J Glob Optim (2017) 69:927–949
DOI 10.1007/s10898-017-0548-3

Implementation of Cartesian grids to accelerate Delaunay-based derivative-free optimization

Pooriya Beyhaghi[1] · **Thomas Bewley**[1]

Received: 17 July 2016 / Accepted: 17 July 2017 / Published online: 9 August 2017

Abstract This paper introduces a modification of our original Delaunay-based optimization algorithm (developed in JOGO DOI:10.1007/s10898-015-0384-2) that reduces the number of function evaluations on the boundary of feasibility as compared with the original algorithm. A weaknesses we have identified with the original algorithm is the sometimes faulty behavior of the generated uncertainty function near the boundary of feasibility, which leads to more function evaluations along the boundary of feasibility than might otherwise be necessary. To address this issue, a second search function is introduced which has improved behavior near the boundary of the search domain. Additionally, the datapoints are quantized onto a Cartesian grid, which is successively refined, over the search domain. These two modifications lead to a significant reduction of datapoints accumulating on the boundary of feasibility, and faster overall convergence.

Keywords Derivative-free optimization · Surrogate functions · Delaunay triangulation · Cartesian grid

1 Introduction

In this paper, a derivative-free optimization algorithm is presented to minimize a (possibly nonconvex) function subject to bound constraints:[1]

$$\text{minimize } f(x) \text{ with } x \in L = \{x | a \leq x \leq b\}, \quad a < b. \tag{1}$$

where $x \in \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$, $a, b \in \mathbb{R}^m$.

[1] Taking a and b as vectors, $a \leq b$ implies that $a_i \leq b_i \ \forall i$.

✉ Pooriya Beyhaghi
p.beyhaghi@gmail.com

Thomas Bewley
bewley@eng.ucsd.edu

[1] Flow Control Lab, University of California, San Diego, La Jolla, CA, USA

The algorithm presented is a modification of the original algorithm for Delaunay-based derivative-free optimization via global surrogates, dubbed Δ-DOGS, developed in [8,9]. In this paper, we will assume that there is a target value f_0 which is achievable ($\exists\, x \in L$ such that $f(x) \leq f_0$), and the goal is to find a point such that $f(x) \leq f_0$. As done in Algorithm 2 of [8], the present algorithm can be modified to problems for which a target value of f_0 is unavailable (as it discussed in the "Appendix"). However, in this paper, we will assume that this target value f_0 is known.

Derivative-free algorithms are suitable for such problems even if neither the gradient of $f(x)$ nor an accurate numerical approximation of this gradient is readily available. This is common in situations in which the function $f(x)$ is derived either from an experiment or from many types of numerical simulations. There are three main classes of deterministic derivative-free algorithms.

The first class is Direct Search Methods, as reviewed in [15], including Pattern Search Methods [21,22] and Mesh Adaptive Direct search [1,4–6]. These methods are generally used to identify a local minimum of a function from some initial guess in parameter space. The more difficult challenge of attempting to identify accurately the global minimum of a nonconvex function $f(x)$, with as few function evaluations as possible, is an issue of significant interest.

The second class of derivative-free optimization algorithms is branch and bound methods, which can be used for the purpose of global optimization. In such algorithms, the search region at each iteration is constrained based on some condition imposed on the objective function. The most common such condition imposed is a bound on the Lipschitz norm, as studied in [11,13,17,20].

The third class of derivative-free optimization algorithms is Response Surface Methods, which employ an interpolating model of the available function evaluations at each iteration in order to summarize the trends evident in the available data. Expected Improvement, Baysian optimization, and other Kriging-based optimization strategies [2,7,10,14,18,19] are the most well-known methods in this class. With such methods, an interpolation strategy is used which inherently builds both an estimate of the function itself, $p(x)$, as well as a model of the uncertainty of this estimate, $e(x)$, over the entire feasible domain of parameter space.

Kriging interpolation, and other correlation-based interpolation strategies, have various shortcomings, the most significant of which is the numerical stiffness of the computational problem of fitting the Kriging model to the datapoints, and the subsequent inaccuracy of this fit. This problem is exacerbated when many datapoints are available, some of which are clustered in a small region of parameter space (see, e.g, the Appendix of [8]). Another important consideration is that the selection of the best interpolation strategy is an application-specific problem; based on various known characteristics of the cost function of interest, different interpolation strategies might be best suited.

Delaunay-based optimization [3,8,9] is a recently-developed response surface method that can uses any interpolation strategy, together with an artificially-generated function modeling the uncertainty associated with this interplant, to find the global minimum of an objective function. This uncertainty model is built on the framework of a Delaunay triangulation of the available datapoints in parameter space. One of the challenges with these new algorithms is their overexploration of the boundary of feasibility. This problem might be exacerbated if the objective function itself has irregular behavior close to the boundary of L, which is common. In this paper, we develop a modified Delaunay-based optimization algorithm which performs fewer function evaluations on the boundary of feasibility.

The structure of this paper is as follows: Sect. 2 reviews the original elements of the Delaunay-based optimization algorithm [8]. Section 3 presents the modified algorithm, and all

the tools that are needed. Section 5 proves the convergence to a point whose objective function is less than the target value for Lipschitz continuous and twice differentiable functions. Section 6 applies both the original and the modified optimization algorithm to a representative example. Some conclusions are presented in Sect. 7.

2 The original Delaunay-based optimization algorithm

In this section, we review our original algorithm for Delaunay-based Derivative-free Optimization via Global Surrogates, dubbed Δ-DOGS. This algorithm is a global, derivative-free optimization algorithm to solve (1) using successive function evaluations inside a feasible domain to find the global minimum. At each iteration of the algorithm, a metric based on an interpolation of the existing function evaluations, a model of the uncertainty of this interpolation, and the target value of the optimization f_0 is used to find the best possible candidate point for the next function evaluation. In this work, the interpolating function and the uncertainty function at iteration k are denote by $p^k(x)$ and $e^k(x)$, respectively. This method can handle any well-behaved interpolation strategy which is twice differentiable with bounded Hessian at all iterations. For the uncertainty function, a piecewise quadratic function is used which is nonnegative everywhere, and which goes to zero at the available datapoints. The uncertainty function is defined as follows:

Definition 1 Consider S as a set of feasible points which includes the vertices of L, and Δ as a Delaunay triangulation of S. Then, for each simplex $\Delta_i \in \Delta$, the local uncertainty function is defined as:

$$e_i(x) = R_i^2 - \|x - Z_i\|^2, \tag{2}$$

where R_i and Z_i are the circumradius and circumcenter of Δ_i. The global uncertainty function $e(x)$ is a piecewise function defined as follows:

$$e(x) = e_i(x) \quad \forall x \in \Delta_i. \tag{3}$$

The uncertainty function $e(x)$ has a number of properties which are established in Lemmas [2:5] of [8], as listed bellow:

a. The uncertainty function $e(x) \geq 0$ for all $x \in L$, and $e(x) = 0$ for all $x \in S$.
b. The uncertainty function $e(x)$ is continuous and Lipschitz.
c. The uncertainty function $e(x)$ is piecewise quadratic, with Hessian of $-2\,I$.
d. The uncertainty function $e(x)$ is equal to the maximum of the local uncertainty functions:

$$e(x) = \max_{1 \leq i \leq n} e_i(x). \tag{4}$$

The steps of Δ-DOGS are presented in Algorithm 1. Implementation and proof of convergence of this Algorithm are given in [8]. As mentioned previously, one of the weaknesses of this algorithm is the sometimes irregular behavior of the uncertainty function $e(x)$ close to the boundary of feasibility (see Fig. 1). This issue was addressed in Sect. 4 of [8] by projecting the point x_k onto the boundary of feasibility whenever x_k is, in a certain sense, close to the boundary of feasibility. Using this approach, the irregular behavior of the uncertainty function close to the boundary of L is reduced somewhat; however, further reduction of the accumulation of datapoints along the boundary of feasibility is sometimes desired, and is achieved by the strategy described in the remainder of this paper.

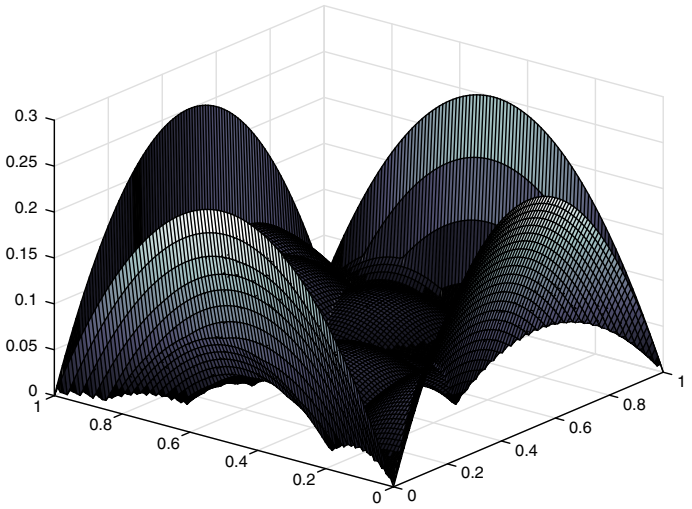

Fig. 1 Illustration of the uncertainty function $e(x)$ for a set S of 10 randomly-selected points within a square domain together with the vertices of the domain. It is seen that $e(x)$ has irregular behavior near the boundary of domain

Algorithm 1 The steps of the original Delaunay-based optimization algorithm, Δ-DOGS are as follows:

0. Set $k = 0$. Take the set of **initialization points** S^0 as the union of all vertices of the feasible domain L together with any user-supplied initialization points of interest (see Sect. 1 in [8] for implementation).
1. Calculate (or, for $k > 0$, update) an appropriate interpolating function $p^k(x)$ through all points in S^k.
2. Calculate (or, for $k > 0$, update) a Delaunay triangulation Δ^k over all of the points in S^k.
3. Find x_k as a global minimizer of $s^k(x)$ in L to obtain x_k, where

$$s^k(x) = \begin{cases} \frac{p^k(x)-f_0}{e^k(x)}, & \text{if } p^k(x) \geq f_0, \\ p^k(x) - f_0, & \text{otherwise}, \end{cases} \tag{5}$$

   where $e^k(x)$ is the uncertainty function (see Definition 1) for the datasset S^k.
4. Take $S^{k+1} = S^k \cup \{x_k\}$, increment k, calculate $f(x_k)$, and repeat from 1.

3 Acceleration of Delaunay-based optimization with a grid

This section presents, in Algorithm 2, the new optimization algorithm, dubbed Δ-DOGS(Z), as well as its essential elements. Before explaining Algorithm 2, some preliminary concepts are required.

Definition 2 Taking $N_\ell = 2^\ell$, the Cartesian grid of level ℓ over the feasible domain $L = \{x | a \leq x \leq b\}$, denoted L_ℓ, is defined as follows:

$$L_\ell = \left\{ x | x_i = a_i + \frac{b_i - a_i}{N_\ell} \cdot z, \quad z \in \{0, 1, \ldots, N_\ell\}, \quad i \in \{0, 1, \ldots, n\} \right\}.$$

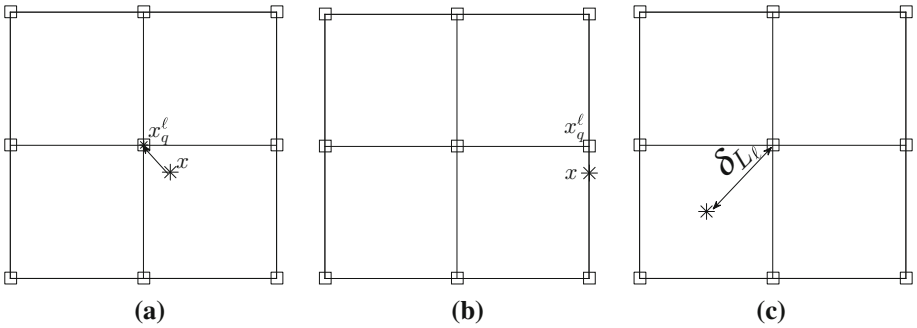

Fig. 2 Representation of a 2D Cartesian grid (with $\ell = 1$), the process of quantization, and the maximum quantization error δ_{L_ℓ}. Gridpoints are denoted by *open squares*, and the point of interest x denoted by a *star*. In *middle figure*, note that the constraints that are binding at x are also binding at the quantization of this point, x_q^ℓ; this is always true when quantizing to a Cartesian grid, a fact which is specifically leveraged in the proof of Lemma 1. **a** Quantization of an interior point (w/o binding constraints). **b** Quantization of a boundary point (w/ binding constriants). **c** Maximum quantization error (a.k.a. "covering radius")

The *quantization* of a point x onto the grid L_ℓ, denoted x_q^ℓ, is a point on the grid which has the minimum distance from x. Note that this quantization process might have multiple solutions; any of these solutions is acceptable. The maximum quantization error (i.e., in the language of sphere packing theory, the "covering radius") of the grid, δ_{L_ℓ}, is defined as follows:

$$\delta_{L_\ell} = \max_{x \in L_\ell} \|x - xq\| = \frac{\|b - a\|}{2N_\ell}. \tag{6}$$

Remark 1 There are three important properties of the Cartesian grid which are used in our optimization algorithm.

a. The grid of level ℓ covering the feasible domain L in an n dimensional space has $(N_\ell + 1)^n$ grid points. Such a grid is best suited for an approximately square domain L; for rectangular domains with high aspect ratios, this grid is easily generalized, as discussed in Remark 2.
b. $\lim_{\ell \to \infty} \delta_{L_\ell} = 0$.
c. If x_q is a quantization of x onto L_ℓ, then $A_a(x) \subseteq A_a(x_q)$, where $A_a(x)$ is the set of active constraints at x. This point is illustrated in Fig. 2.

Remark 2 The square Cartesian grid proposed in Definition 2 is easily generalized to a rectangular Cartesian grid by defining

$$L_\ell = \left\{ x | x_i = a_i + \frac{b_i - a_i}{N_{\ell,i}} \cdot z^i, \quad z^i \in \{0, 1, \ldots, N_{\ell,i}\}, \quad i \in \{0, 1, \ldots, n\} \right\},$$

where $N_{\ell,i} = c_i\, 2^\ell$ for small integers c_i, which are selected such that the grid spacings of the initial grid, $\Delta x_{0,i} \triangleq (b_i - a_i)/N_{0,i}$, are approximately equal in each direction i. For rectangular domains L with high aspect ratios, a grid defined in such a manner is significantly better suited.

Definition 3 Consider x as a point in L, and S as a nonempty set of points in L, such that $z \in S$ is the closest point in S from x. The pair (x, S) is called *activated* if and only if $A_a(x) \subseteq A_a(z)$, where $A_a(x)$ is the set of active constraints at x. Note that the domain L has a total of $2\,n$ constraints.

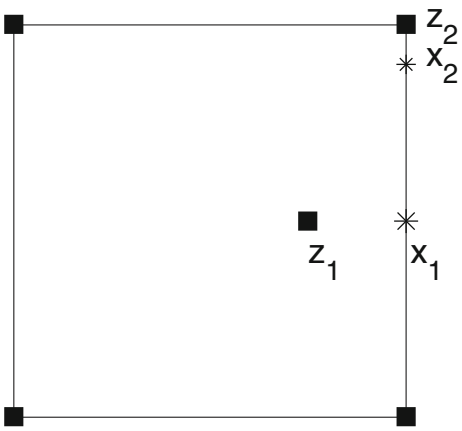

Fig. 3 The set S is shown by *black squares*. The pair (x_1, S) is not activated, since z_1 (the closest point to x_1 in S) is not on the same boundary of L that x_1 lies. The pair (x_2, S) is activated, since z_2 (the closest point to x_2 in S) is located on the same boundary of L that x_2 lies

Remark 3 If there are multiple points z which share the minimum distance from x in S, then the pair (x, S) is activated if, for all such z, $A_a(x) \subseteq A_a(z)$.

Remark 4 If x is on the interior of L, then the pair (x, S) is activated for any nonempty set S. However, if x is on the boundary of L, the pair (x, S) may or may not activated, depending on the position of x and the points in S (see Fig. 3).

Definition 4 Consider S as a set of points in L which is partitioned into two subsets, $S = S_E \cup S_U$, as follows:

- The evaluated points are denoted S_E, where the function values are available.
- The support points are denoted S_U, where the function values are not available. The support points will be helpful when developing the triangulation.

The *continuous search function* (see Fig. 4) at iteration k, denoted $s_c^k(x)$, is defined for all $x \in L$ such that

$$s_c^k(x) = \begin{cases} \dfrac{p^k(x) - f_0}{e^k(x)} & \text{if } p^k(x) \geq f_0, \\ p^k(x) - f_0 & \text{otherwise,} \end{cases} \tag{7}$$

whereas the *discrete search function* (see Fig. 4) at iteration k, denoted $s_d^k(x)$, is defined for all $x \in L$ such that

$$s_d^k(x) = \begin{cases} \dfrac{p^k(x) - f_0}{\text{Dis}\{x, S_E^k\}} & \text{if } p^k(x) \geq f_0, \\ p^k(x) - f_0 & \text{otherwise,} \end{cases} \tag{8}$$

where $e^k(x)$ is the uncertainty function (see Definition 1) constructed with all the points in S^k, $p^k(x)$ is an interpolating function passing through all the points in S_E^k, and $\text{Dis}\{x, S_E^k\} = \min_{z \in S_E^k} \|x - z\|$.

Remark 5 Note that the continuous search function $s_c^k(x)$ used here is similar to the search function $s^k(x)$ defined and used in [8,9]. However, the uncertainty function $e(x)$ and the interpolating function $p(x)$, upon which $s_c^k(x)$ is based, are developed based on two different set of points (S^k and S_E^k, respectively).

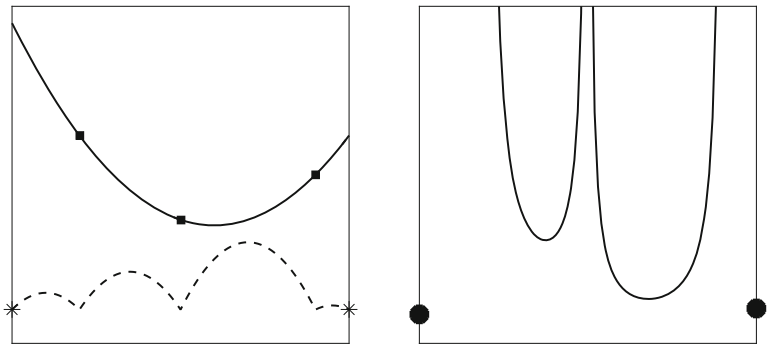

Fig. 4 Illustration of continuous and discrete search function. **a** Illustrates: (*solid line*) the interpolating function $p(x)$, (*dashed line*) the uncertainty function $e(x)$, (*black squares*) evaluation points S_E, and (*stars*) support points S_U. **b** Illustrates: (*sold line*) the continuous search function $s_c^k(x)$, and (*closed circles*) the discrete search function $s_d^k(x)$. **a** $p(x)$, $e(x)$, S_E and S_U. **b** $s_c(x)$ and $s_d(x)$

Now we have all the tools necessary to present the modified optimization algorithm considered in this work. The following three key modifications to Δ-DOGS are performed to obtain Δ-DOGS(Z), as listed in Algorithm 2:

1. The datapoints in Algorithm 2 are restricted to lie on the Cartesian grid, which is occasionally refined as the iteration proceeds.
2. At each iteration, two different sets of points are considered, S_E and S_U. Function evaluations are available only for the points in S_E.
3. Two different search functions, $s_c(x)$ and $s_d(x)$, are considered at each iteration. One of them, $s_c(x)$, is minimized over the entire feasible domain L. The other, $s_d(x)$, is minimized only over the points in S_U.

Defining x_k as the minimizer of the continuous search function $s_c^k(x)$ in L, y_k as the quantization of x_k onto the grid L_ℓ, and w_k as the minimizer of the discrete search function $s_d^k(x)$ in S_U^k, there are four possible cases at each iteration of Algorithm 2, corresponding to four of the numbered steps of this algorithm:

(6) The pair (x_k, S^k) is not activated. This is called an *inactivated step*: y_k is simply added to S_U^k, and no function evaluation is performed. [Note that the other three steps below, in contrast, are said to be *activated*.]
(7) The pair (x_k, S^k) is activated and $s_d^k(w_k) < s_d^k(x_k)$. This is called an *evaluating step*: w_k is removed from S_U^k, added to S_E^k, and $f(w_k)$ calculated.
(8) The pair (x_k, S^k) is activated, $s_d^k(x_k) \leq s_d^k(w_k)$, and $y_k \notin S_E^k$. This is called an *identifying step*: the new point y_k is added to S_E^k, and $f(y_k)$ is calculated.
(9) The pair (x_k, S^k) is activated, $s_d^k(x_k) \leq s_d^k(w_k)$, and $y_k \in S_E^k$. This is called a *grid refinement step*: L_ℓ is refined, and the sets S_E^k and S_U^k are unchanged.

At any given iteration k of Algorithm 2, exactly one of the above four cases applies, and the corresponding step is taken. Iterations at which evaluating and identifying steps are taken are illustrated in Fig. 5 (note that, in 1D, all iterations after the initialization are activated).

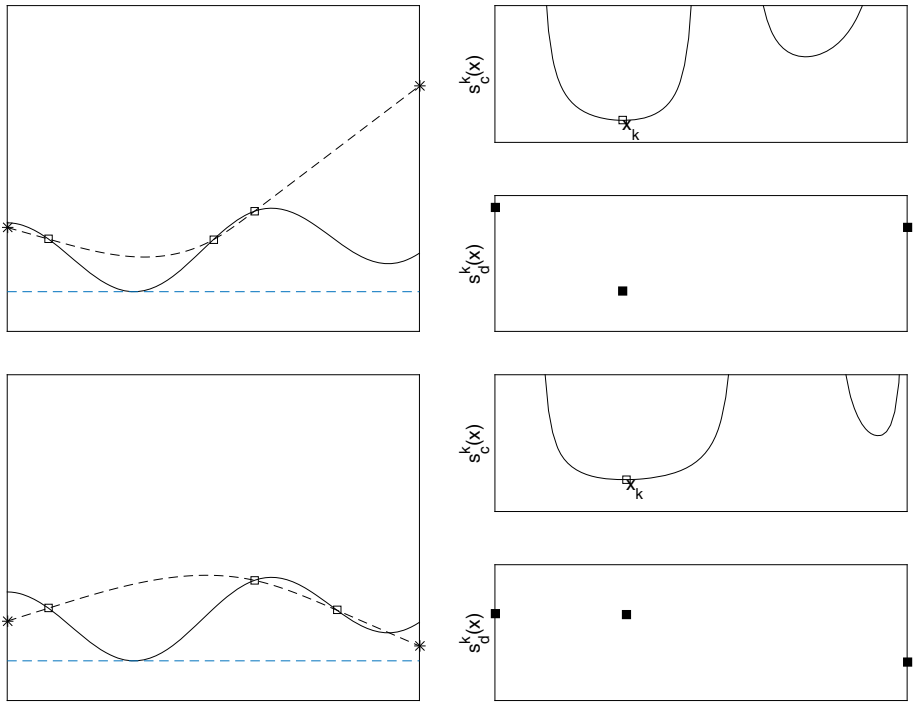

Fig. 5 Illustration of *identifying* (*first row*) and *evaluating* (*second row*) iterations of Algorithm 2. *Left figures*: (*dashed line*) the interpolating function $p^k(x)$; (*solid line*) the objective function $f(x)$; (*open squares*) evaluated points S_E^k; (*stars*) support points S_U^k. *Right figures*: (*solid line*) the continuous search function $s_c^k(x)$; (*closed squares*) the discrete search function $s_d^k(x)$; (*open square*) the global minimizer x_k of the continuous search function $s_c^k(x)$. **a** (*left*) $p^k(x)$ and f(x), and (*right*) $s_c^k(x)$ and s_d^k (x), for an iteration at which an *identifying* step is performed. See caption below for legend, **b** (*left*) $p^k(x)$ and f(x), and (*right*) $s_c^k(x)$ and $s_d^k(x)$, for an iteration at which an *evaluating* step is performed

Algorithm 2 The steps of the Modified Delaunay-based optimization algorithm, Δ-DOGS(Z) are as follows:

0. Set $k = 0$ and initialize ℓ. Take the initial set of support points S_U^0 as all 2^n vertices of the feasible domain L. Choose at least $n + 1$ points on the initial grid, $n + 1$ of which are affinely independent, put them in S_E^0, and calculate $f(x)$ at each of these points.
1. Calculate (or, for $k > 0$, update) an appropriate interpolating function $p^k(x)$ through all points in S_E^k.
2. Calculate (or, for $k > 0$, update) a Delaunay triangulation Δ^k over all of the points in $S^k = S_U^k \cup S_E^k$.
3. Find x_k as the minimizer of $s_c^k(x)$ (see Definition 4) in L, and take y_k as its quantization onto the grid L_ℓ.
4. Find w_k as the minimizer of $s_d^k(x)$ (see Definition 4) in S_U^k.
5. If the pair (x_k, S^k) is not activated (see Definition 3), then take $S_U^{k+1} = S_U^k \cup \{y_k\}$ and $S_E^{k+1} = S_E^k$, increment k, and repeat from 2.
6. If $s_d^k(w_k) \leq s_d^k(x_k)$, then take $S_U^{k+1} = S_U^k - \{w_k\}$ and $S_E^{k+1} = S_E^k \cup \{w_k\}$, calculate $f(w_k)$, and increment k; if $f(w_k) > f_0$, repeat from 2, otherwise halt.

7. If $y_k \notin S_E^k$, then take $S_U^{k+1} = S_U^k$ and $S_E^{k+1} = S_E^k \cup \{y_k\}$, calculate $f(y_k)$, and increment k; if $f(y_k) > f_0$, repeat from 2, otherwise halt.
8. Take $S_U^{k+1} = S_U^k$ and $S_E^{k+1} = S_E^k$, increment both k and ℓ, and repeat from 1.

4 Trust restriction based on the decrease of the interpolating function

We now describe a modification of Algorithm 2 which improves its convergence. At each iteration, $s_c^k(x)$ must be minimized over all $x \in L$, and $s_d^k(x)$ must be minimized over all $x \in S_U^k$. Note that, if $s_c^k(x_k) < 0$, then there is a point $x \in L$ for which $p^k(x) < f_0$, and thus $p^k(x_k) < f_0$.

Definition 5 Those iterations of Algorithm 2 for which $p^k(x_k) < f_0$ are called *trust restriction* iterations.

If $p^k(x_k)$ is much less than f_0, the value of the interpolation itself may be unreliable near x_k. This can happen only when x_k is, in a sense, far from the available datapoints (at which the function values are all greater than f_0). In this case, we propose a strategy to identify a point x for which $p^k(x) = f_0$ (which is, thus, closer to the existing datapoints), and evaluate the function at this new point instead. This approach, akin to the venerable *trust region* approach (see, e.g., [16]), is a more promising strategy for rapidly finding a value of x for which $f(x) \leq f_0$, as the algorithm focuses at any given iteration k on promising regions in L where the interpolant is, in a sense, reasonably reliable.

To accomplish this, at the end of step 4 of Algorithm 2, it is checked whether or not $p^k(x_k) < f_0$. If it is (that is, if this is a trust restriction iteration), then a point x_c is identified as the closest point in S_E^k to x_k; since the algorithm has not yet terminated, $p^k(x_c) > f_0$. There is thus at least one point $\hat{x}$ on the line segment between x_c and x_k such that $p^k(\hat{x}) = f_0$. Finding such a point $\hat{x}$ (see, for example, Fig. 6) is a simple one-dimensional root finding problem for the computationally inexpensive function $p^k(x) - f_0$. A false position method may be used to find $\hat{x}$ such that $p^k(\hat{x}) = f_0$. The point x_k is then replaced by $\hat{x}$, and Algorithm 2 proceeds from step 5 as before.

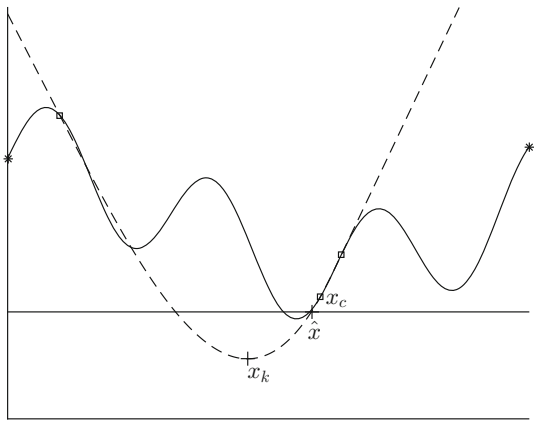

Fig. 6 Illustration of a trust restriction iteration of Algorithm 2: (*dashed line*) interpolating function $p^k(x)$; (*solid line*) objective function $f(x)$; (*open squares*) evaluated points S_E^k; (*stars*) support points S_U^k; and (*solid horizontal line*) target value f_0

5 Analysis of the new algorithm

We now analyze the convergence properties of Algorithm 2. If the algorithm terminates after finite number of iterations k, then a point x_k is found for which the function value is less than or equal to the target value f_0; otherwise, all computed values of the objective function are greater than the target value. In this section, we will show, in the latter case, that a limit point of the datapoints that are obtained in the evaluation set S_E includes a feasible point whose objective function is equal to the target value. Therefore, for this analysis, we will assume that Algorithm 2 proceeds for an infinite number of iterations.

Before analyzing the convergence of Algorithm 2, we first show that Algorithm 2 includes an infinite number of mesh refinements. To show this, a preliminary lemma is first established.

Lemma 1 *If k as an inactivated iteration of* Algorithm 2*; then, $y_k \notin S^k$.*

Proof We establish this lemma by contradiction. Assume that $y_k \in S^k$, and that iteration k is inactivated; that is, assume that there is a point $z_k \in S^k$ with minimum distance from x_k for which $A_a(x_k) \nsubseteq A_x(z_k)$. Since all points in S^k are on the grid L_ℓ of iteration k, and y_k is a quantizer of x_k on this grid, $\|x_k - y_k\| \leq \|x_k - z_k\|$. On the other hand, since $y_k \in S^k$ and z_k is the closest point to x_k in S^k, $\|x_k - y_k\| \geq \|x_k - z_k\|$. This leads to $\|x_k - z_k\| = \|x_k - y_k\|$. As a result; the point z_k is also a quantizer of x_k on the Cartesian grid L_ℓ, which is in contradiction with $A_a(x_k) \nsubseteq A_a(z_k)$, as illustrated in Fig. 2. □

Theorem 1 *There are an infinite number of mesh refinement iterations if* Algorithm 2 *proceeds without terminating.*

Proof This theorem is also estabshied by contradiction. Assume that there are a finite number of mesh refinement iterations as Algorithm 2 proceeds, then all datapoints must lie on a grid with some level ℓ. At each iteration of Algorithm 2, if it is an identifying iteration, then $|S_E^k|$ and $|S^k|$ are both incremented by one. If it is an evaluating iteration, then $|S_E^k|$ is incremented by one and $|S^k|$ is fixed. if it is inactivated, then $|S_E^k|$ is fixed and $|S_E^k|$ is incremented by one. Therefore, at each iteration of the algorithm which is not mesh refinement, we will increment the value of $|S^k| + |S_E^k|$ by at least one. Since the number of points on the grid of level ℓ is finite, we must have only finite number of iterations which are not mesh refinements, which is in contradiction with the fact that there are infinite number of iterations for Algorithm 2.□

We now analyze the convergence of Algorithm 2. To do this, the following conditions are imposed for the objective and interpolating functions.

Assumption 1 The interpolating functions $p^k(x)$, objective function $f(x)$, and $p^k(x) - f(x)$ are Lipschitz with the same Lipschitz constant $\hat{L}$.

Assumption 2 A constant $\hat{K} > 0$ exists for which

$$\nabla^2\{f(x) - p^k(x)\} + 2\,\hat{K}\,I > 0, \quad \forall x \in L \text{ and } k > 0, \tag{9}$$

$$\nabla^2\{p^k(x)\} - 2\,\hat{K}\,I < 0, \quad \forall x \in L \text{ and } k > 0, \tag{10}$$

$$\nabla^2\{f(x)\} - 2\,\hat{K}\,I < 0, \ \forall x \in L. \tag{11}$$

We now establish four Lemmas which together help to prove convergence. In the first, we determine a bound for the maximum violation from the local minimum of a general twice differentiable function from its local minimum. This bound is used in Lemma 4 to prove that a solution will be obtained as the location of the datapoints become dense in the feasible domain.

Lemma 2 *Consider $G(x)$ as a twice differentiable function such that $\nabla^2 G(x) - 2\,K_1 I \leq 0$, and $x^* \in L$ as a local minimizer of $G(x)$ in L. Then, for each $x \in L$ such that $A_a(x^*) \subseteq A_a(x)$, we have*:

$$G(x) - G(x^*) \leq K_1 \|x - x^*\|^2. \tag{12}$$

Proof Define function $G_1(x) = G(x) - K_1 \|x - x^*\|^2$. By construction, $G_1(x)$ is concave; therefore,

$$\begin{aligned} G_1(x) &\leq G_1(x^*) + \nabla G_1(x^*)^T (x - x^*), \\ G_1(x^*) &= G(x^*), \quad \nabla G_1(x^*) = \nabla G(x^*), \\ G(x) &\leq G(x^*) + \nabla G(x^*)^T (x - x^*) + K_1 \|x - x^*\|^2. \end{aligned}$$

Since the feasible domain is a bounded domain, the constrained qualification holds (see [12]); therefore, x^* is a KKT point. Therefore, using $A_a(x^*) \subseteq A_a(x)$ leads to $\nabla G(x^*)^T (x - x^*) = 0$, which verifies (12). □

Lemma 3 *Consider k as an iteration of* Algorithm 2 *which is activated and a trust restriction. Then*

$$p^k(z_k) - f_0 \leq 2\,\{K + \hat{K}\}\|x_k - z_k\|^2, \tag{13}$$

where $K = s_c^k(x_k) > 0$.

Proof Since x_k is a global minimizer of $s_c^k(x) = \frac{p^k(x) - f_0}{e^k(x)}$, and $s_c^k(x) \geq 0$ for all $x \in L$, then x_k is a global minimizer of $T^k(x) = p^k(x_k) - K\,e_k(x)$ too, and $T^k(x_k) = f_0$ (see Sect. 5 in [8] for discussion of why). Consider $\Delta_i^k \in \Delta^k$ as the simplex which includes x_k. By construction, $e^k(x_k) = e_i^k(x_k)$. Define $T_i^k(x) = p^k(x) - K\,e_i^k(x)$, then $T_i^k(x)$ is a twice differentiable function in L, and

$$\nabla^2 T_i^k(x) = \nabla^2\{p^k(x)\} + 2\,K\,I, \quad \nabla^2 T_i^k(x) - 2\{\hat{K} + K\}I \leq 0.$$

By (4), $e_i^k(z_k) \leq e^k(z_k)$, which leads to $T^k(x) \leq T_i^k(x)$ for all points $x \in L$, x_k is a global minimizer of $T^k(x)$, and $T^k(x_k) = T_i^k(x_k)$. Therefore, x_k is a global minimizer of $T_i^k(x)$ as well.

Since iteration k is activated, $A_a(x_k) \subseteq A_a(z_k)$; thus, using Lemma 2, we have:

$$\begin{aligned} T_i^k(z_k) - T_i^k(x_k) &\leq 2\,\{K + \hat{K}\}\|z_k - x_k\|^2. \\ T^k(z_k) \leq T_i^k(z_k), &\quad T^k(x_k) = T_i^k(x_k), \\ T^k(z_k) - T^k(x_k) &\leq 2\,\{K + \hat{K}\}\|z_k - x_k\|^2, \\ T^k(z_k) - f_0 &\leq 2\,\{K + \hat{K}\}\|z_k - x_k\|^2. \end{aligned}$$

Since $z_k \in S_E^k$, $e^k(z_k) = 0$ and $p^k(z_k)$, which leads to $T^k(z_k) = p^k(z_k)$ which shows (13).□

Lemma 4 *Consider x^* as a global minimizer of $f(x)$ in L. Then, for each iteration of* Algorithm 2 *which is not a trust restriction, we have*:

$$\min\left\{\frac{s_c^k(x^*)}{\hat{K}}, \min_{z \in S_U^k}\left\{\frac{s_d^k(z)}{\hat{L}}\right\}\right\} \leq 2. \tag{14}$$

Proof Consider Δ_i^k as a simplex in Δ^k which includes x^* whose vertices are $\{V_1^k, V_2^k, \ldots, V_{n+1}^k\}$. Define $L^k(x)$ as the unique linear function in Δ_i^k such that $L(V_i^k) = 2\,f(V_i^k) -$

$p^k(V_i^k)$, and define $G^k(x) = p^k(x) + L^k(x) - 2\,\hat{K}\,e^k(x) - 2\,f(x)$. Then, for each vertex $V_i^k(x)$ of Δ_i^k,

$$G^k(V_i) = p^k(V_i) + L^k(V_i^k) - 2\hat{K}\,e^k(V_i) - 2\,f(V_i^k) = 0.$$

Moreover, since $\nabla^2 L_k(x) = 0$, and $\nabla^2 e^k(x) = -2\,I$ inside the simplex Δ_i^k, then according to Assumption 2, $\nabla^2 G^k(x) \geq 0$. Thus, $G^k(x)$ is convex in Δ_i^k, and its maximum is located at one of its vertices; therefore,

$$G^k(x^*) \leq 0, \quad p^k(x^*) + L^k(x^*) - 2\,f(x^*) - 2\,\hat{K}\,e^k(x^*) \leq 0.$$

Since f_0 is assumed to be achievable, $f_0 \geq f(x^*)$, and thus

$$p^k(x^*) + L^k(x^*) - 2\,f_0 - 2\,\hat{K}\,e^k(x^*) \leq 0.$$

Since $x^* \in \Delta_i^k$ and $L^k(x)$ is linear, it follows that

$$\begin{aligned}
&\min_{1\leq j\leq n+1} L^k\left(V_j^k\right) \leq L^k(x^*),\\
&\min_{1\leq j\leq n+1}\left\{2\,f\left(V_j^k\right) - p^k\left(V_j^k\right)\right\} = \min_{1\leq j\leq n+1} L^k\left(V_j^k\right),\\
&\min_{z\in S^k}\{2\,f(z) - p^k(z)\} \leq \min_{1\leq j\leq n+1}\left\{2\,f\left(V_j^k\right) - p^k\left(V_j^k\right)\right\},\\
&p^k(x^*) - f_0 - 2\,\hat{K}\,e^k(x^*) + \min_{z\in S^k}\{2\,f(z) - p^k(z) - f_0\} \leq 0.
\end{aligned} \tag{15}$$

Define $\hat{z}$ as the closest point to z in S_E^k. By construction, $p^k(\hat{z}) - f(\hat{z}) = 0$. Furthermore, by Assumption 1, the function $p^k(x) - f(x)$ is Lipschitz with constant $\hat{L}$, and thus

$$\begin{aligned}
p^k(z) - f(z) &\leq \hat{L}\|z - \hat{z}\| = \hat{L}\,\mathrm{Dis}(z, S_E^k),\\
p^k(z) - 2\,\hat{L}\,\mathrm{Dis}(z, S_E^k) &\leq 2\,f(z) - p^k(z).
\end{aligned} \tag{16}$$

Using (15) and (16) leads to:

$$p^k(x^*) - f_0 - 2\,\hat{K}\,e^k(x^*) + \min_{z\in S^k}\left\{p^k(z) - f_0 - 2\,\hat{L}\mathrm{Dis}\left(z, S_E^k\right)\right\} \leq 0. \tag{17}$$

Since iteration k is not a mesh refinement, $p^k(x) - f_0 > 0$ for all $x \in L$. Thus, $\frac{1}{s_d^k(x)}$ and $\frac{1}{s_c^k(x)}$ are well defined functions everywhere in L, and equation (17) can be rewritten as:

$$(p^k(x^*) - f_0)\left(1 - \frac{2\,\hat{K}}{s_d^k(x^*)}\right) + \min_{z\in S^k}\left\{\left(p^k(z) - f_0\right)\left(1 - \frac{2\,\hat{L}}{s_d^k(z)}\right)\right\} \leq 0 \tag{18}$$

Since $p^k(x^*) - f_0 > 0$ and $p^k(z) - f_0 > 0 \quad \forall z \in S^k$, (14) is verified. □

Lemma 5 *Consider k as a mesh refinement iteration of* Algorithm 2 *which is not a trust restriction. Then*

$$\min_{z\in S_E^k} f(z) - f_0 \leq \max\left\{3\,\hat{L}\,\delta_k, 6\,\hat{K}\,\delta_k^2\right\}, \tag{19}$$

where δ_k is the maximum discretization error of the Cartesian grid L_ℓ at this iteration.

Proof Since iteration k is mesh refinement, by construction, $s_d^k(x_k) \leq s_d^k(w_k)$. Additionally, x_k and w_k are the global minimizer of $s_c^k(x)$ and $s_d^k(x)$ in L and S_U^k respectively. Thus, by using (14) in Lemma 4, we have:

$$\min\left\{\frac{s_c^k(x_k)}{\hat{K}}, \frac{s_d^k(x_k)}{\hat{L}}\right\} \leq 2. \tag{20}$$

There are two possible cases: In the first case, $s_c^k(x_k) \leq 2\hat{K}$; thus, using Lemma 3, we have:

$$p^k(y_k) - f_0 \leq 2\,[2\hat{K} + \hat{K}]\|x_k - y_k\|^2 = 6\,\hat{K}\,\|y_k - x_k\|^2.$$

Since $y_k \in S_E^k$, $f(y_k) = p^k(y_k)$. Furthermore, $\|x_k - y_k\| \leq \delta_k$; thus, (19) is verified in this case. In the second case, $s_d^k(x_k) \leq 2\hat{L}$. Since $y_k \in S_E^k$, and all points in S_E^k are on the grid L_ℓ, it follows that $\mathrm{Dis}(x_k, S_E^k) = \|x_k - y_k\| = \delta_k$, and thus

$$p^k(x_k) - f_0 \leq 2\,\hat{L}\|x_k - y_k\|, \quad p^k(y_k) - p^k(x_k) \leq \hat{L}\|x_k - y_k\|,$$
$$f(y_k) - f_0 \leq 3\,\hat{L}\|x_k - y_k\| \leq 3\,\hat{L}\delta_k.$$

Thus, (19) is shown for both cases. □

Remark 6 If iteration k of Algorithm 2 is a mesh refinement and a trust restriction, then $p^k(x_k) = f_0$. Additionally, $p^k(x)$ is Lipschitz with constant $\hat{L}$; therefore,

$$p^k(y_k) - f_0 \leq \hat{L}\|x_k - y_k\| \leq \hat{L}\delta_k. \tag{21}$$

Moreover, $y_k \in S_E^k$, then

$$f(y_k) - f_0 \leq \hat{L}\|x_k - y_k\| \leq \hat{L}\delta_k. \tag{22}$$

Theorem 2 *If* Algorithm 2 *is not terminated at any iteration, then the set* $S^\infty = \lim_{k\to\infty} S^k$ *has a limit point, denoted* $v \in L$, *such that* $f(v) = f_0$.

Proof According to Theorem 1, there is an infinite number of mesh refinement iterations during the execution of Algorithm 2, denoted here $\{k_1, k_2, \ldots\}$. Consider $v_i \in argmin_{z\in S^{k_i}} f(z)$. According to Lemma 5 and Remark 6, we have:

$$f(v_i) - f_0 \leq \max\left\{3\,\hat{L}\,\delta_{k_i}, 6\,\hat{K}\,\delta_{k_i}^2\right\}, \tag{23}$$

Since Algorithm 2 is not terminated at any iteration, $f(v_i) - f_0 \geq 0$. Additionally, $\lim_{i\to\infty} \delta_{k_i} = 0$ and f(x) is continuous (see Assumption 1 and 2), which leads to $\lim_{i\to\infty} f(v_i) = f_0$. □

6 Results

In this section, we compare the performance of Δ-DOGS(Z), as given Algorithm 2, with the original Δ-DOGS algorithm, as given in Algorithm 1 (and originally presented as Algorithm 2 in [8]). Note that the new Δ-DOGS(Z) algorithm modifies the orginal Δ-DOGS algorithm in two essential ways:

Trust restriction that is, restricting the update based on the decrease of the interpolating function (Sect. 4), and

Grid quantization that is, restricting function evaluations to lie on a grid that is successively refined as convergence is approached (Sect. 3).

To characterize independently the performance gains associated with these two modifications, four different algorithms are considered:

1. The original Δ-DOGS algorithm, denoted in this paper as Algorithm 1.
2. The original Δ-DOGS with trust restriction, denoted Algorithm 1A.
3. The original Δ-DOGS with grid quantization, denoted Algorithm 1B.
4. The new Δ-DOGS(Z) algorithm, given by the original Δ-DOGS with both trust restriction and grid quantization, denoted in this paper as Algorithm 2.

6.1 Numerical tests on problems for which the solution is on the interior

In this section, numerical tests are performed on two different test problems, the Styblinski Tang test problem

$$f(x) = \sum_{i=1}^{n} \frac{x_i^4 - 16\,x_i^2 + 5\,x_i}{2} + 39.1660\,n, \quad \text{where} \quad L = \{x | -5 \le x_i \le 5\}, \tag{24}$$

and the Schwefel test problem

$$f(x) = 418.9829\,n - \sum_{i=1}^{n} x_i \sin(x_i), \quad \text{where} \quad L = \{x | 0 \le x_i \le 500\}. \tag{25}$$

The global minimum for each of these test problems is zero, and they have, respectively, 2^n and 4^n local minima inside their indicated search domains L, with global solutions at, respectively, $x_i = -2.907\ \forall i$, and $x_i = 420.9878\ \forall i$.

In this section, the Styblinski Tang test problem (24) is analyzed for $n = \{2, 3, 4, 5\}$, and the Schwefel test function (25) is analyzed for $n = \{2, 3, 4\}$. The target value considered in all optimizations performed is the actual value of the global minimum, $f_0 = 0$. For Algorithms 1B and 2, an initial grid level of $\ell_0 = 3$ is considered, and the optimizations are continued until the grid level of $\ell = 8$ is terminated. To facilitate a fair comparison, Algorithms 1 and 1A are terminated when $\text{Dis}(x_k, S^k) \le \delta_{L_8}$, where δ_{L_8} is the maximum quantization error of grid level $\ell = 8$, which leads to a comparable level of accuracy for all four methods tested.

For Algorithms 1B and 2, the initial $n + 1$ datapoints S_E^0 are given by

$$S_E^0 = \left\{ x^0, x^0 + \frac{b_i - a_i}{2^{\ell_0}}\, e^i,\ \forall i \in \{1, 2, \ldots, n\} \right\} \tag{26}$$

where, for each i, e^i is the i'th unit vector, and x^0 is an initial point on the grid of level ℓ_0. The results of Algorithms 1B and 2 will, naturally, depend upon the choice of x^0; thus, two different values for x^0 are considered. For the Styblinski Tang test function, we take (a) $x_i^{0,a} = 0\ \forall i$, and (b) $x_i^{0,b} = -2\ \forall i$. For the Schwefel test function, we take (a) $x_i^{0,a} = 100\ \forall i$, and (b) $x_i^{0,b} = 400\ \forall i$.

For the $n = 2$ cases, the position of the datapoints that are used during the optimization process are illustrated in Fig. 7 for problem (24), and Fig. 8 for problem (25). It is observed that Algorithm 2 significantly reduces the accumulation of datapoints on the boundary of the feasibility, which accelerates convergence.

It is seen that trust restriction modification sometimes improves convergence, particularly if a good initial value x^0 is used. However, this modification does not make a major improvement if extensive global exploration of the domain is initially required, before identifying the neighborhood of the global solution. The convergence histories in the higher-dimensional cases are shown in Fig. 9 for problem (24), and Fig. 10 for problem (25).

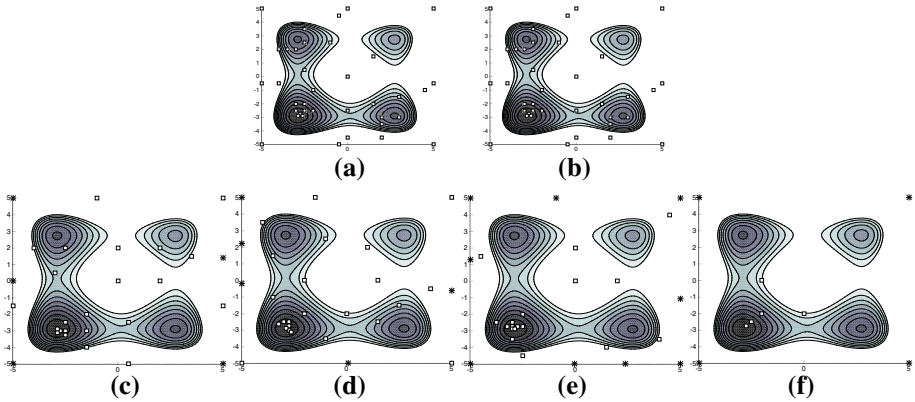

Fig. 7 Implementation of Algorithms 1, 1A, 1B, and 2 on problem (24) for $n = 2$ dimensions: (*open square*) evaluated points, (*stars*) support points. **a** Alg. 1, # of fn. evals: 37. **b** Alg. 1A, # of fn. evals: 37. **c** Alg. 1B, $x^{0,a}$, # of fn. evals: 22. **d** Alg. 1B, $x^{0,b}$, # of fn. evals: 24. **e** Alg. 2, $x^{0,a}$, # of fn. evals: 16. **f** Alg. 2, $x^{0,b}$, # of fn. evals: 5

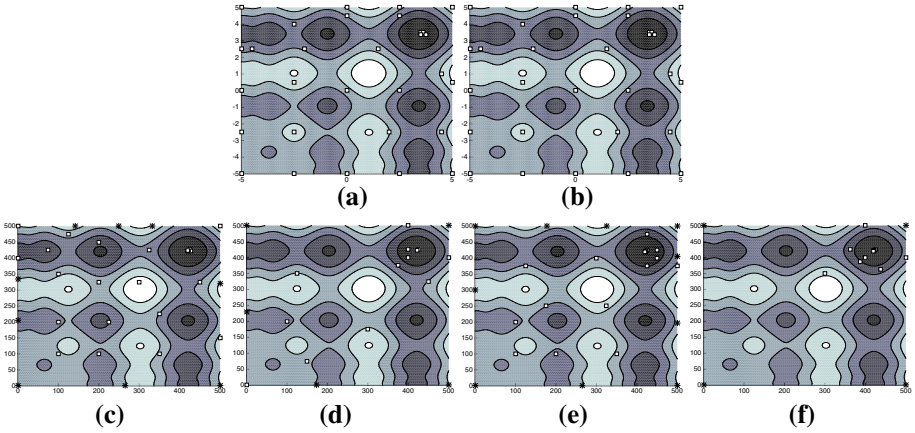

Fig. 8 Implementation of Algorithms 1, 1A, 1B, and 2 on problem (25) for $n = 2$ dimensions: (*open square*) evaluated points, (*stars*) support points. **a** Alg. 1, # of fn. evals: 30. **b** Alg. 1A, # of fn. evals: 30. **c** Alg. 1B, $x^{0,a}$, # of fn. evals: 17. **d** Alg. 1B, $x^{0,b}$, # of fn. evals: 11. **e** Alg. 2, $x^{0,a}$, # of fn. evals: 24. **f** Alg. 2, $x^{0,b}$, # of fn. evals: 13

The results are summarized in Table 1. Note that the performance of Algorithms 1B and 2 in the various dimensions reported in Table 1, for both problems (24) and (25), are compared by starting from 4 different initial points x^0 on the initial grid (generated uniformly randomly in the feasible domain), and averaging to determine the "typical" number of function evaluations and support points required.

To summarize, it is observed that the two modifications of Algorithm 1 that are presented in this article (that is, trust restriction and grid quantization) significantly and consistently improve its convergence behavior. A key reason for this is that fewer datapoints accumulate on the boundary of the feasible domain using the new Algorithm 2. Most of the boundary points that are used by Algorithm 1 are needed simply to regularize the triangulation; Algorithm 2 avoids performing function evaluations at these boundary points by dividing S^k into evaluated

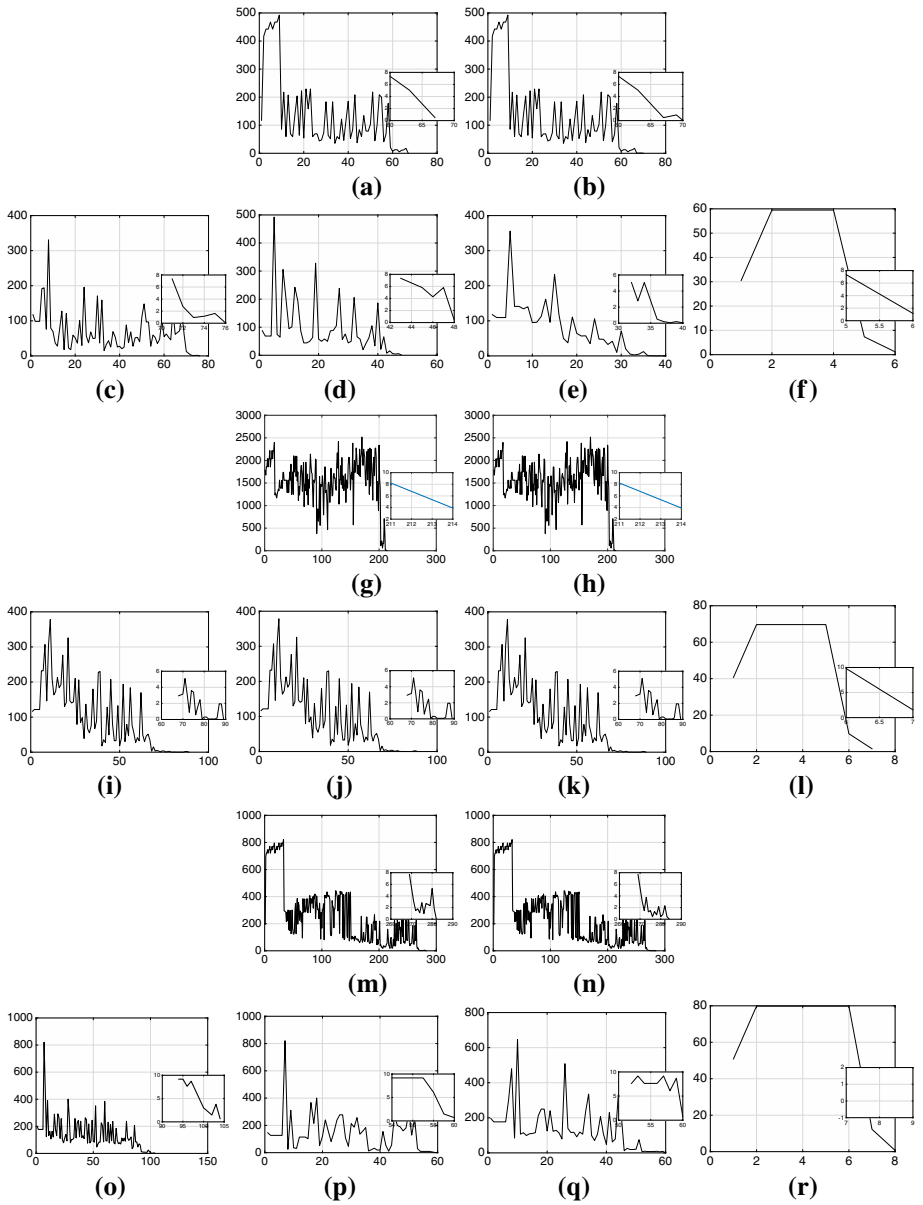

Fig. 9 Convergence histories for Algorithms 1, 1A, 1B, and 2 with two different initial points on problem (24) for $n = 3$, 4 and 5. The behavior near convergence is shown in the *insets*. **a** Alg. 1, $n = 3$. **b** Alg. 1A, $n = 3$. **c** Alg. 1B, $n = 3$, $x^{0,a}$. **d** Alg. 1B, $n = 3$, $x^{0,b}$. **e** Alg. 2, $n = 3$, $x^{0,a}$. **f** Alg. 2, $n = 3$, $x^{0,b}$. **g** Alg. 1, $n = 4$. **h** Alg. 1A, $n = 4$. **i** Alg. 1B, $n = 4$, $x^{0,a}$. **j** Alg. 1B, $n = 4$, $x^{0,b}$. **k** Alg. 2, $n = 4$, $x^{0,a}$. **l** Alg. 2, $n = 4$, $x^{0,a}$. **m** Alg. 1, $n = 5$. **n** Alg. 1A, $n = 5$. **o** Alg. 1B, $n = 5$, $x^{0,a}$. **p** Alg. 1B, $n = 5$, $x^{0,b}$. **q** Alg. 2, $n = 5$, $x^{0,a}$. **r** Alg. 2, $n = 5$, $x^{0,b}$

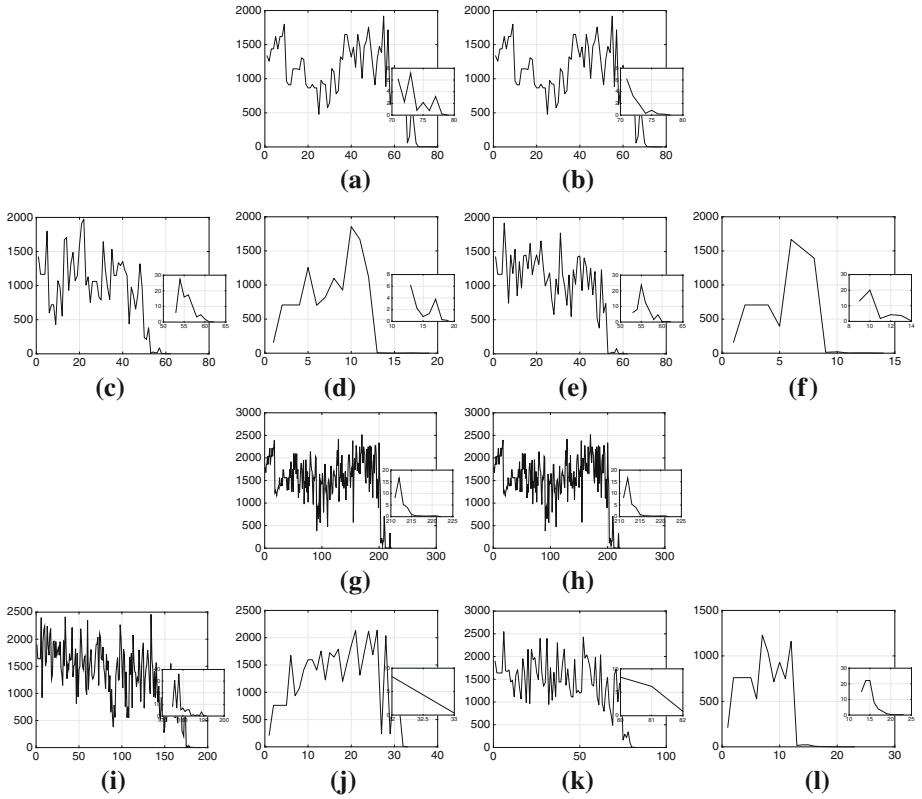

Fig. 10 Convergence histories for Algorithms 1, 1A, 1B, and 2 with two different initial points on problem (25) for $n = 3$ and 4. **a** Alg. 1, $n = 3$. **b** Alg. 1A, $n = 3$. **c** Alg. 1B, $n = 3$, $x^{0,a}$. **d** Alg. 1B, $n = 3$, $x^{0,b}$. **e** Alg. 2, $n = 3$, $x^{0,a}$. **f** Alg. 2, $n = 3$, $x^{0,b}$. **g** Alg. 1, $n = 4$. **h** Alg. 1A, $n = 4$. **i** Alg. 1B, $n = 4$, $x^{0,a}$. **j** Alg. 1B, $n = 4$, $x^{0,b}$. **k** Alg. 2, $n = 4$, $x^{0,a}$. **l** Alg. 2, $n = 4$, $x^{0,b}$

points S_E^k and support points S_U^k, thereby more rapidly exploring the interior of the feasible domain during the optimization process.

6.2 Numerical tests on problems for which the solution is on the boundary

In the previous section, it was shown that Algorithm 2 has a signicantly improved rate of convergence, as compared with Algorithm 1, when the global solution of the problem considered is on the interior the feasible domain. In this section, we consider the case in which the global solution of the problem considered is on the boundary of the feasible domain. We focus our attention on two cases, one in $n = 2$ dimensions, and one in $n = 4$ dimensions.

The first case considered is the Styblinski Tang problem (24) with $n = 2$, with the feasible domain L modified as follows:

$$-2.91 \leq x_1 \leq 5, \quad -5 \leq x_2 \leq 5. \tag{27}$$

The global solution of this problem is $x^* = [-2.91, 2.907]$, which is on the feasible domain boundary, with one active constraint.

Table 1 Application of Algorithms 1, 1A, 1B, and 2 to the Styblinski Tang test problem (24) and the Schwefel test problem (25)

Test problem	Dimension	Algorithm	# of function evaluations	# of support points
(24)	$n = 2$	1	37	N/A
		1A	37	N/A
		1B	22.25	7.25
		2	20.25	8.25
	$n = 3$	1	70	N/A
		1A	67	N/A
		1B	43.75	31.5
		2	35.25	29.5
	$n = 4$	1	149	N/A
		1A	149	N/A
		1B	80.25	83.75
		2	58	65.75
	$n = 5$	1	284	N/A
		1A	282	N/A
		1B	150.25	232.5
		2	114.25	220.75
(25)	$n = 2$	1	30	N/A
		1A	30	N/A
		1B	25.25	9.25
		2	22.25	9.75
	$n = 3$	1	78	N/A
		1A	79	N/A
		1B	34.75	60.25
		2	29.25	59.5
	$n = 4$	1	222	N/A
		1A	222	N/A
		1B	103.25	211.75
		2	83.00	202.75

For Algorithms 1B and 2, the number of function evaluations and support points averaged over 4 random initial points x^0 are reported

The second case considered is the Styblinski Tang problem (24) with $n = 4$, with the feasible domain L modified as follows:

$$-2.91 \leq x_1 \leq 5, \quad -2.91 \leq x_2 \leq 5, \quad -5 \leq x_3 \leq 5, \quad -5 \leq x_4 \leq 5. \tag{28}$$

The global solution of this problem is $x^* = [-2.91, -2.91, -2.907, -2.907]$, which is on the feasible domain boundary, with two active constraints.

The implementation of Algorithms 1 and 2 on the $n = 2$ problem constrained by (27) is shown in Fig. 11. As in the previous section, Algorithm 2 is initialized with two different initial points, defined as follows:

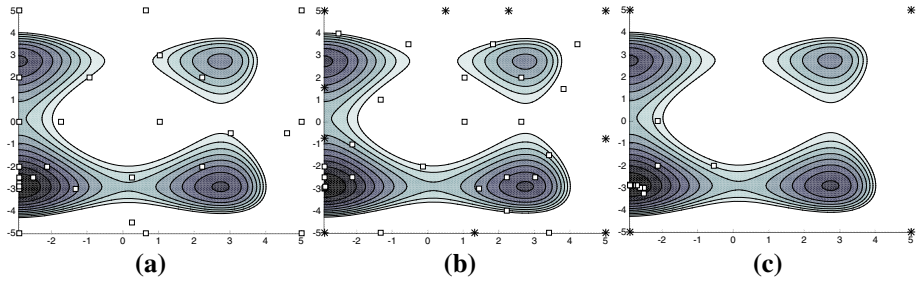

Fig. 11 Implementation of Algorithms 1 and 2 on the $n = 2$ Styblinski Tang test problem inside the feasible domain characterized by (27). **a** Algorithm 1, # of fn. evals: 25. **b** Algorithm 2, $x^{0,a}$, # of fn. evals: 26. **c** Algorithm 2, $x^{0,b}$, # of fn. evals: 12

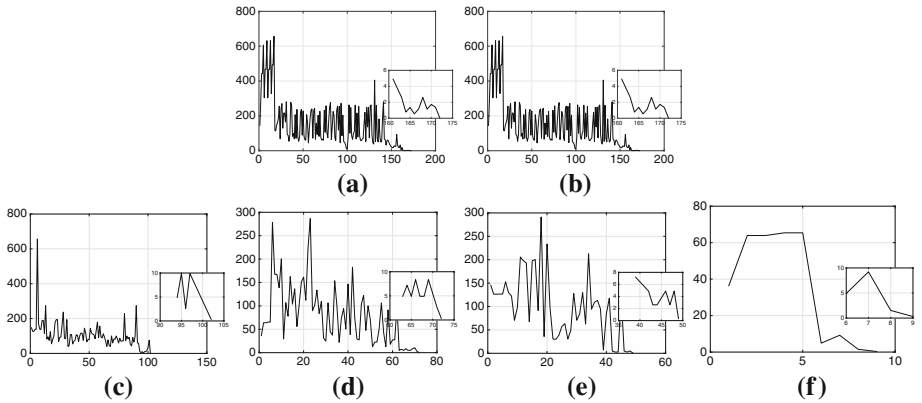

Fig. 12 Convergence history of Algorithms 1, 1A, 1B, and 2 on the $n = 4$ Styblinski Tang test problem inside the feasible domain characterized by (28). **a** Alg. 1. b Alg. 1A. **c** Alg. 1B, $x^{0,a}$. **d** Alg. 1B, $x^{0,b}$. **e** Alg. 2, $x^{0,a}$. **f** Alg. 2, $x^{0,b}$

$$x^{0,a} = [-1.328,\ -2], \quad x^{0,a} = [1.045,\ 0]. \tag{29}$$

It is observed that the number of function evaluations accumulating on those boundaries which do not include the solution is reduced in Algorithm 2, though the boundary that contains the solution is still effectively explored. Thus, again, Algorithm 2 is seen to minimize the function evaluations on the boundary which are not necessary.

The convergence history of Algorithms 1, 1A, 1B, and 2 on the $n = 4$ problem constrained by (28) are shown in Fig. 12. Again, Algorithms 1B and 2 are initialized with two different initial points, defined as follows:

$$x^{0,a} = [-1.328,\ -1.328,\ -2,\ -2], \quad x^{0,b} = [1.045,\ 1.045,\ 0,\ 0].$$

The averaged performance of the application of the Algorithm 1, 1A, 1B and 2 in the $n = 4$ case are summarized in Table 2. It is again observed that Algorithm 2 significantly outperforms Algorithm 1.

It is observed that, the performance of Algorithm 2 is better this case too. The reason of this phenomenon, is that the number of expolation of the objective function on the unnecessary boundaries are reduced, but the boundary which includes the solution is explored.

Table 2 Summary of the convergence of Algorithms 1, 1A, 1B, 2 on the $n = 4$ Styblinski Tang test problem inside the feasible domain characterized by (28)

Algorithm	# of function evaluations	# of support points
1	173	N/A
1A	173	N/A
1B	100.5	131.25
2	87.75	116.25

For Algorithms 1B and 2, the number of function evaluations and support points averaged over 4 random initial points x^0 are reported

7 Conclusions

In this paper, we have modified the original Delaunay-based derivative-free optimization algorithm Δ-DOGS, proposed in [8], in order to accumulate fewer evaluation points on the boundary of feasibility, thereby exploring the interior of the feasible domain more rapidly. The resulting algorithm, dubbed Δ-DOGS(Z), has three main modifications as compared with the original algorithm:

- Two different sets of points are considered during the optimization procession: evaluation points and support points. The latter set helps to regulate the triangulation developed.
- Since the uncertainty function is zero at some points which are not in the evaluation set, another metric for the search function is used at these points.
- The datapoints that are used in the Algorithm 2 all lie on a Cartesian grid which is successively refined as the iterations proceed.

As with our original Delaunay-based optimization algorithm, as well as any other derivative-free optimization algorithm, there is a significant curse of dimensionality, and optimization in only moderate-dimensional problems (i.e., $n \lesssim 10$) is expected to be numerically tractable. A key bottleneck of the present class of algorithms as the dimension of the problem is increased is the overhead associated with the enumeration of the Delaunay triangulation. Another limitation of the algorithm presented here is its restriction to bound-constrained domains; note that the original Delaunay-based optimization algorithm developed in [8] can handle any linearly-constrained domains. Another potential weakness is that the Cartesian grid used here is not the best option for the discretization as the dimension n is increased (for further explanation, see [7]). In future work, the algorithm developed here will be modified to deal with general linearly-constrained domains, and different lattices will be considered as alternatives to the Cartesian grid.

The optimization algorithm developed in this paper, using both polyharmonic spline interpolation as well as a new interpolation method developed by our group, dubbed Multivariate Adaptive Polyharmonic Splines (MAPS), has already been successfully applied to a challenging real-world application involving the minimization of drag on a hydrofoil [3]; the Δ-DOGS(Z) algorithm developed in the present work showed a significantly improved rate of convergence as compared with the original Δ-DOGS algorithm. Additional benchmark test problems and application-based optimization problems will be considered in future work.

Acknowledgements The authors gratefully acknowledge AFOSR FA 9550-12-1-0046 in support of this work.

Appendix: Modified algorithm for problems without target value

In this appendix, we present a modified algorithm that does not required a target value for the objective function. The algorithm developed is quite similar to Algorithm 2, with the continuous and discrete search functions modified as follows:

$$s_c^k(x) = p^k(x) - K^k\, e^k(x), \tag{30}$$

$$s_d^k(x) = p^k(x) - L^k\, \mathrm{Dis}\{x, S_E^k\}. \tag{31}$$

The parameters L^k and K^k are two positive series which are defined as follows:

$$L^k = L_0\, \ell^k,\ K^k = K_0\, \ell^k. \tag{32}$$

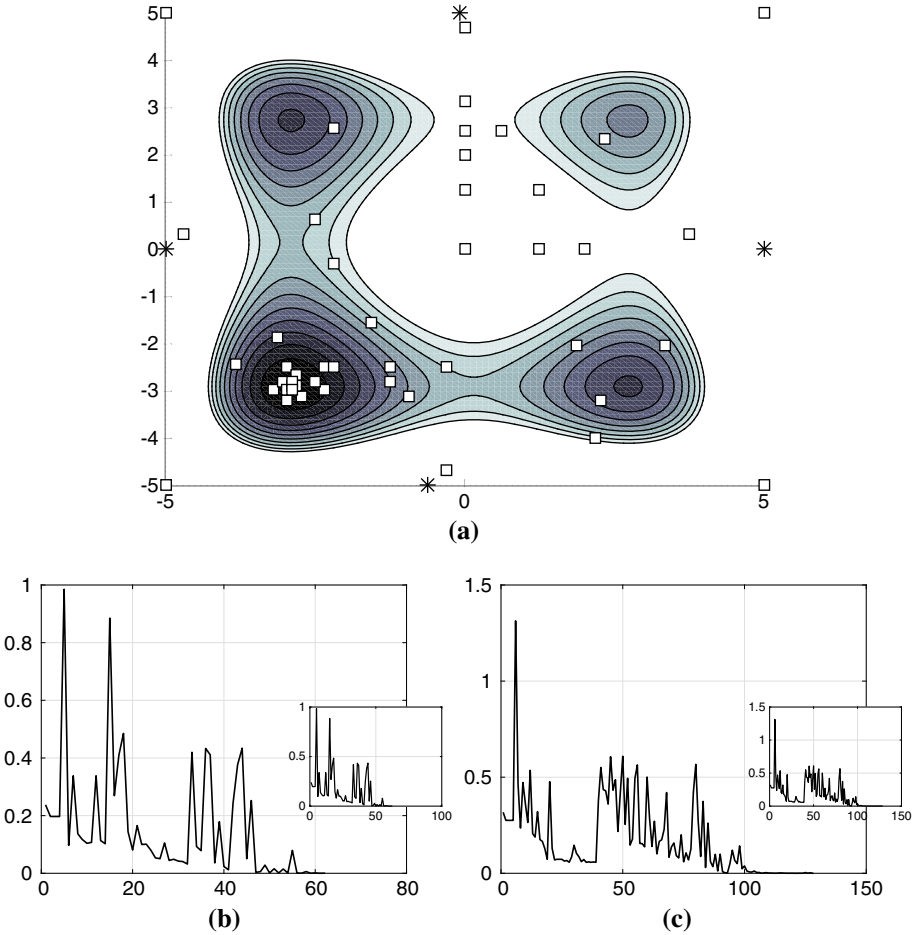

Fig. 13 Application of the modified algorithm discussed in the "Appendix", without leveraging knowledge of the target value, for minimizing the Styblinski Tang test problem (24). The position of the evaluated points (*squares*) and support points (*stars*) in the $n = 2$ case are depicted in (**a**). Convergence histories are shown in **b** for $n = 3$ and **c** for $n = 4$. **a** Location of evaluated and support points, $n = 2$. **b** Convergence history, $n = 3$. **c** Convergence history, $n = 4$

where ℓ^k is the level of the grid at step k. The convergence analysis of this modified algorithm is similar to the analysis presented in Sect. 5, with the main differences as follows:

1. Equation (14) is modified to:

$$\min\left\{s_c^k(x^*),\ \min_{z\in S_U^k}\left\{s_d^k(z)\right\}\right\} \le f(x^*). \tag{33}$$

   Note that above equation is not true for all iterations k, but it is true once

$$K^k \ge \hat{K} \quad \text{and} \quad L^k \ge \hat{L};$$

   note that the series K^k and L^k increase without bound, and thus (33) is satisfied for sufficiently large k.
2. Equation (19) is modified to

$$\min_{z\in S_E^k} f(z) - f(x^*) \le \max\left\{(L^k+\hat{L})\,\delta_k,\ (K^k+\hat{K})\,\delta_k^2\right\}. \tag{34}$$

   Moreover, we have:

$$\lim_{k\to\infty} L^k\delta_k = \lim_{k\to\infty} L_0\delta_0\frac{\ell^k}{2^{\ell^k}} = 0,$$
$$\lim_{k\to\infty} K^k\delta_k^2 = \lim_{k\to\infty} K_0\delta_0^2\frac{2^\ell}{4^\ell} = 0.$$

   As a result, the right hand size of (34) converges to zero as $k\to\infty$.

We have implemented this modified algorithm on the problem of minimizing the Styblinski Tang test problem (24), for $n = \{2, 3, 4\}$, inside the domain $-5 \le x_i \le 5\ \forall i$, with the initial point given by $x_i^0 = 0.5\ \forall i$. In these computations, the value of $K_0 = 50$ and $L_0 = 5$ were used; note that these parameter values happen to be good for this test problem. In general, selecting well these two parameters, which ultimately affect the convergence *rate* of the resulting algorithm, involves an exercise in trial and error; note, however, that (following the modified analysis described above) convergence is proved for this modification of Algorithm 2 for any choice of K_0 and L_0. An analogous issue was encountered when selecting K in Algorithm 1 of [8]. Figure 13 shows the positions of the function evaluations and support points for the $n = 2$ case, and the convergence histories for the $n = 3$ and $n = 4$ cases. The convergence of the modified algorithm proposed here is, again, seen to be quite rapid.

References

1. Abramson, M.A., Audet, C., Dennis, J.E., Le Digabel, S.: OrthoMADS: a deterministic MADS instance with orthogonal directions. SIAM J. Optim. **20**(2), 948–966 (2009)
2. Alimohammadi, Shahrouz., He, Dawei.: Multi-stage algorithm for uncertainty analysis of solar power forecasting. In: Power and Energy Society General Meeting (PESGM), 2016, pp. 1-5. IEEE (2016)
3. Alimohammadi, S., Beyhaghi, P., Bewley, T.: Delaunay-based optimization in CFD leveraging multivariate adaptive polyharmonic splines (MAPS). In: 58th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference (2017)
4. Audet, C., Dennis, J.E.: A pattern search filter method for nonlinear programming without derivatives. SIAM J. Optim. **14**(4), 980–1010 (2004)
5. Audet, C., Dennis, J.E.: Mesh adaptive direct search algorithms for constrained optimization. SIAM J. Optim. **17**(1), 188–217 (2006)

6. Audet, C., Dennis, J.E.: A progressive barrier for derivative-free nonlinear programming. SIAM J. Optim. **20**(1), 445–472 (2009)
7. Belitz, Paul, Bewley, Thomas: New horizons in sphere-packing theory, part II: lattice-based derivative-free optimization via global surrogates. J. Glob. Optim. **56**(1), 61–91 (2013)
8. Beyhaghi, P., Cavaglieri, D., Bewley, T.: Delaunay-based derivative-free optimization via global surrogates, part I: linear constraints. J. Glob. Optim. **63**, 1–52 (2015)
9. Beyhaghi, P., Bewley, T.: Delaunay-based Derivative-free optimization via global surrogates, part II: convex constraints. J. Glob. Optim. **2016**, 1–33 (2016)
10. Booker, A.J., Deniss, J.E., Frank, P.D., Serafini, D.B., Torczon, V., Trosset, M.W.: A Rigorous Framework for Optimization of Expensive Function by Surrogates. Springer-Verlag, Berlin (1999)
11. Galperin, E.A.: The cubic algorithm. J. Math. Anal. Appl. **112**, 635–640 (1985)
12. Gill, P.E., Murray, W., Wright, M.H.: Practical optimization, pp. 99–104. Academic Press, London (1981)
13. Jones, D.R., Perttunen, Cary D., Stuckman, Bruce E.: Lipschitzian optimization without the Lipschitz constant. J. Optim. Theory Appl. **79**(1), 157–181 (1993)
14. Jones, D.R.: A taxonomy of global optimization methods based on response surfaces. J. Glob. Optim. **21**, 345–383 (2001)
15. Lewis, R.M., Torczon, V., Trosset, M.W.: Direct Search Method: Then and Now, NASA/CR-2000-210125, ICASE Report No.2000-26 (2000)
16. Wright, S., Nocedal, J.: Numerical Optimization. Springer, Berlin (1999)
17. Paulavicius, P., Zilinskas, J.: Simplical Optimization. Springer, Berlin (2014)
18. Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning. MIT Press, Cambridge (2006)
19. Schonlau, M., Welch, W.J., Jones, D.J.: A Data-Analytic Approach to Bayesian Global Optimization, Department of Statistics and Actuarial Science and The Institute for Improvement in Quality and Productivity, 1997 ASA Conference (1997)
20. Shubert, B.O.: A sequential method seeking the global maximum of a function. SIAM J. Numer. Anal. **9**(3), 379–388 (1972)
21. Torczon, V.: Multi-Directional Search, A Direct Search Algorithm for Parallel Machines. Ph.D. thesis, Rice University, Houston, TX (1989)
22. Torczon, V.: On the convergence of pattern search algorithms. SIAM J. Optim. **7**(1), 1–25 (1997)