Noname manuscript No.
(will be inserted by the editor)

Continuous Piecewise Linear Delta-Approximations for Bivariate and Multivariate Functions

Steffen Rebennack · **Josef Kallrath**

November 12, 2014

Abstract For functions depending on two variables, we automatically construct triangulations subject to the condition that the continuous, piecewise linear approximation, under- or overestimation never deviates more than a given δ-tolerance from the original function over a given domain. This tolerance is ensured by solving subproblems over each triangle to global optimality. The continuous, piecewise linear approximators, under- and overestimators involve shift variables at the vertices of the triangles leading to a small number of triangles while still ensuring continuity over the entire domain. For functions depending on more than two variables, we provide appropriate transformations and substitutions, which allow the use of one- or two-

S. Rebennack (✉)

Division of Economics and Business, Colorado School of Mines, Golden, Co, USA

E-mail: srebenna@mines.edu

J. Kallrath

Department of Astronomy, University of Florida, Gainesville, FL, USA

E-mail: kallrath@astro.ufl.edu

dimensional δ-approximators. We address the problem of error propagation when using these dimensionality reduction routines. We discuss and analyze the trade-off between one-dimensional and two-dimensional approaches and we demonstrate the numerical behavior of our approach on nine bivariate functions for five different δ-tolerances.

Keywords global optimization · nonlinear programming · mixed-integer nonlinear programming · non-convex optimization · error propagation

Mathematics Subject Classification (2000) 90C26

1 Introduction

In this paper, we are interested in approximating nonlinear multivariate functions. The computed approximations should not deviate more than a pre-defined tolerance $\delta > 0$ from the original function. In addition, the approximated functions should be continuous and piecewise linear so they can be represented using mixed-integer linear programming (MILP) techniques. Thus, we are not seeking the computation of a global optimum of a nonconvex function but a function approximation instead.

The motivation to compute these δ-approximations is to approximate a global optimization problem via an MILP problem. The δ-tolerance of the obtained approximation allows for the computation of safe bounds for the original global optimization problem, if the approximation is constructed carefully. Such MILP representations are of particular interest if the global optimization problem is embedded into a, typically much larger, MILP problem. Examples are water supply network optimization and transient technical optimization of gas networks, generalized pooling and integrated water systems problems, gas lifting and well scheduling for enhanced oil recovery, and electrical networks [1,2]. Our own motivation comes from large-scale production planning problems [3,4], and power system optimization problems [5–7].

The incremental approach [2,8] producing Delaunay triangulations is most closely related to our work, but does not involve shift variables at the vertices of the triangles. Our approach can also handle arbitrary, indefinite functions regardless of their curvature. Instead of reviewing a rich body of literature related to piecewise linear approximation, we point the reader to the following papers: Ref. [1] presents explicit, piecewise linear formulations of two- or three-dimensional functions based on simplices; Ref. [9] uses triangulations for quadratically constrained problems; Ref. [10] compares different formulations (one-dimensional, rectangle, triangle) to approximate two-dimensional functions; and Refs. [11,12] are the first to compute optimal breakpoint systems for univariate functions. The recent invention of modified SOS-2 formulations, growing only logarithmically in the number of support areas (breakpoints, triangles, or simplices), relieves somewhat the pressure to seek for a minimum number of support areas involved in the linear approximation of functions [13].

We extend the ideas for univariate functions [11] to higher dimensions. The contributions of this paper are threefold:

1. We develop algorithms to automatically compute triangulations and the construction of continuous, piecewise linear functions over such systems of triangles which approximate nonlinear functions in two variables to δ-accuracy.
2. We classify a rich class of n-dimensional functions which can be transformed into lower dimensional functions along with the error propagation.

3. We demonstrate both the one- and two-dimensional approximation techniques on a test bed of nine bivariate functions.

In the remainder of this paper, we construct δ-accurate piecewise linear approximators, over- and underestimators for bivariate functions in Section 2. Transformations and higher dimensional functions are treated in Section 3. Section 4 provides our numerical results. We conclude in Section 5.

2 Bivariate Functions

In the one-dimensional case, we construct convex linear combinations of breakpoint-limited disjunct intervals covering the region of interest. In the two-dimensional case, we start with rectangular regions and we are seeking support areas which cover the rectangle and which can easily be made larger or smaller reflecting the curvature of the function we want to approximate. While functions depending on two or more variables are treated by equally-sized simplices leading to direct SOS-2 representations in [1], our approach utilizes different-sized triangles to better adjust to the function.

Consider a triangle $\mathscr{T}_1 \subset \mathrm{I\!R}^2$ in the x_1-x_2-plane established by three points (vertices) $P_j = (X_{1j}, X_{2j}) \in \mathrm{I\!R}^2$, $j = 1,2,3$. We assume that at most two of them are colinear, *i.e.*, all three of them never lie on the same line. Each point $p = (x_1, x_2) \in \mathscr{T}_1$ can be represented as a convex combination of these three points, *i.e.*,

$$p = \sum_{j=1}^{3} \lambda_j P_j \qquad \text{with} \qquad \lambda_j \geq 0 \qquad \text{and} \qquad \sum_{j=1}^{3} \lambda_j = 1.$$

Let $f(p) = f(x_1, x_2)$ be a real-valued function in two arguments x_1 and x_2 defined over a rectangle $\mathbb{D} := [X_{1-}, X_{1+}] \times [X_{2-}, X_{2+}] \supset \mathscr{T}_1$. We can construct a linear approximation $\ell(p)$ of f over $\mathscr{T}_1$ by a convex combination of the function values, $f_j = f(P_j) = f(X_{1j}, X_{2j})$, at the points p:

$$\ell(p) = \sum_{j=1}^{3} \lambda_j f_j.$$

2.1 Constructing the Triangulation

Our goal is now to construct a triangulation of $\mathbb{D}$ by a set $\mathscr{T}$ of triangles $\mathscr{T}_t$, with $\bigcup_{\mathscr{T}_t \in \mathscr{T}} \mathscr{T}_t \supseteq \mathbb{D}$, with a minimal number of triangles subject to the constraint

$$\Delta_t := \max_{p \in \mathscr{T}_t} |\ell(p) - f(p)| \leq \delta, \quad \forall \mathscr{T}_t \in \mathscr{T}. \tag{1}$$

The construction of triangulations with a minimum number of triangles remains an open problem. The *direct approach*, in which we proceed similarly as in the one-dimensional case by allowing a fixed number of breakpoints $p_b := (x_b, y_b)$ distributed in plane $\mathbb{D}$, raises severe problems:

1. How to construct non-overlapping triangles (from the p_b) covering the rectangle?
2. How to ensure continuity in the vertices? If we know that two triangles $\mathscr{T}_1$ and $\mathscr{T}_2$ share edge e_{12}, then we have to apply the continuity constraints to the two shared vertices $\mathbf{v}^v_{e_{12}\mathscr{T}_1} = \mathbf{v}^v_{e_{12}\mathscr{T}_2}$ with $v = 1, 2$ belonging to edge e_{12}.

Using a *marching scheme* (*cf.* the moving breakpoint approach [11, Section 4]) leads to complications in constructing an irregular grid of triangles. Therefore, we proceed differently and use a triangle refinement approach.

To begin with, we divide $\mathbb{D}$ into two triangles $\mathscr{T}_1$ and $\mathscr{T}_2$. Then, for a given triangle $\mathscr{T}_t$ with vertices $[v_{t1}, v_{t2}, v_{t3}]$ and fixed shift variables $[s_{t1}, s_{t2}, s_{t3}]$, we solve the following (potentially nonconvex) nonlinear programming (NLP) problem:

$$\Delta_t := \max \quad |\ell(p_t) - f(p_t)| \tag{2}$$

$$\text{s.t.} \quad p_t = \sum_{j=1}^{3} \lambda_j v_{tj} \tag{3}$$

$$\sum_{j=1}^{3} \lambda_j = 1 \tag{4}$$

$$\ell(p_t) := \sum_{j=1}^{3} \lambda_j \phi\left(v_{ti}\right) \tag{5}$$

$$\lambda_j \in [0,1], \quad p_t \in \mathscr{T}_t, \quad j = 1,2,3, \tag{6}$$

with $\phi(\cdot)$ defined as

$$\phi(v_{tj}) = f(v_{tj}) + s_{tj}, \quad j = 1,2,3 \tag{7}$$

for vertices v_{tj} contained in triangle $\mathscr{T}_t$; (7) is the analogon to equation (3) in [11].

If $\Delta_t \leq \frac{\delta}{2}$, then we keep the triangle $\mathscr{T}_t$ along with the shift variables s_{tj}; *i.e.*, we add $\mathscr{T}_t$ to $\mathscr{T}$. Otherwise, we try a different value for the shift variables s_{tj} as follows

$$s_{tjd} := \left(\frac{2d}{D+1} - 1\right)\frac{\delta}{2} \tag{8}$$

for some $D \in \mathbb{N}$ and $d \in \{1,\ldots,D\}$. Care has to be taken to identify shift variables which have been fixed before. Thus, for each triangle $\mathscr{T}_t$, we solve between 1 and D^3 NLP problems (2)-(6).

If none of the shift variable combinations satisfy $\Delta_t \leq \frac{\delta}{2}$, we use a so-called *sub-division rule* to sub-divide triangle $\mathscr{T}_t$ into smaller triangles. Given is $\mathscr{T}_t$ with vertices $[v_{t1}, v_{t2}, v_{t3}]$ and their three center points p_{ti} of each side of the triangle, *i.e.*, $p_{t1} := (v_{t1} + v_{t2})/2$, $p_{t2} := (v_{t2} + v_{t3})/2$ and $p_{t3} := (v_{t1} + v_{t3})/2$. Now, we divide $\mathscr{T}_t$ into four triangles as follows (see Figure 1):

$$\tilde{\mathscr{T}}_1 = [v_{t1}, p_{t1}, p_{t3}],\ \tilde{\mathscr{T}}_2 = [v_{t2}, p_{t1}, p_{t2}],\ \tilde{\mathscr{T}}_3 = [v_{t3}, p_{t2}, p_{t3}],\ \tilde{\mathscr{T}}_4 = [p_{t1}, p_{t2}, p_{t3}]. \tag{9}$$

Once all triangles in $\mathscr{T}$ yield a piecewise linear $\frac{\delta}{2}$-approximator for f, we need to remove potential discontinuities at the boundary of the triangles. The idea is to sub-divide such $\mathscr{T}_t$, which contain vertices of other triangles at the boundary of $\mathscr{T}_t$, into smaller triangles, without introducing any new vertices. The piecewise linear functions constructed on the smaller triangles deviate then at most δ from f; the shift variables at the vertices remain untouched. One simple rule of sub-division is to iteratively connect one of the vertices on the boundary of $\mathscr{T}_t$ with another vertex of the boundary or with a vertex of $\mathscr{T}_t$, but not with a vertex at the same side of the triangle. This idea is illustrated in Figure 1 as well.

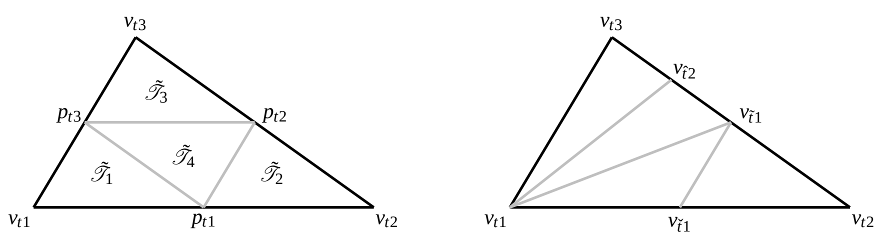


Fig. 1: Sub-division rule (left) and removal of discontinuities (right).

The described procedure is summarized in Algorithm 2.1. Set $\overline{\mathscr{T}}$ contains all triangles which have not been checked; set $\mathbb{S}$ is a set of ordered pairs $\{s_{tj}, v_{tj}\}$ which assigns each vertex v_{tj} of a triangle a shift variable s_{tj}. Using Algorithm 2.1, we obtain a triangulation for the domain $\mathbb{D}$, leading to a continuous, piecewise linear approximation with the desired approximation δ. We make the following observations:

(1) If continuity for an approximator is not required, then Algorithm 2.1 can be modified as follows: steps 29-33 can be removed and the criteria for Δ_t in steps 17 and 19 can be relaxed to $\Delta_t \leq \delta$.

(2) Computationally, it is an advantage to terminate loop 13-17 if the error $\Delta_t > \delta$.

(3) Algorithm 2.1 works for any compact domain which can be partitioned into a (finite) set of triangles. Step 6 of the algorithm needs to be adjusted accordingly.

(4) The only requirement for any sub-division rule is that after finitely many iterations, triangles of arbitrarily small side lengths can be computed. Thus, we suggest an alternative sub-division rule: using the point p_t^* of maximal deviation of ℓ and f over $\mathscr{T}_t$, obtained by solving (2)-(6), we divide $\mathscr{T}_t$ with vertices $[v_{t1}, v_{t2}, v_{t3}]$ into three triangles as follows (we found fixing the free shift variables to zero for computing point p_t^* as computationally most efficient):

$$[v_{t1}, v_{t2}, p_t^*], \quad [v_{t2}, v_{t3}, p_t^*], \quad [v_{t3}, v_{t1}, p_t^*]. \tag{10}$$

The case that p_t^* happens to lie on any of the three sides of the triangle $\mathscr{T}_t$ needs special care. First, we remove the triangle with zero area. Second, we need to ensure that the calculated function approximation is continuous at this particular side of the triangle $\mathscr{T}_t$. The continuity can be ensured by a simple trick: divide the one neighboring triangle which contains point p_t^* into three triangles using (10). This preserves the approximation tolerance and a deviation of δ instead of $\frac{\delta}{2}$ up front can be allowed. This sub-division rule has one drawback: one could imagine a case where the computed triangles get arbitrarily small in area, but not in the side length. To avoid this issue, one could sub-divide the triangles into smaller triangles using the subdivision rule used by Algorithm 2.1 every fixed iteration count. The subdivision rule using a point of maximal deviation has the advantage over the other sub-division rule presented above that the number of triangles is expected to increase slower.

Algorithm 2.1 Heuristic to Compute Triangulation and δ-Approximator

// **INPUT:** Continuous function f, scalar $\delta > 0$, and shift variable discretization size $D \in \mathbb{N}$

// **OUTPUT:** Set of triangles $\mathscr{T}$ and shift values $\mathbb{S}$

// Initialize

$\mathscr{T} := \emptyset$, $\mathbb{S} := \emptyset$

// rectangle $\mathbb{D} = [X_{1-}, X_{1+}] \times [X_{2-}, X_{2+}]$ is divided into two triangles

$\overline{\mathscr{T}} := \{[(X_{1-}, X_{2-}), (X_{1+}, X_{2-}), (X_{1-}, X_{2+})], [(X_{1-}, X_{2+}), (X_{1+}, X_{2-}), (X_{1+}, X_{2+})]\}$

// **Divide triangles until all triangles satisfy the $\frac{\delta}{2}$-criteria**

repeat

    // obtain and remove triangulation

    choose $\mathscr{T}_t \in \overline{\mathscr{T}}$ with vertices $[v_{t1}, v_{t2}, v_{t3}]$ and update $\overline{\mathscr{T}} \leftarrow \overline{\mathscr{T}} \setminus \mathscr{T}_t$

    // obtain fixed variables

    if $\{s_{tj}, v_{tj}\} \in \mathbb{S}$, then obtain s_{tj} and fix this variable for formulation (2)-(6); for all $j = 1, 2, 3$

    repeat

        for all vertices v_{tj} with un-fixed shift s_{tj}, obtain a discretized value via (8)

        // optimize

        solve (2)-(6) with fixed shift variables to obtain Δ_t

    until $\Delta_t \leq \frac{\delta}{2}$ or all discretize values for the shift variables have been checked

    // check $\frac{\delta}{2}$-criteria

    if $\Delta_t \leq \frac{\delta}{2}$ **then**

        // update the set of triangles...

        $\mathscr{T} \leftarrow \mathscr{T} \cup \{\mathscr{T}_t\}$

        // ...and the shift variables

        $\mathbb{S} \leftarrow \mathbb{S} \cup \{\{s_{tj}, v_{tj}\}, j = 1, 2, 3\}$

    else

        construct new triangles via (9) and add them to set $\overline{\mathscr{T}}$

    end if

until $\overline{\mathscr{T}} = \emptyset$

// **Remove discontinuities**

for all $\mathscr{T}_t \in \mathscr{T}$ **do**

    if $\exists \mathscr{T}_{\tilde{t}} \in \mathscr{T}$ where $v_{\tilde{t}j}$ lies on one of the sides of $\mathscr{T}_t$ for some j **then**

        sub-divide triangle $\mathscr{T}_t$

    end if

end for

2.2 Deriving Good δ-Underestimators and δ-Overestimators

The easiest way to construct δ-under- and overestimators $\ell_{\pm}(x)$ in the bivariate case is to exploit the interpolation-based approximation $\ell(x)$ of $f(x)$ with accuracy $\frac{\delta}{2}$ by setting $\ell_{\pm}(x) := \ell(x) \pm \frac{\delta}{2}$. However, if the $\frac{\delta}{2}$-approximator for f does not possess a minimal number of triangles, then the computed δ-under- and overestimators are not minimal in the number of triangles used in the triangulation [11].

Our specific calculation of δ-underestimators or δ-overestimators follows very closely the idea of δ-approximators. We focus our discussions on δ-underestimators. Instead of solving (1), we use for $\ell_{-}(p)|_{p\in\mathscr{T}_t}$

$$\begin{aligned} \Delta_t^{+} := \max_{p\in\mathscr{T}_t} \ & \left(f(p) - \ell_{-}(p)\right) \leq \delta \\ \text{s.t.} \ & \ell_{-}(p) \leq f(p), \quad \forall p \in \mathscr{T}_t. \end{aligned} \tag{11}$$

We discretize the continuum conditions (11), for a given triangle, into I grid points p_{ti}. This is achieved by choosing λ_{1i} and λ_{2i} with $i \in \mathbb{I} := \{1,\dots,I\}$, yielding to $\lambda_{3i} = 1 - \lambda_{1i} - \lambda_{2i}$. This generates a system of grid points p_{ti},

$$p_{ti} = \sum_{j=1}^{3} \lambda_{ji} v_{tj}, \quad \forall \mathscr{T}_t \in \mathscr{T}, \quad \forall i \in \mathbb{I}. \tag{12}$$

Let $\mathscr{T}_t$ be a triangle with vertices $[v_{t1}, v_{t2}, v_{t3}]$. The NLP (2)-(6) is replaced by:

$$\Delta_t^{D+} := \min \ \eta \tag{13}$$

$$\text{s.t.} \quad \eta \geq f(p_{ti}) - \ell_{-}(p_{ti}), \quad \forall i \in \mathbb{I} \tag{14}$$

$$\ell_{-}(p_{ti}) \leq f(p_{ti}), \quad \forall i \in \mathbb{I} \tag{15}$$

$$\ell_{-}(p_{ti}) := \textstyle\sum_{j=1}^{3} \lambda_{ji} \phi\left(v_{tj}\right), \quad \forall i \in \mathbb{I} \tag{16}$$

$$\eta \geq 0, \quad s_{tj} \in \left[-\tfrac{1}{3}\delta, 0\right], \quad j = 1,2,3, \tag{17}$$

with $\phi(\cdot)$ as given by (7); the λ_{ji} are fixed and obtained by (12). Notice that the shift variables are not discretized, in contrast to the approach described in Section 2.1.

If $\Delta_t^{D+} > \frac{1}{3}\delta$, one can proceed with a sub-division rule as in the case for δ-approximators to further divide the triangle $\mathscr{T}_t$. However, if $\Delta_t^{D+} \leq \frac{1}{3}\delta$, we need to ensure that the derived ℓ_- is indeed an underestimator for f. Therefore, we check

$$z_{\pm}^{\max *} := \max_{p \in \mathscr{T}_t} \big(f(p) - \ell_-(p)\big) \leq \frac{1}{3}\delta \quad \text{and} \quad z_{\pm}^{\min *} := \min_{p \in \mathscr{T}_t} \big(f(p) - \ell_-(p)\big) \geq 0.$$

If both conditions are met, then the computed ℓ is an underestimator for f on triangle $\mathscr{T}_t$. Thus, we can keep $\mathscr{T}_t$ as well as the shift variables s_{ti}^*. Otherwise, we have to divide the triangle $\mathscr{T}_t$ further. To ensure continuity at the boundary of the triangles $\mathscr{T}_t$, we proceed as in the case for δ-approximators (steps 29-33 of Algorithm 2.1). Shifting the obtained approximator by $-\frac{1}{3}\delta$ ensures a piecewise linear, continuous δ-underestimator for f.

3 Multivariate Functions and their Linear Approximations

The ideas and concepts developed for univariate and bivariate functions can be extended to approximate functions of higher dimensions by piecewise linear constructs. However, the number of support areas, usually simplices, increases exponentially [2].

An open question is whether it worthwhile to exploit special properties, *e.g.*, separability, of the functions to reduce the dimensionality, or is it more efficient to approximate the function directly in its dimensionality. Intuitively, one might argue that the reduction of dimensionality pays out, but this is not obvious and may depend both on the problem and on the branching strategy used by the selected MILP solver.

Transformations for special nonlinear expressions enable us to utilize one- and two-dimensional techniques to construct δ-approximators for n-dimensional functions. We summarize four function types and their transformation tricks in Table 1.

I: Separable functions. We apply the one-dimensional δ-approximators to each of the n one-dimensional functions $f_i(x_i)$ separately. The obtained approximation error for $f(x)$ is then the sum of the individual errors δ_i for each expression $f_i(x_i)$.

II: Positive function products. For products of functions, we require that all functions are positive. Otherwise, assume without loss of generality that exactly one function, $f_j(x_j)$, is non-positive. As f_j is continuous on the compactum $[X_{j-},X_{j+}]$, f_j is bounded. Therefore, $L_j := \min_{x\in[X_{j-},X_{j+}]} f_j(x)$ is finite. Now, substitute

$$\prod_{i=1}^{n} f_i(x_i) = (f(x_j)+D_j)\prod_{i=1,i\neq j}^{n} f_i(x_i) - D_j \prod_{i=1,i\neq j}^{n} f_i(x_i)$$

with $D_j = L_j + k$ and some positive number k, *e.g.*, $k = 1$. As

$$(f(x_j)+D_j)\prod_{i=1,i\neq j}^{n} f_i(x_i) := \tilde{f}(x) > 0 \quad \text{and} \quad D_j \prod_{i=1,i\neq j}^{n} f_i(x_i) := \overline{f}(x) > 0,$$

we can apply the transformation to both functions $\tilde{f}(x)$ and $\overline{f}(x)$ separately. Note that the error obtained by the transformation of the product of positive functions depends on $f(x)$. If $\ln(f(x))$ has error $\delta = \sum_{i=1}\delta_i$, then $\Delta f(x)$ follows from

$$f(x) + \Delta f(x) = e^{\ln(f(x))+\delta} = f(x)\cdot e^{\delta}$$

and $\Delta f(x) = f(x)\left(e^{\delta} - 1\right)$, which for small values of δ reduces to $\Delta f(x) \approx f(x)\cdot\delta$. Thus, we loose the separation property between the x_i variables regarding the discretization error, *i.e.*, although the discretization errors of x_i and x_j are separated for $i \neq j$, the discretization error of the product $f_i(x_i)\cdot f_j(x_j)$

depends on both x_i and x_j (as well as on $f_i(x_i)$ and $f_j(x_j)$). However, if "good" bounds on $f(x)$ are available, then this approach may still be computationally feasible, *e.g.*, $0 < f(x) \leq 1$ is desirable as this guarantees an approximation error for $f(x)$ of at most $e^\delta - 1$, or δ for small values of δ, respectively.

III: Exponentials. Chains of exponentials $f_1(x)^{f_2(x)}$ for n-dimensional functions $f_1(x)$ and $f_2(x)$ with $x \in \mathbb{R}^n$ require some care related to the arguments. The transformation works only for $f_1(x) > 0$ and $f_2(x) > 1$.

IV: Substitutions. Complicated terms with more variables appearing as arguments of functions can always be replaced by substitutions. Let $f(x) = f_1\big(f_2(x)\big)$ be a nested function with $x \in \mathbb{D} \subseteq \mathbb{R}^n$. Define $u := f_2(x)$ and $f_2 : \mathbb{D} \to \tilde{\mathbb{D}}$. If function $f_2(x)$ is approximated with an absolute error of δ_2, then a maximal error of $\gamma(\delta_2)$ is derived for f_1 (if f_1 is represented exactly). The function $\gamma(\delta_2)$ is the maximal deviation of function f_1 in its domain over a small variation with magnitude δ_2. Note that $\gamma(\delta_2)$ can be overestimated using the derivative of f_1 as follows:

$$\gamma(\delta_2) \leq f'_* \delta_2, \tag{18}$$

where $f'_* = \max_{u \in \tilde{\mathbb{D}}} \frac{\partial f_1(u)}{\partial u}$, if f is differentiable in a domain containing $\tilde{\mathbb{D}}$. The errors of an approximation of f_1 and $\gamma(\delta_2)$ are then additive for function $f(x)$.

4 Computational Results

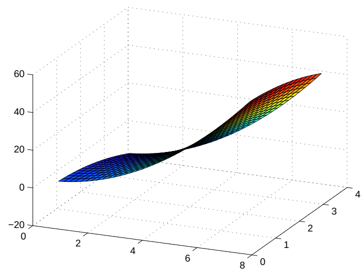
We use the modeling language GAMS (v. 23.6), employing the global optimization solver LindoGlobal and run the computational tests on a standard desktop computer as described in [11].

Table 1: Transformations for n-dimensional functions; $f_i(x_i) : [X_{i-}, X_{i+}] \subset \mathbb{R} \to \mathbb{R}$ for all $i = 1, \ldots, n$ and $f(x) : [X_-, X_+] \subset \mathbb{R}^n \to \mathbb{R}$; all functions are continuous.

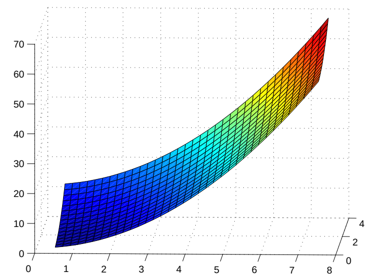
	Function $f(x)$	**Transformation**	**Approx. Error for** $f(x)$	**Comment**
I	$\sum_{i=1}^n \pm f_i(x_i)$	treat each term $f_i(x_i)$ individually	$\sum_{i=1}^n \delta_i$	δ_i is approx. error of $f_i(x_i)$
II	$\prod_{i=1}^n f_i(x_i)$	$\ln(f(x)) = \sum_{i=1}^n \ln(f_i(x_i))$	$f(x)\left(e^{\sum_{i=1}^n \delta_i} - 1\right)$	$f_i(x_i) > 0$ for all i; δ_i is approx. error of $\ln(f_i(x_i))$
III	$f_1(x)^{f_2(x)}$	$\ln(\ln(f(x))) = \ln(f_1(x)) + \ln(\ln(f_2(x)))$	$f(x)\left(e^{e^{(\delta_1+\delta_2)}} - 1\right)$	$f_1(x), f_2(x) > 1$; δ_1 is approx. error of $\ln(f_1(x))$ and δ_2 is approx. error of $\ln(\ln(f_2(x)))$
IV	$f_1\left(f_2(x)\right)$	$f_1(u)$ and $f_2(x)$	$\delta_1 + \gamma(\delta_2)$	δ_1 is approx. error of $f(u)$ and δ_2 is approx. error of $f_2(x)$
$\gamma(\delta_2) := \max\limits_{x \in \mathbb{D},\, x-\delta_2 \leq y \leq x+\delta_2} \lvert f_1(x) - f_1(y) \rvert$				

The nine different functions tested are summarized in Table 2. The columns X_- and X_+ define the lower and upper bounds, respectively, on both decision variables x_1 and x_2. The functions are plotted in Figure 2.

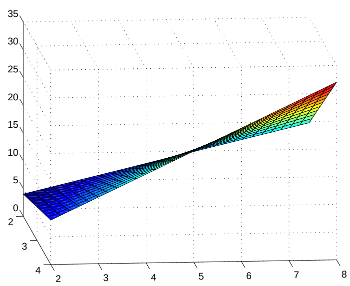
Table 3 summarizes the transformations applied towards functions 1 though 7 of Table 2. The column "Type" indicates which type of transformation, as defined in Table 1, has been applied. For all computations, we choose both δ_1 and δ_2 to be equal. For type I transformations, this leads to $\delta_1 = \delta_2 = \frac{\delta}{2}$ (*cf.* Table 1 column



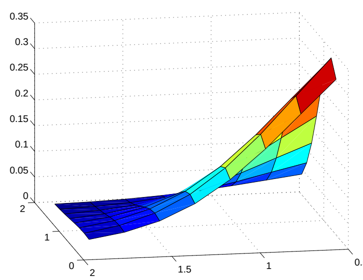
(a) Function 1



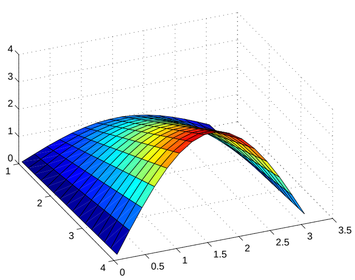
(b) Function 2



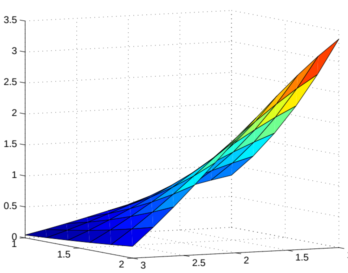
(c) Function 3



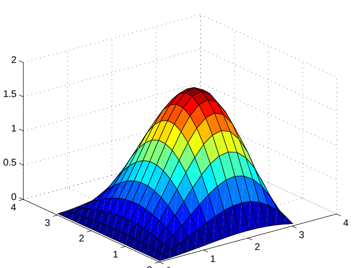
(d) Function 4



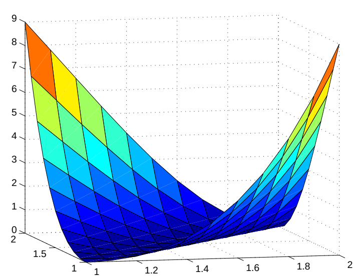
(e) Function 5



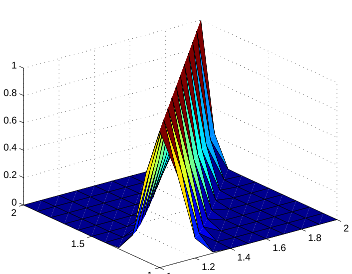
(f) Function 6



(g) Function 7



(h) Function 8



(i) Function 9

Table 2: Two-dimensional functions tested.

#	$f(x)$	X_-	X_+	Comment
1	$x_1^2 - x_2^2$	[0.5,0.5]	[7.5,3.5]	D.C. function [14]
2	$x_1^2 + x_2^2$	[0.5,0.5]	[7.5,3.5]	convex function
3	$x_1 \cdot x_2$	[2.0,2.0]	[8.0,4.0]	–
4	$x_1 \cdot \exp(-x_1^2 - x_2^2)$	[0.5,0.5]	[2.0,2.0]	maximum function value: ≈ 0.334
5	$x_1 \sin(x_2)$	[1.0,0.05]	[4.0,3.1]	concave function on domain
6	$\frac{\sin(x_1)}{x_1} x_2^2$	[1.0,1.0]	[3.0,2.0]	–
7	$x_1 \sin(x_1) \sin(x_2)$	[0.05,0.05]	[3.1,3.1]	–
8	$\left(x_1^2 - x_2^2\right)^2$	[1.0,2.0]	[1.0,2.0]	–
9	$\exp\left(-10(x_1^2 - x_2^2)^2\right)$	[1.0,1.0]	[2.0,2.0]	steep peak at $x_1 = x_2$

"approx. error"). The individual approximation errors for type II transformations are

$$\delta_1 = \delta_2 := \frac{1}{2} \ln\left(\frac{\delta}{m^*} + 1\right), \qquad \text{with} \qquad m^* := \max_{x \in [X_-, X_+]} |f(x)|.$$

If the exact value of m^* is missing, then we use an overestimator m^+ for m^*, *i.e.*, $m^+ \geq m^*$. The values for m^+ and/or m^* are given in column "Comment" of Table 3.

For functions 8 and 9 of Table 2, we apply the substitution rule, *i.e.*, case IV of Table 1. The resulting one-dimensional function $f_1(u) : \tilde{\mathbb{D}} \to \mathrm{I\!R}$ is stated along with the two-dimensional, nested function $f_2(x_1, x_2)$; the domain of f_2 is stated in Table 2. The choice for the approximation errors δ_1 and δ_2 for f_1 and f_2, respectively, are stated in the last two columns of the table. The two-dimensional function $f_2(x_1, x_2) = x_1^2 - x_2^2$ can be approximated by applying a type I transformation, choosing an individual approximation error of $\frac{\delta_2}{2}$, for instance. In order to compute δ_2 for function 9, we have used the maximal derivative of $2\sqrt{\frac{5}{e}}$ in order to overestimate $\gamma(\delta_2)$, see (18).

Table 3: Transformations to univariate functions for functions 1 to 7 of Table 2.

#	$f_1(x_1)$	$f_2(x_2)$	Type	Comment
1	x_1^2	$-x_2^2$	I	–
2	x_1^2	x_2^2	I	–
3	$\ln(x_1)$	$\ln(x_2)$	II	$m^+ = m^* = 32$
4	$\ln(x_1) - x_1^2$	$-x_2^2$	II	$m^+ = 0.3341$
5	$\ln(x_1)$	$\ln\big(\sin(x_2)\big)$	II	$m^+ = m^* = 4$
6	$\ln\big(\sin(x_1)\big) - \ln(x_1)$	$2\ln(x_2)$	II	$m^+ = 3.37$
7	$\ln\big(\sin(x_1)\big) + \ln(x_1)$	$\ln\big(\sin(x_2)\big)$	II	$m^+ = 1.82$

For our computations via Algorithm 2.1, we use the maximal deviation point in each triangle as the sub-division rule, as described in Section 2.1. Empirically, we observed that a discretization of the shift variables of $-\frac{\delta}{2}, -\frac{\delta}{4}, 0, \frac{\delta}{4}, \frac{\delta}{2}$ is a good trade-off between computational time and number of triangles computed.

Table 4: Substitutions for function 8 and 9 of Table 2.

#	$f_1(u)$	$\tilde{\mathbb{D}}$	$f_2(x_1,x_2)$	δ_1	δ_2
8	u^2	[0,4]	$x_1^2 - x_2^2$	$\frac{\delta}{2}$	$\delta_2 = 4 - \sqrt{4 + \frac{\delta}{2}}$
9	$\exp(-10u^2)$	[0.4]	$x_1^2 - x_2^2$	$\frac{\delta}{2}$	$\delta_2 = \frac{\delta}{4}\sqrt{\frac{e}{5}}$

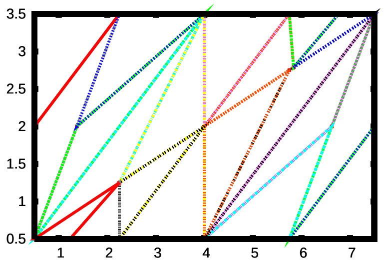
The computational results for functions 1 through 9 of Table 2 are summarized in Table 5. For each function $f(x)$, we choose five consecutive values for the approximation error δ among the set $\{1.50, 1.00, 0.50, 0.25, 0.10, 0.05, 0.03, 0.01, 0.001\}$, dependent on the scaling of the function. The results for the 2-D approach are computed by Algorithm 2.1. The column $|\mathscr{T}|$ states the number of triangles used. For the 1-D approach, we use the Algorithm 4.1 in [11]. The approximation error δ_i is applied to both functions $f_1(x_1)$ and $f_2(x_2)$, except for functions 8 and 9. B_1 and B_2 are the computed number of breakpoints for function f_1 and f_2, respectively. Column "$|R|$" reports on the number of rectangles resulting from the obtained breakpoint systems; again, functions 8 and 9 are different. There, we report the number of rectangular prisms leading to feasible values for x_1, x_2 and u. For both 1-D and 2-D, "dev." summarizes the maximal deviation of the obtained piecewise linear, continuous function over the triangulation compared to the approximated function $f(x)$. These values have been obtained by solving a series of global optimization problems after the approximations have been computed (the computational times are not reported). The columns "CPU (sec.)" provide the computational times in seconds.

From the numerical results presented in Table 5, we derive two main conclusions: (1) At a first glance, the advantage of applying approximations schemes seems not as striking as expected because separate one-dimensional piecewise linear approximations seem to require less breakpoints (particularly for functions which separate well, *e.g.*, functions 1 and 2). However, whether this is really an advantage depends on the behavior of the MILP solver when both the triangles and the one-dimensional breakpoint systems are implemented. (2) A limitation of one-dimensional separable approaches is the numerical accuracy required. For instance, the numerical errors when using logarithmic separations approaches involve the function values themselves. This may request very small errors of the order of 0.001 or smaller.

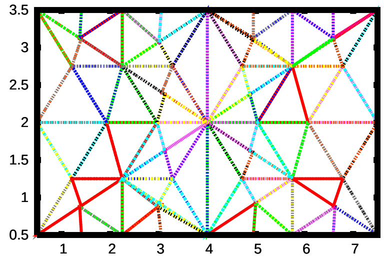
Triangulations calculated by Algorithm 2.1 are shown in Figure 3 for different values of δ, before the final refinement, to ensure continuity, has been applied.

Table 5: Computation results for triangulations and one-dimensional transformations.

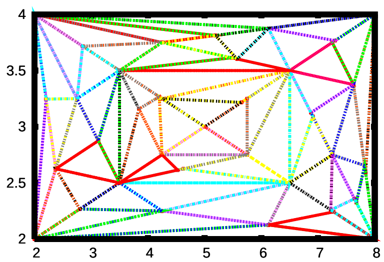
#	δ	2-D $\lvert\mathscr{T}\rvert$	2-D dev.	2-D CPU (sec.)	1-D δ_i	B_1	B_2	$\lvert R\rvert$	1-D dev.	1-D CPU (sec.)
1	1.50	16	1.4844	30.8	0.7500	4	3	6	1.4764	0.5
	1.00	20	0.9844	84.4	0.5000	5	3	8	0.9967	0.4
	0.50	48	0.5000	150.4	0.2500	6	4	15	0.4990	0.5
	0.25	80	0.2461	272.6	0.1250	9	5	32	0.2499	1.2
	0.10	224	0.1000	380.6	0.0500	13	6	60	0.1000	1.6
2	1.50	24	1.5000	26.8	0.7500	4	3	6	1.5000	0.5
	1.00	28	0.9712	7.4	0.5000	5	3	8	1.0000	0.4
	0.50	84	0.4554	38.0	0.2500	6	4	15	0.5000	0.5
	0.25	121	0.2428	35.8	0.1250	9	5	32	0.2500	1.2
	0.10	351	0.0949	171.7	0.0500	13	6	60	0.1000	1.6
3	1.00	4	0.7500	0.8	0.0153	4	3	6	0.5446	0.4
	0.50	12	0.4444	72.4	0.0077	5	3	8	0.4966	0.5
	0.25	20	0.2344	4.7	0.0038	7	4	18	0.1697	0.6
	0.10	59	0.0968	59.3	0.0015	10	6	45	0.0889	1.0
	0.05	94	0.0490	45.3	0.0007	15	8	98	0.0413	1.3
4	0.10	2	0.0976	0.3	0.1309	3	3	4	0.0908	0.4
	0.05	6	0.0346	18.7	0.0697	4	4	9	0.0454	0.6
	0.03	10	0.0288	12.7	0.0429	5	4	12	0.0279	0.7
	0.01	31	0.0097	54.6	0.0147	7	6	30	0.0100	0.9
	0.001	350	0.0010	652.6	0.0014	19	16	270	0.0009	2.7
5	1.00	5	0.9542	1.0	0.1115	3	7	12	0.8911	0.6
	0.50	8	0.4803	13.1	0.0588	3	9	16	0.3219	1.2
	0.25	16	0.2442	30.0	0.0303	3	13	24	0.2441	1.3
	0.10	44	0.0975	74.6	0.0123	5	19	72	0.0924	1.7
	0.05	85	0.0483	141.9	0.0062	6	26	125	0.0434	2.4
6	0.50	2	0.4461	1.8	0.0691	4	2	3	0.4988	0.5
	0.25	4	0.2104	1.0	0.0357	6	3	10	0.1813	0.6
	0.10	9	0.0976	25.8	0.0146	8	4	28	0.0971	1.4
	0.05	23	0.0451	14.4	0.0073	10	4	27	0.0495	1.0
	0.03	40	0.0297	161.4	0.0044	13	5	48	0.0228	2.6
7	1.00	6	0.4885	7.7	0.2189	5	6	20	0.9764	1.2
	0.50	6	0.4885	1.3	0.1213	7	8	42	0.4280	1.4
	0.25	21	0.2351	30.8	0.0643	9	11	80	0.2089	1.3
	0.10	96	0.0980	73.0	0.0267	13	15	168	0.0944	2.6
	0.05	274	0.0498	305.5	0.0135	18	21	340	0.0497	3.4
8	1.00	6	0.8204	22.8	0.0310	4	4	6	0.6117	0.5
	0.50	9	0.4340	15.6	0.0155	4	4	7	0.2852	0.5
	0.25	12	0.2439	22.9	0.0077	6	6	18	0.1575	0.7
	0.10	40	0.0959	202.8	0.0031	8	8	40	0.7312	0.8
	0.05	87	0.0500	174.1	0.0015	11	11	83	0.0384	1.0
9	1.00	2	1.0000	0.8	0.1100	2	3	3	0.5000	0.3
	0.50	4	0.4909	66.6	0.0460	3	3	4	0.2507	0.3
	0.25	6	0.1744	4.4	0.0230	3	4	7	0.1359	0.4
	0.10	84	0.0945	231.5	0.0092	5	5	18	0.0737	0.6
	0.05	86	0.0480	57.8	0.0046	5	7	34	0.0412	0.7



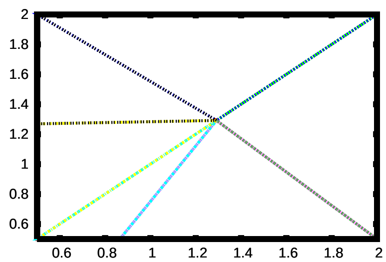
(a) Func. 1: $x_1^2 - x_2^2$, $\delta = 1.00$



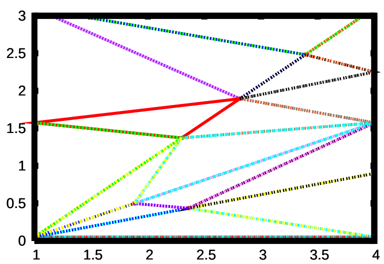
(b) Func. 2: $x_1^2 + x_2^2$, $\delta = 0.50$



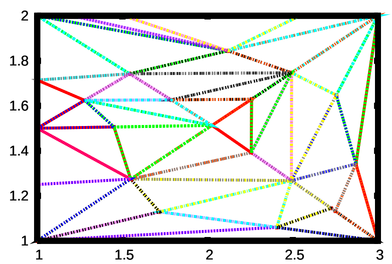
(c) Func. 3: $x_1 \cdot x_2$, $\delta = 0.10$



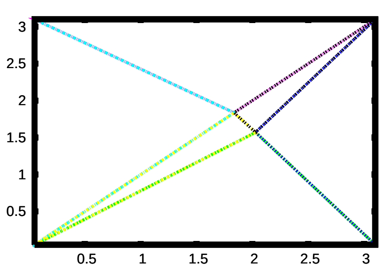
(d) Func. 4: $x_1 \cdot \exp(-x_1^2 - x_2^2)$, $\delta = 0.05$



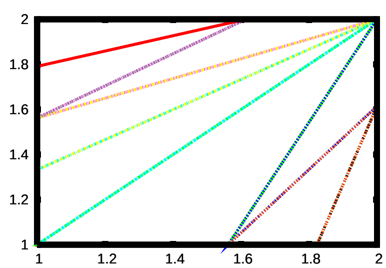
(e) Func. 5: $x_1 \sin(x_2)$, $\delta = 0.25$



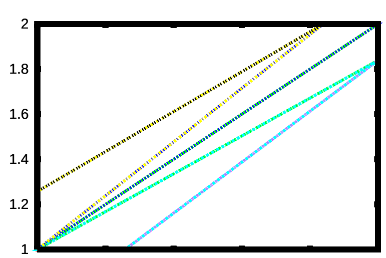
(f) Func. 6: $\frac{\sin(x_1)}{x_1} x_2^2$, $\delta = 0.03$



(g) Func. 7: $x_1 \sin(x_1) \sin(x_2)$, $\delta = 0.50$



(h) Func. 8: $\left(x_1^2 - x_2^2\right)^2$, $\delta = 0.50$



(i) Func. 9: $\exp\left(-10(x_1^2 - x_2^2)^2\right)$, $\delta = 0.25$

5 Conclusions

For bivariate nonlinear functions, we automatically generate triangulations for continuous piecewise linear approximations as well as over- and underestimators satisfying a specified δ-accuracy. The methods we have developed require the solution of nonconvex mathematical programming problems to global optimality. We allow the deviation of the computed interpolation, associated with the triangulation, at the vertices of the triangles through shift variables in an effort to reduce the number of required triangles.

We presented four different dimension reduction techniques allowing to utilize approaches approximating lower dimensional functions. The computational results for the one-dimensional approaches applied to two-dimensional problems are quite promising in that the piecewise linear approximations are computed fast, requiring very few support areas.

There are several promising directions for future research. We have mentioned two open problems in the paper. In addition, when using the proposed dimension reduction transformations, we face the problem of choosing the individual approximation errors δ_i. For our computations, we have chosen them equally. An optimal selection of δ_i's leading to a piecewise linear function requiring the least number of breakpoints for a given accuracy δ is an interesting problem in this context.

References

1. Misener, R., Floudas, C.A.: Piecewise-linear approximations of multidimensional functions. Journal of Optimization Theory and Applications **145**, 120–147 (2010)

2. Geißler, B., Martin, A., Morsi, A., Schewe, L.: Using piecewise linear functions for solving MINLPs. In: J. Lee, S. Leyffer (eds.) Mixed Integer Nonlinear Programming, The IMA Volumes in Mathematics and its Applications, vol. 154, pp. 287–314. Springer (2012)
3. Timpe, C., Kallrath, J.: Optimal planning in large multi-site production networks. European Journal of Operational Research **126(2)**, 422–435 (2000)
4. Kallrath, J.: Solving planning and design problems in the process industry using mixed integer and global optimization. Annals of Operations Research **140**, 339–373 (2005)
5. Zheng, Q.P., Rebennack, S., Iliadis, N.A., Pardalos, P.M.: Optimization models in the natural gas industry. In: S. Rebennack, P.M. Pardalos, M.V. Pereira, N.A. Iliadis (eds.) Handbook of Power Systems I, chap. 6, pp. 121–148. Springer (2010)
6. Frank, S., Steponavice, I., Rebennack, S.: Optimal power flow: a bibliographic survey I. Energy Systems **3**(3), 221–258 (2012)
7. Frank, S., Steponavice, I., Rebennack, S.: Optimal power flow: a bibliographic survey II. Energy Systems **3**(3), 259–289 (2012)
8. Geißler, B.: Towards globally optimal solutions for MINLPs by discretization techniques with applications in gas network optimization. Dissertation, Universität Erlangen-Nürnberg, (2011)
9. Linderoth, J.: A simplicial branch-and-bound algorithm for solving quadratically constrained quadratic programs. Mathematical Programming Ser. B **103**, 251–282 (2005)
10. D'Ambrosio, C., Lodi, A., Martello, S.: Piecewise linear approximation of functions of two variables in MILP models. Operations Research Letters **38**, 39–46 (2010)
11. Rebennack, S., Kallrath, J.: Continuous piecewise linear delta-approximations for univariate functions: computing minimal breakpoint systems. Journal of Optimization Theory and Applications (2014). Submitted
12. Kallrath, J., Rebennack, S.: Computing area-tight piecewise linear overestimators, underestimators and tubes for univariate functions. In: S. Butenko, C. Floudas, T. Rassias (eds.) Optimization in Science and Engineering. Springer (2014)
13. Vielma, J.P., Nemhauser, G.: Modeling disjunctive constraints with a logarithmic number of binary variables and constraints. Mathematical Programming **128**, 49–72 (2011)
14. Horst, R., Pardalos, P.M., Thoai, N.V.: Introduction to global optimization, 2nd edn. Kluwer (2000)