0.091	0.253	103
$x \in [1, 12]$	0.05	6	4	–	0.001	**0.002**	0.566	0.323	0.617	283
	0.01	10	5	–	0.069	**0.081**	18.22	30.89	14.31	177
	0.005	13	7	0.002	1.43	**1.47**	123.07	990.69	179.45	84
(6) $2x^2 + x^3$	0.1	12	4	–	–	**0.003**	56.82	494.09	222.31	18,940
$x \in [-2.5, 2.5]$	0.05	16	5	–	0.010	**0.019**	1807.48	30,213.70	431.77	22,725
	0.01	35	4	0.002	0.353	**0.385**	†	†	⋆	$> 224,416$
	0.005	48	5	0.015	7.60	**7.65**	†	†	⋆	$> 11,294$
(7) $e^{-x}\sin(x)$	0.1	15	6	0.001	–	**0.018**	8179.06	†	656.77	36,487
$x \in [-4, 4]$	0.05	20	7	–	0.030	**0.065**	26,038.26	†	1845.62	28,394
	0.01	44	7	0.014	0.641	**0.702**	†	†	⋆	$> 123,077$
	0.005	62	12	0.017	9.89	**9.95**	†	†	⋆	> 8683
(8) $e^{-100(x-2)^2}$	0.1	5	4	–	–	**0.007**	0.224	1.05	1.00	32
$x \in [0, 3]$	0.05	6	4	–	0.002	**0.009**	1.41	1.64	1.34	149
	0.01	12	4	0.001	0.104	**0.122**	176.68	2122.90	192.58	1448
	0.005	15	6	–	2.16	**2.18**	18,964.50	†	473.52	217
(9) $1.03e^{-100(x-1.2)^2}$	0.1	8	6	–	–	**0.009**	1.56	4.71	18.13	173
$+e^{-100(x-2)^2}$	0.05	10	4	–	0.005	**0.011**	31.78	65.98	23.34	2890
$x \in [0, 3]$	0.01	22	5	–	0.207	**0.229**	18,564.82	35,682.20	2854.95	12,467
	0.005	28	6	0.003	4.57	**4.61**	46,852.98	†	3854.62	834
(10) MF94	0.1	17	7	0.001	0.001	**0.034**	621.73	2835.53	353.03	10,383
$x \in [0, 2\pi]$	0.05	22	5	0.001	0.102	**0.187**	7431.55	5015.80	1215.60	6501
	0.01	46	6	0.024	1.20	**1.28**	†	†	⋆	$> 67,500$
	0.005	67	10	0.011	10.53	**10.58**	†	†	⋆	> 8166

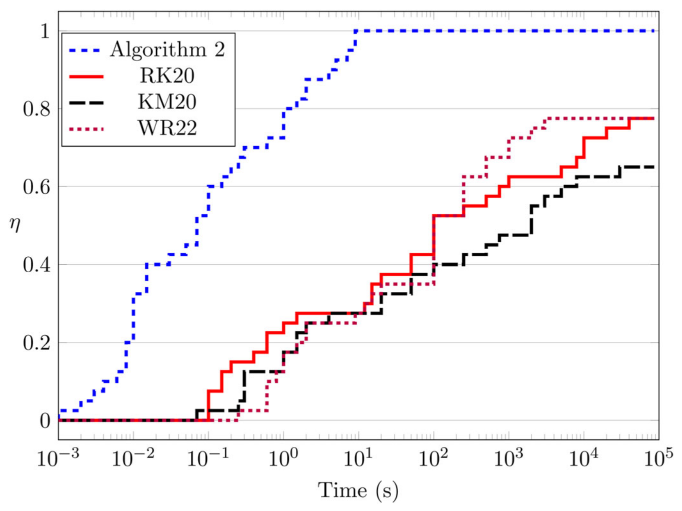


Figure 7. Continuous functions: η refers to the fraction of instances solved to optimality within the given time (logarithmic scale).

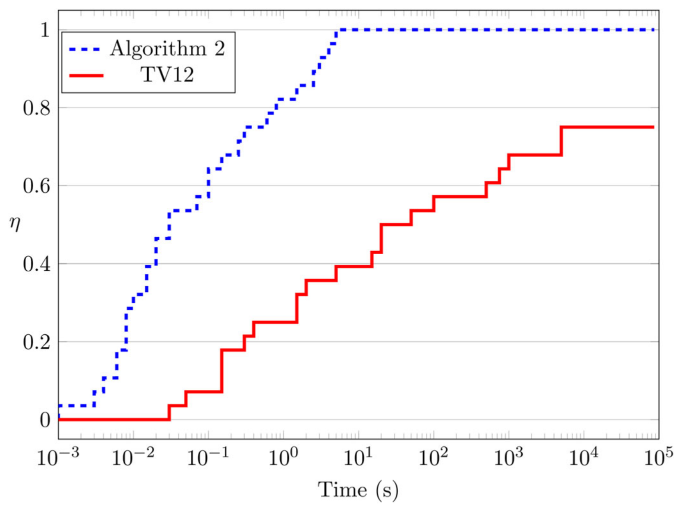


Figure 8. Convex functions: η refers to the fraction of instances solved to optimality within the given time (logarithmic scale).

19 times faster for each instance, and up a maximum of 328,000 times faster (on the same instance as described in Section 5.1). On instances where the state-of-the-art approach is able to find the solution within the time limit, Algorithm 2 is on average more than 6900 times faster.

Figure 8 shows the fraction of instances that are solved within a given time limit for the two approaches. Again, we see that for any time budget (above 0.001 seconds), Algorithm 2 is able to solve a greater fraction of instances, and is able to solve all instances within 5.56 seconds. Within this time budget, the approach by Toriello and Vielma (2012) is only able to solve 36% of the instances. Therefore, it is clear that Algorithm 2 is significantly better for fitting convex PWL functions to continuous functions.

6. Conclusions

We have proposed an efficient algorithm to model continuous functions using continuous PWL regression functions. This algorithm implements the linear time algorithm for PWL function fitting by Imai and Iri (1986) within the adaptive discretization framework presented by Rebennack and Kallrath (2015b). We adapt the framework to account for the slightly modified PWL fitting algorithm, which seeks to minimize the number of breakpoints required to fit the function within a given error tolerance.

We have shown that if the function being modeled is convex, then the presented algorithm is guaranteed to output a convex PWL function. Under certain conditions, we have shown that the runtime of our presented algorithm can be bounded by a function of the domain length of the function being modeled, the desired accuracy, and the number of data points in the discretization. In particular, this gives some insight into the function characteristics that lead to faster runtimes, which is useful when applying MILP techniques to solve complicated non-linear programs.

Experimental results have shown that the presented model is significantly faster than five state-of-the-art models which fit continuous PWL functions, up to several orders of magnitude faster. Future work should implement this algorithm in real-world settings as well as considering higher dimensional approaches.

Acknowledgments

We would like to thank Prof. Dr. Peter Sanders for introducing us to the relevant algorithms. We would also like to thank the editor for the insightful suggestions, and the work of the reviewers.

Funding

This work was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) [Grant 445857709].

Notes on contributors

John Alasdair Warwicker completed a master's degree in mathematics from Loughborough University and a PhD in theoretical computer science from the University of Sheffield. Since 2019, he has been working as a Research Associate at the Institute of Operations Research at Karlsruhe Institute of Technology, in the chair of Stochastic Optimization. His research interests lie on the border of discrete mathematics, computer science, operations research and machine learning. Specific areas of interest include the intersection of optimization and machine learning, and analyses of heuristics, hyper-heuristics and matheuristics.

Steffen Rebennack completed a degree in mathematics at Heidelberg University, and a master's degree in industrial & systems engineering at the University of Florida. He also obtained a PhD in industrial & systems engineering from the University of Florida, before working at the Colorado School of Mines as an assistant professor and an associate professor. Since 2017, he has been working as chair professor for the Stochastic Optimization group in the Institute of Operations Research at Karlsruhe Institute of Technology. His research interests include stochastic and large-scale global optimization problems, with a focus on applications in power systems analysis.

ORCID

John Alasdair Warwicker http://orcid.org/0000-0002-6274-2638

Steffen Rebennack http://orcid.org/0000-0002-8501-2785

References

Aggarwal, A., Booth, H., O'Rourke, J., Suri, S. and Yap, C.K. (1989) Finding minimal convex nested polygons. *Information and Computation*, **83**(1), 98–110.

Charnes, A., Cooper, W.W. and Rhodes, E. (1978) Measuring the efficiency of decision making units. *European Journal of Operational Research*, **2**(6), 429–444.

Chen, D.Z. and Wang, H. (2013) Approximating points by a piecewise linear function. *Algorithmica*, **66**(3), 682–713.

Codsi, J., Ngueveu, S.U. and Gendron, B. (2021) LinA: A faster approach to piecewise linear approximations using corridors and its application to mixed-integer optimization. https://hal.science/hal-03336003/.

Ding, Y. (2019) *Data Science for Wind Energy*. CRC Press, Taylor & Francis Group LLC.

Feijoo, B. and Meyer, R.R. (1988, March) Piecewise-linear approximation methods for nonseparable convex optimization. *Management Science*, **34**(3), 411–419.

Gablonsky, J.M. and Kelley, C.T. (2001) A locally-biased form of the direct algorithm. *Journal of Global Optimization*, **21**(1), 27–37.

Geißler, B., Martin, A., Morsi, A. and Schewe, L. (2012) Using piecewise linear functions for solving MINLPs, in *Mixed Integer Nonlinear Programming*, Volume 154, pp. 287–314. Springer, New York, NY.

Goldberg, N., Kim, Y., Leyffer, S. and Veselka, T.D. (2014) Adaptively refined dynamic program for linear spline regression. *Computational Optimization and Applications*, **58**(3), 523–541.

Goldberg, N., Rebennack, S., Kim, Y., Krasko, V. and Leyffer, S. (2021) MINLP formulations for continuous piecewise linear function fitting. *Computational Optimization and Applications*, **79**(1), 223–233.

Goodrich, M.T. (1995) Efficient piecewise-linear function approximation using the uniform metric. *Discrete & Computational Geometry*, **14**(4), 445–462.

Guan, Y., Pan, K. and Zhou, K. (2018) Polynomial time algorithms and extended formulations for unit commitment problems. *IISE Transactions*, **50**(8), 735–751.

Gunnerud, V. and Foss, B. (2010) Oil production optimization—a piecewise linear model, solved with two decomposition strategies. *Computers & Chemical Engineering*, **34**(11), 1803–1812.

Hakimi, S. and Schmeichel, E. (1991) Fitting polygonal functions to a set of points in the plane. *CVGIP: Graphical Models and Image Processing*, **53**(2), 132–136.

Hastie, T., Tibshirani, R., Friedman, J.H. and Friedman, J.H. (2009) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Volume 2. Springer, New York, NY.

Horst, R., Pardalos, P.M. and Van Thoai, N. (2000) *Introduction to Global Optimization*. Kluwer Academic Publishers, Dodrect, Netherlands.

Hwangbo, H., Johnson, A.L. and Ding, Y. (2018) Spline model for wake effect analysis: Characteristics of a single wake and its impacts on wind turbine power generation. *IISE Transactions*, **50**(2), 112–125.

Imai, H. and Iri, M. (1986) An optimal algorithm for approximating a piecewise linear function. *Journal of Information Processing*, **9**(3), 159–162.

Johnson, S.G. (2021) The NLopt nonlinear-optimization package. http://github.com/stevengj/nlopt.

Kallrath, J. and Rebennack, S. (2014) Computing area-tight piecewise linear overestimators, underestimators and tubes for univariate functions, in *Optimization in Science and Engineering*, pp. 273–292. Springer, New York, NY.

Kazda, K. and Li, X. (2021) Nonconvex multivariate piecewise-linear fitting using the difference-of-convex representation. *Computers & Chemical Engineering*, **150**, 107310.

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

Krasko, V. and Rebennack, S. (2017) Two-stage stochastic mixed-integer nonlinear programming model for post-wildfire debris flow hazard management: Mitigation and emergency evacuation. *European Journal of Operational Research*, **263**(1), 265–282.

Lee, J. and Leyffer, S. (2011) *Mixed Integer Nonlinear Programming*. Springer, New York, NY.

Lohmann, T. and Rebennack, S. (2017) Tailored Benders decomposition for a long-term power expansion model with short-term demand response. *Management Science*, **63**(6), 2027–2048.

Maranas, C.D. and Floudas, C.A. (1994) Global minimum potential energy conformations of small molecules. *Journal of Global Optimization*, **4**(2), 135–170.

McCoy, K., Krasko, V., Santi, P., Kaffine, D. and Rebennack, S. (2016, Aug) Minimizing economic impacts from post-fire debris flows in the Western United States. *Natural Hazards*, **83**(1), 149–176.

Meijering, E. (2002) A chronology of interpolation: From ancient astronomy to modern signal and image processing. *Proceedings of the IEEE*, **90**(3), 319–342.

Muriel, A. and Munshi, F.N. (2004) Capacitated multicommodity network flow problems with piecewise linear concave costs. *IIE Transactions*, **36**(7), 683–696.

Nesterov, Y. and Nemirovskii, A. (1994) *Interior-Point Polynomial Algorithms in Convex Programming*. SIAM, Philadelphia, PA.

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

Rebennack, S. (2016a) Computing tight bounds via piecewise linear functions through the example of circle cutting problems. *Mathematical Methods of Operations Research*, **84**(1), 3–57.

Rebennack, S. (2016b) Combining sampling-based and scenario-based nested Benders decomposition methods: application to stochastic dual dynamic programming. *Mathematical Programming*, **156**(1), 343–389.

Rebennack, S. and Kallrath, J. (2015a, October) Continuous piecewise linear delta-approximations for bivariate and multivariate functions. *Journal of Optimization Theory and Applications*, **167**(1), 102–117.

Rebennack, S. and Kallrath, J. (2015b) Continuous piecewise linear delta-approximations for univariate functions: computing minimal breakpoint systems. *Journal of Optimization Theory and Applications*, **167**(2), 617–643.

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

Toriello, A. and Vielma, J.P. (2012) Fitting piecewise linear continuous functions. *European Journal of Operational Research*, **219**(1), 86–95.

Wagner, A.K., Soumerai, S.B., Zhang, F. and Ross-Degnan, D. (2002) Segmented regression analysis of interrupted time series studies in medication use research. *Journal of Clinical Pharmacy and Therapeutics*, **27**(4), 299–309.

Wang, D.P., Huang, N.F., Chao, H.S. and Lee, R.C.T. (1993) Plane sweep algorithms for the polygonal approximation problems with applications, in *Algorithms and Computation*, pp. 515–522. Springer, Berlin, Heidelberg.

Wang, Y. (2011) *Smoothing Splines: Methods and Applications*. CRC Press, Taylor & Francis Group LLC.

Warwicker, J.A. and Rebennack, S. (2022) A comparison of two mixed-integer linear programs for piecewise linear function fitting. *INFORMS Journal on Computing*, **34**(2), 1042–1047.

Warwicker, J.A. and Rebennack, S. (2023a) A unified framework for bivariate clustering and regression problems via mixed-integer linear programming. *Discrete Applied Mathematics*, **336**, 15–36.

Warwicker, J.A. and Rebennack, S. (2023b) Generating optimal robust continuous piecewise linear regression with outliers through combinatorial Benders decomposition. *IISE Transactions*, **55**(8), 755–767.

Yue, D. and You, F. (2013) Sustainable scheduling of batch processes under economic and environmental criteria with minlp models and algorithms. *Computers & Chemical Engineering*, **54**, 44–59.