J Glob Optim (2016) 66:331–382
DOI 10.1007/s10898-015-0384-2

Delaunay-based derivative-free optimization via global surrogates, part I: linear constraints

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

Received: 31 March 2015 / Accepted: 4 November 2015 / Published online: 13 November 2015

Abstract A new derivative-free optimization algorithm is introduced for nonconvex functions within a feasible domain bounded by linear constraints. Global convergence is guaranteed for twice differentiable functions with bounded Hessian, and is found to be remarkably efficient even for many functions which are not differentiable. Like other Response Surface Methods, at each optimization step, the algorithm minimizes a metric combining an interpolation of existing function evaluations and a model of the uncertainty of this interpolation. By adjusting the respective weighting of these two terms, the algorithm incorporates a tunable balance between global exploration and local refinement; a rule to adjust this balance automatically is also presented. Unlike other methods, any well-behaved interpolation strategy may be used. The uncertainty model is built upon the framework of a Delaunay triangulation of existing datapoints in parameter space. A quadratic function which goes to zero at each datapoint is formed within each simplex of this triangulation; the union of each of these quadratics forms the desired uncertainty model. Care is taken to ensure that function evaluations are performed at points that are *well situated* in parameter space; that is, such that the simplices of the resulting triangulation have circumradii with a known bound. This facilitates well-behaved local refinement as additional function evaluations are performed.

Keywords Derivative-free optimization · Surrogate functions · Delaunay triangulation · Linear constraints

✉ Pooriya Beyhaghi
pbeyhagh@ucsd.edu

Daniele Cavaglieri
dcavagli@ucsd.edu

Thomas Bewley
bewley@ucsd.edu

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

1 Introduction

In this paper, a new derivative-free optimization algorithm is presented to minimize a (possibly nonconvex) function subject to linear constraints on a bounded feasible region in parameter space[1]:

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

where $x \in \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and L is assumed to be bounded. A special case of this problem, with simpler "box" constraints, is also considered:

$$\text{minimize } f(x) \text{ with } x \in L_{\text{box}} = \{x | a \leq x \leq b\}, \tag{1b}$$

where $a, b \in \mathbb{R}^n$. The algorithm developed here is extended in Part II of this study to handle more general convex constraints on the feasible region of parameter space. Derivative-free algorithms are well suited for such problems even if neither the derivative of $f(x)$ nor its accurate numerical approximation is readily available, as is the case when $f(x)$ is nonsmooth. This is common in situations in which the function $f(x)$ is derived either from an experiment or from many types of numerical simulations.

An important class of derivative-free algorithms, dating back to the 1960s, is Direct Search Methods, as reviewed in [19]. An early and famous algorithm in this class is the Nelder–Mead simplex algorithm, variations of which are implemented in several numerical optimization packages. This method is examined in, e.g., [33]. Another category of Direct Search Methods, called Adaptive Direction Search Algorithms, includes the Rosenbrock [29] and Powell [28] methods. More modern methods in this class, dubbed Pattern Search Methods, are characterized by a series of exploratory moves on a regular pattern of points in parameter space called a lattice (often, the Cartesian grid is used); the Generalized Pattern Search (GPS) is a typical example. The efficiency and convergence of such algorithms is examined in [34] and [37].

In general, Direct Search Methods identify a local minimum of a function from some initial guess in parameter space. The harder problem 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.

Response Surface Methods employ an underlying (inexpensive-to-compute, differentiable) model of the actual (expensive-to-compute, possibly nondifferentiable) function of interest in order to summarize the trends evident in the available datapoints, at which the function has already been evaluated, over the entire feasible region in parameter space as the iteration proceeds. The Trust Region method is one of the first optimization algorithms appearing in the literature which uses such a model; however, the model used by this method does not use all of the available datapoints at each step. Other Response Surface Methods, such as the Expected Improvement algorithm [31], typically use all available datapoints to build an inexpensive and useful model (often called a "surrogate") of the actual function of interest. An insightful review of global optimization methods based on such surrogate functions is given by [17].

The most popular surrogate function used in such global optimization schemes is the Kriging method [18,24,30], 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. With this interpolation strategy, the function is modeled as a Gaussian random variable at every point within the feasible domain of parameter space. This

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

stochastic model is constructed carefully, such that the variance of the random variable is zero, and the expected value of the random variable is equal to the (known) function value at each datapoint available in parameter space. Away from the datapoints, the expected value of the random variable in the Kriging model effectively interpolates the known function values, and the variance of the random variable is greater than zero, effectively quantifying the distance in parameter space to the nearest available datapoints. As eloquently described in [17], the estimate $p(x)$ and the uncertainty of the estimate, $e(x)$, provided by this model may be used together to identify a point in the feasible domain with a high probability of a reduced function value. A particularly efficient algorithm for global optimization is the Surrogate Management Framework (SMF; see [4]), which combines the Expected Improvement algorithm with a Generalized Pattern Search. This algorithm was significantly extended in [3], in which the search is coordinated by a lattice derived from a dense sphere packing, with significantly improved uniformity of grid points over parameter space as compared with the Cartesian grid, in order to accelerate convergence.

The Kriging interpolation strategy has 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 there are many datapoints available, some of which are clustered in a small region of parameter space, as illustrated in "Appendix". Furthermore, both the computation of the Kriging interpolant itself, as well as the minimization over the feasible region of parameter space of the search function based on this interpolant, are nonconvex optimization problems; both of these problems must be solved with another global optimization algorithm, which represents a sometimes significant computational expense.

As discussed above, modern Response Surface Methods need both an estimate of the function itself as well as a model of the uncertainty of this estimate over the entire feasible domain of parameter space. Most interpolation methods, other than Kriging, don't provide this. For the specific case of interpolation with radial basis functions, an uncertainty function has been proposed and used by [14].

The Response Surface Method proposed in this work is innovative in the way it facilitates the use of *any* well behaved interpolation strategy that the user might favor for the particular problem under consideration. [In the present work, our numerical examples use polyharmonic spline interpolation, which is reasonably well behaved even when the available datapoints are clustered in various regions of parameter space; this interpolation strategy is fairly standard, though other interpolation schemes could easily be used in its place.] To accomplish this, the present work proposes an artificially-generated function modeling the "uncertainty" of the interpolant based on the distance to the nearest datapoints. This uncertainty model is built directly on the framework of a Delaunay triangulation of the available datapoints in parameter space.

The structure of the paper is as follows. Section 2 discusses how the present algorithm may be initialized. Section 3 then proposes a simple strawman form of the algorithm based on the present ideas, laying out the essential elements of the final algorithm and analyzing its various properties, including a proof of convergence under the conditions that (a) the underlying function of interest $f(x)$ has bounded Lipschitz norm, and (b) the maximum circumradii of the simplicies in the triangulations are bounded as the algorithm proceeds. This simple strawman form of the optimization algorithm, however, fails to ensure condition (b). Section 4 modifies the strawman form of the optimization algorithm proposed previously by, when necessary, adjusting the points in parameter space at which new function evaluations are performed, thereby ensuring condition (b). Section 5 presents a rule to adjust the parameter which tunes the balance between global exploration and local refinement as the iteration

proceeds. Section 6 addresses how the algorithm may be modified to run efficiently using parallel computations. In Sect. 7, the algorithm proposed is applied to a select number of test functions in order to illustrate its behavior. Some conclusions are presented in Sect. 8.

2 Initialization

The optimization algorithm developed in this paper is initialized as follows:

Algorithm 1 *(A) perform function evaluations at all of the vertices of the feasible domain L,*
(B) remove all redundant constraints from the rows of $Ax \leq b$, and
(C) project out any equality constraints implied by multiple rows of $Ax \leq b$; in other words, we project the feasible domain onto the lower dimensional space that satisfies the equality constraints.

This algorithm will be described in detail in the remainder of Sect. 2. The optimization algorithm developed in later sections then builds a Delaunay triangulation within the convex hull of the available function evaluations, which coincides with the feasible domain itself, and incrementally updates this Delaunay triangulation at each new datapoint (that is, at each new feasible point $x \in L$ at which $f(x)$ is computed as the iteration proceeds). This approach is justified by the following result, which is proved in [1]:

Theorem 1 *The convex hull of the vertices of a bounded domain L constrained such that $Ax \leq b$ is equivalent to the domain L itself.*

Due to the simplicity of step (A) of Algorithm 1, this step is recommended for most low-dimensional problems. In high-dimensional problems bounded by many linear constraints, however, the feasible domain might have a lot of vertices, and it might be unnecessarily expensive to follow such an approach; in such cases, Part II of this work demonstrates how this initialization step may be cleverly sidestepped.

In the case of box constraints, (1b), step (A) of Algorithm 1 corresponds to 2^n function evaluations which are trivial to enumerate.

In the more general case of linear constraints, (1a), identifying the vertices of the feasible domain is slightly more involved. We proceed as follows:

Definition 1 The **active set** of the constraints $Ax \leq b$ at a given point $\hat{x} \in R^n$ in parameter space, denoted $A_{a(\hat{x})}\,\hat{x} = b_{a(\hat{x})}$, is given by those constraints (that is, by those rows of $Ax \leq b$) that hold with equality at $\hat{x}$. A **feasible point** $\hat{x}$ (satisfying $A\hat{x} \leq b$) is called a **vertex** of the **feasible domain** (that is, the set of all $x \in \mathbb{R}^n$ such that $Ax \leq b$) if $rank(A_{a(\hat{x})}) = n$.

A simple brute-force method to find all of the vertices of the feasible domain then follows:

(1) Check the rank of all $\binom{m}{n}$ $n \times n$ linear systems that may be chosen from the $m > n$ rows of $Ax \leq b$.
(2) For those linear systems in step 1 that have rank n, solve $A_{a(\hat{x})}\,\hat{x} = b_{a(\hat{x})}$.
(3) For each solution found in step 2, check to see if $A\hat{x} \leq b$; if this condition holds, it is a vertex.

The set of points thus generated is then scrutinized to eliminate duplicates. This brute-force method is tractable only in relatively low-dimensional problems (note that most problems that

are viable candidates for derivative-free optimization are, in fact, fairly low-dimensional). The number of vertices is typically much less than the number of linear systems considered by this method; for example, $m = 20$ constraints in $n = 10$ dimensions requires us to examine $184,756$ $n \times n$ matrices in step 1, and would typically result in roughly $M \sim O(10^3)$ vertices.

Finding the M vertices of an n-dimensional polyhedron is a well-known problem in convex analysis; see, e.g., [2,22] and [23]. These papers suggest a somewhat more involved yet significantly more computationally efficient iterative procedure, based on the simplex method, to find the vertices of the feasible domain in problems that are high-dimensional and/or have many linear constraints. With this approach, a pivot operation is used to move from one vertex of the feasible domain to its neighbors (the vertex v_1 and v_2 are called neighbors if their active sets differ in exactly one row). The number of linear solves required by this approach is $O(nM)$.

Step (B) of Algorithm 1 then removes all redundant constraints given by the redundant rows of $Ax \leq b$. Each row of $Ax \leq b$ is checked at each vertex of the feasible domain. Those rows that are not satisfied as equalities at at least n distinct vertices are eliminated, as they do not play a role in defining an $(n-1)$-dimensional face of the feasible domain. Of the rows that remain, the rows of the augmented matrix $\left[A\ b\right]$ that are multiples of other rows are also eliminated, as they define identical faces.

Finally, step (C) of Algorithm 1 projects out all equality constraints in the problem formulation, as algorithms for the construction of an n-dimensional Delaunay triangulation will encounter various problems if the feasible domain actually has dimension less than n. In the case of (1a), equality constraints may easily be found and projected out, resulting in a lower-dimensional optimization problem. To illustrate, consider $\{x_1, x_2, \ldots, x_M\}$ as the set of vertices of the feasible domain of x, computed as described above. Define the $n \times (M-1)$ matrix C as follows:

$$C = \left[(x_1 - x_2)\ (x_1 - x_3)\ \cdots\ (x_1 - x_M)\right]. \tag{2}$$

The rank r of the matrix C is the rank of the optimization problem at hand. If $r < n$, there are one or more equality constraints to contend with. In this case, taking the reduced $\underline{Q}R$ decomposition $C = \underline{Q}R$, the r linearly-independent columns of $\underline{Q}$ provide a new basis in which the optimization problem may be written. Defining $x = x_1 + \underline{Q}\,\underline{x}$, a new r-dimensional optimization problem is posed in the space of $\underline{x} \in R^r$, and the feasible domain of $\underline{x}$ is defined by $(A\underline{Q})\underline{x} \leq b - Ax_1$.

3 Strawman form of algorithm

Algorithm 2 *Prepare the problem for optimization by executing Algorithm 1, as described in Sect. 2. Assume that the resulting optimization problem is n dimensional, and that the feasible domain L has M vertices. Then, proceed as follows:*

0. *Take the set of initialization points S^0 as all M of the vertices of the feasible domain L together with one or more user-specified points of interest on the interior of L. Evaluate the function $f(x)$ at each of these initialization points. Set $k = 0$.*
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.*[2]
3. *For each simplex Δ_i^k of the triangulation Δ^k:*
   a. *Calculate the circumcenter z_i^k and the circumradius r_i^k of the simplex Δ_i^k.*
   b. *Define the* **local uncertainty function**
   $$e_i^k(x) = (r_i^k)^2 - \|x - z_i^k\|^2. \tag{3}$$
   c. *Define the* **local search function**
   $$s_i^k(x) = p^k(x) - K\, e_i^k(x). \tag{4}$$
   d. *Minimize the local search function $s_i^k(x)$ within Δ_i^k.*
4. *Find the smallest of all of the local minima identified in step 3d. Evaluate $f(x)$ at this new datapoint x_k, and set $S^{k+1} = S^k \cup \{x_k\}$. Increment k and repeat from step 1 until convergence.*

The local uncertainty functions $e_i^k(x)$ model the uncertainty in the unexplored regions within each simplex of the triangulation Δ^k at step k. As discussed in Sect. 3.1, the union of these simplices coincides precisely with feasible domain of parameter space. The **global uncertainty function** $e^k(x)$ and the **global search function** $s^k(x)$ are defined over the feasible domain as $e_i^k(x)$ and $s_i^k(x)$, respectively, within each simplex Δ_i^k. Note that $e^k(x)$ reaches zero by construction at each datapoint, and $e^k(x)$ reaches a maximum within each simplex as far from all of the available datapoints as possible; it is shown in Sect. 3.2 that $e^k(x)$ is Lipschitz. In Sect. 3.3, a method of simplifying the searches performed in step 3d of Algorithm 2 is discussed.

The (single, constant) tuning parameter K specifies the trade-off in Algorithm 2 between global exploration (which is emphasized for large K) and local refinement (which is emphasized for small K). In Sect. 3.4, global convergence of Algorithm 2 is proved for functions $f(x)$ with bounded Lipschitz norm, assuming sufficiently large K and boundedness of the circumradii of the triangulation generated by Algorithm 2. In Sect. 4, a small but technically important modification of the Algorithm 2 is introduced which guarantees boundedness of the circumradii of the triangulation generated as the iteration proceeds.

3.1 Characterizing the triangulation

The uncertainty function in Algorithm 2 is built on the framework of a Delaunay triangulation of the feasible domain with, in a certain sense, maximally regular simplices, which we now characterize.

Definition 2 Consider the $(n + 1)$ vertices $V_0, V_1, \ldots, V_n \in \mathbb{R}^n$ such that the vectors $(V_0 - V_1), (V_0 - V_2), \ldots, (V_0 - V_n)$ are linearly independent. The convex hull of these vertices is called a **simplex** (see, e.g., [5, p. 32]). Associated with this simplex, the **circumcenter** z is the point that is equidistant from all $n + 1$ vertices, the **circumradius** r is the distance between z and any of the vertices V_i, and the **circumsphere** is the set of all points within a distance r from z.

Lemma 1 *For any simplex, the circumcenter is unique.*

[2] Delaunay triangulations always exist, but are not necessarily unique. This algorithm builds on a Delaunay triangulation at each step, even if it is not unique. If a different Delaunay triangulation is used at a given step k, a different point x_k will be found, but the convergence properties are unaffected.

Proof Assume z is equidistant from $V_0, \ldots, V_n$, i.e.

$$\|V_0 - z\| = \|V_1 - z\| = \cdots = \|V_n - z\|$$

For $i = 1, \ldots, n$, simplification leads to:

$$\begin{aligned} & V_0^2 - 2V_0^T z = V_i^2 - 2V_i^T z \\ \Rightarrow \quad & 2\,(V_0 - V_i)^T z = V_0^2 - V_i^2. \end{aligned}$$

Thus, z is equidistant from all vertices if

$$2 \begin{bmatrix} (V_0 - V_1)^T \\ \vdots \\ (V_0 - V_n)^T \end{bmatrix} z = \begin{bmatrix} V_0^2 - V_1^2 \\ \vdots \\ V_0^2 - V_n^2 \end{bmatrix}. \tag{5}$$

This system has a unique solution if the matrix on the LHS is nonsingular; which follows from the linear independence of $(V_0 - V_1), (V_0 - V_2), \ldots, (V_0 - V_n)$ in Definition 2. □

The two following definitions are taken from [12].

Definition 3 If S is a set of points in $\mathbb{R}^n$, a **triangulation** of S is a set of simplices whose vertices are elements of S such that the following conditions hold:

- Every point in S is a vertex of at least one simplex in the triangulation. The union of all of these simplices fully covers the convex hull of S.
- The intersection of two different simplices in the triangulation is either empty or a k-simplex such that $k = 0, 1, \ldots, n-1$. For example, in the case of $n = 3$ dimensions, the intersection of two simplices (in this case, tetrahedra) must be an empty set, a vertex, an edge, or a triangle.

Definition 4 A **Delaunay triangulation** is a triangulation (see Definition 3) such that the intersection of the open circumsphere around each simplex with S is empty. This special class of triangulation, as compared with other triangulations, has the following properties:

- The maximum circumradius among the simplices is minimized.
- The sum of the squares of the edge lengths weighted by the sum of the volumes of the elements sharing these edges is minimized.

Delaunay triangulations exhibit an additional property which makes them essential in Algorithm 2. By the definitions of $e_i^k(x)$ and $e^k(x)$ above, it follows that $e_i^k(x) = e^k(x)$ within the simplex Δ_i^k. The following may be established *if* the triangulation Δ^k is Delaunay:

Lemma 2 *Assume the triangulation Δ^k is Delaunay. For any i and any feasible point $x \in L$, $e^k(x) \geq e_i^k(x)$.*

Proof By Theorem 1 and Definition 3, since $x \in L$, a simplex Δ_j^k exists which contains x (that is, $x \in \Delta_j^k$). We must show that, for all $i \neq j$, $e_i^k(x) \leq e_j^k(x)$. By construction, $e_j^k(x) = 0$ at the vertices of simplex Δ_j^k; since the triangulation is Delaunay (see Definition 4), these vertices are not inside the circumsphere of the simplex Δ_i^k. Thus, $e_i^k(x) \leq 0$ at the vertices of simplex Δ_j^k. It follows simply from the definition of $e_i^k(x)$ that, for all $x \in L$,

$$e_i^k(x) - e_j^k(x) = (r_i^k)^2 - (r_j^k)^2 - |z_i^k|^2 + |z_j^k|^2 + 2(z_i^k - z_j^k)^T x;$$

that is, $e_i^k(x) - e_j^k(x)$ is a linear function of x. Since $e_i^k(x) - e_j^k(x) \leq 0$ at the vertices of simplex Δ_j^k, it follows that $e_i^k(x) - e_j^k(x) \leq 0$ everywhere within simplex Δ_j^k. □

Remark 1 Lemma 2 holds only for Delaunay triangulations, not arbitrary triangulations. Lemma 2 is used in Sect. 3.3 to simplify the searches performed in step 3d of Algorithm 2.

Remark 2 In step 3a of Algorithm 2, linear systems of the form given in (5) must be solved in order to find the circumcenter of each simplex. The use of Delauney triangulations improves the accuracy of these numerical solutions. If the ratio between the circumradius and the maximum distance between two edges of a simplex is large, this system is ill conditioned. Delaunay triangulations (see Definition 4) minimize the maximum circumradius of the simplices in the triangulation, thereby minimizing the worst-case ill conditioning of the linear systems of the form given in (5) that need to be solved.

The determination of Delaunay triangulations is a benchmark problem in computational geometry, and a large number of algorithms have been proposed; extensive reviews are given in [9] and [12]. Qhull (used by Matlab and Mathematica, see [39]), Hull (see [40]), and CGAL-DT (see [41]) are among the most commonly-used approaches today for computing Delaunay triangulations in moderate dimensions. In the present work, a Delaunay triangulation must be performed over a set of initial evaluation points, then updated at each iteration when a new datapoint is added. Hence, the incremental method originally proposed in [36] is particularly appealing. The New-DT and Del-graph algorithms (see [6] and [7]) are the leading, memory-efficient implementations of this incremental approach; the present work implements the Del-graph algorithm.

The most expensive step of Algorithm 2, apart from the function evaluations, is the minimization of $s_j^k(x)$ (in step 3d) in each simplex Δ_j^k. The cost of this step is proportional to the total number of simplices S in the Delaunay triangulation. As derived in [25], a worst-case upper bound for the number of simplices in a Delaunay triangulation is $S \sim O(N^{\frac{n}{2}})$, where N is the number of vertices and n is the dimension of the problem. As shown in [10] and [11], for vertices with a uniform random distribution, the number of simplices is $S \sim O(N)$.

3.2 Smoothness of the uncertainty

We now characterize precisely the smoothness of the uncertainty function proposed in Algorithm 2.

Lemma 3 *The function $e^k(x)$ is C_0 continuous.*

Proof Consider a point x on the boundary between two different simplices Δ_i^k and Δ_j^k with circumcenters z_i^k and z_j^k and local uncertainty functions $e_i^k(x)$ and $e_j^k(x)$. By Definition 3, the intersection of Δ_i^k and Δ_j^k, when it is nonempty, is another simplex of lower dimension, denoted here simply as Δ. The projection of z_i^k and z_j^k on the lower-dimensional hyperplane that contains Δ is by construction its circumcenter, denoted here as z. Thus, the lines from z_i^k to z and from z_j^k to z are perpendicular to the simplex Δ. Now consider x_Δ as one of the vertices of the simplex Δ. Some trivial analysis of the triangles $z_i^k - x - z$ and $z_i^k - x_\Delta - z$ give:

$$\begin{aligned} e_i^k(x) &= \|z_i^k - x_\Delta\|^2 - \|z_i^k - x\|^2, \\ \|z_i^k - x_\Delta\|^2 &= \|z_i^k - z\|^2 + \|z - x_\Delta\|^2, \\ \|z_i^k - x\|^2 &= \|z_i^k - z\|^2 + \|z - x\|^2. \end{aligned}$$

Combining these three equations gives

$$e_i^k(x) = \|z - x_\Delta\|^2 - \|z - x\|^2.$$

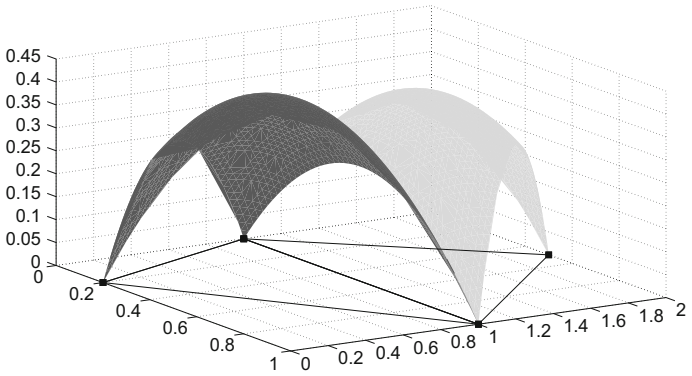

Fig. 1 The uncertainty function $e^k(x)$ over two neighboring simplices in two dimensions

By similar reasoning, we obtain

$$e_j^k(x) = \|z - x_\Delta\|^2 - \|z - x\|^2.$$

Hence, $e_i^k(x) = e_j^k(x)$ for all $x \in \Delta$ (that is, at the interface of simplices Δ_i^k and Δ_j^k). □

The continuity of $e^k(x)$ is illustrated in 2D in Fig. 1, where two neighboring simplices (triangles) with vertices $\{(0.2, 0), (0, 1), (1, 1)\}$ and $\{(0, 1), (1, 1), (0.5, 2)\}$ are represented.

We now establish a stronger property, that the uncertainty function is Lipschitz.

Lemma 4 *The function $e^k(x)$ generated by Algorithm 2 at the kth iteration is Lipschitz within the convex polyhedron L, with a Lipschitz constant of r_{max}^k, where r_{max}^k is the maximum circumradius of the triangulation Δ^k.*

Proof We first show that $e^k(x)$ is Lipschitz inside each simplex; we then show that $e^k(x)$ is Lipschitz everywhere.

Assume first that x_1 and x_2 are inside the simplex Δ_i^k, with circumcenter z_i^k and circumradius r_i^k. By (3), we have

$$e^k(x_1) - e^k(x_2) = \|x_2 - z_i^k\|^2 - \|x_1 - z_i^k\|^2.$$

Now, assume that z^* is a projection of z_i^k along the line from x_1 to x_2, and that x_M is the midpoint between x_1 and x_2. It follows that

$$\begin{aligned} |\,\|x_2 - z_i^k\|^2 - \|x_1 - z_i^k\|^2\,| &= |\,\|x_2 - z^*\|^2 - \|x_1 - z^*\|^2\,| \\ &= 2\,\|z^* - x_M\|\,\|x_1 - x_2\|. \end{aligned}$$

Since

$$\|z^* - x_M\| \le \max\left(\|z^* - x_1\|,\ \|z^* - x_2\|\right) \le r_i^k,$$

we have

$$|e^k(x_1) - e^k(x_2)| \le 2\,r_i^k\,\|x_1 - x_2\|. \tag{6}$$

Thus, the uncertainty function $e^k(x)$ is Lipschitz inside each simplex.

In order to prove that $e^k(x)$ is Lipschitz over the entire feasible domain L, consider now x_1 and x_2 as two arbitrary points inside L, and define a series of points $t_1,\ t_2,\ \ldots,\ t_m$ on the line segment between x_1 to x_2 such that $t_1 = x_1$ and $t_m = x_2$, and such that each couple (t_i, t_{i+1}) lies within the same simplex, with circumradius r_i. In other words, each point t_i for $1 < i < m$ must be at the interface between two neighboring simplices along the line from $t_1 = x_1$ to $t_m = x_2$. In this framework, we have

$$|e^k(x_1) - e^k(x_2)| \leq \sum_{i=1}^{m-1} |e^k(t_i) - e^k(t_{i+1})|.$$

Since t_i and t_{i+1} are in the same simplex, (6) gives

$$|e^k(t_i) - e^k(t_{i+1})| \leq 2\, r_i^k\, \|t_i - t_{i+1}\|.$$

Since $t_1,\ t_2,\ \ldots,\ t_m$ lie along the same line, we have

$$\|t_1 - t_m\| = \sum_{i=1}^{m-1} \|t_i - t_{i+1}\|.$$

Combining these three equations, we have

$$\begin{aligned} |e^k(x_1) - e^k(x_2)| &\leq 2 \max_{1\leq i\leq m-1} (r_i^k)\, \|x_1 - x_2\| \\ &\leq 2\, r_{\max}^k\, \|x_1 - x_2\|. \end{aligned}$$

□

3.3 Minimizing the search function

At iteration k of Algorithm 2, the search function $s^k(x) = p^k(x) - K e^k(x)$ must be minimized over $x \in L$. Recall that, within each simplex Δ_i^k in the triangulation, the uncertainty function $e^k(x)$ is defined by $s_i^k(x) = p^k(x) - K e_i^k(x)$ for $x \in \Delta_i^k$. In order to minimize $s^k(x)$ over the entire feasible domain L, the minima $x_{\min,i}^k$ must first be found within each simplex Δ_i^k as follows:

$$x_{\min,i}^k = \operatorname{argmin}_{x\in\Delta_i^k}\ s_i^k(x); \tag{7a}$$

the global minimum $x_{\min}^k = \operatorname{argmin}_{x\in L} s^k(x)$ is then given by $x_{\min}^k = x_{\min,i_{\min}}^k$ where

$$i_{\min} = \operatorname{argmin}_{i\in\{1,\ldots,S^k\}} \left[s_i^k \left(x_{\min,i}^k \right) \right], \tag{7b}$$

where S^k is the number of simplices in the triangulation Δ^k. In other words, in order to find $x_{\min}^k$, we must first solve S^k nonconvex optimization problems with linear constraints $x \in \Delta_i^k$. This computational task is significantly simplified by following result.

Lemma 5 *If the linear constraints $x \in \Delta_i^k$ in the optimization problems defined in (7a) are relaxed to the entire feasible domain, $x \in L$, the resulting value of x_{min}^k remains unchanged.*

Proof By Lemma 2, for any feasible point $x \in L$ and for any i such that $1 \leq i \leq S^k$, $e^k(x) \geq e_i^k(x)$. More precisely,

$$e^k(x) = \max_{i\in\{1,\ldots,S^k\}} \left[e_i^k(x) \right].$$

Since K is a positive real number,

$$s^k(x) = p^k(x) - K\, e^k(x) = \min_{i\in\{1,\ldots,S^k\}} \Big[p^k(x) - K\, e_i^k(x)\Big].$$

By the definition of $x^k_{\min}$

$$s^k(x^k_{\min}) = \min_{x\in L} \Big[s^k(x)\Big].$$

Combining the above equations and swapping the order of the minimization gives

$$\begin{aligned} s^k(x^k_{\min}) &= \min_{x\in L} \min_{i\in\{1,\ldots,S^k\}} \Big[p(x) - K\, e_i^k(x)\Big] \\ &= \min_{i\in\{1,\ldots,S^k\}} \min_{x\in L} \Big[p(x) - K\, e_i^k(x)\Big]. \end{aligned}$$

It can be observed that $\min_{x\in L}[p(x) - K\, e_i^k(x)]$ is just the optimization problem (7a) with the linear constraint $x \in \Delta_i^k$ relaxed to $x \in L$; thus, when this constraint is relaxed in this manner in (7), the resulting value of $s^k(x^k_{\min})$ remains unchanged. □

At each iteration k, we thus seek to minimize the local search function $s_i^k(x)$ for $x \in L$ for each $i \in \{1, \ldots, S^k\}$; if for a given i the minimizer of $s_i^k(x)$ lies outside of Δ_i^k, that minimizer is not the global minimum of $s^k(x)$. Hence, we may terminate and reject any such search as soon as it is seen that the minimizer of $s_i^k(x)$ lies outside of Δ_i^k.

Using a local optimization method with a good initial guess within each simplex, the local minimum of $s_i^k(x)$ within the simplex Δ_i^k, if it exists, can be found relatively quickly. For the smooth local search functions $s_i^k(x)$ we use in the present work, we have analytic expressions for both the gradient and the Hessian; these expressions play a valuable role in the local minimization of these functions.

Recall that the local search functions $s_i^k(x)$ considered in Algorithm 2 are linear combinations of the local uncertainty functions $e_i^k(x)$ and the user's interpolation function of choice, $p^k(x)$. The local uncertainty function $e_i^k(x)$ is a quadratic function whose gradient and Hessian are

$$\nabla e_i^k(x) = -2\,(x - x_{s_i}), \qquad \nabla^2 e_i^k(x) = -2\, I.$$

For the polyharmonic spline interpolation method used in the present numerical implementation, analytical expressions for the gradient and Hessian of the interpolation function are derived in the appendix. If a different interpolation strategy is used, for the purpose of the following discussion, we assume that analytical expressions for the gradient and the Hessian of the interpolation function are similarly available.

In order to locally minimize the function $s_i^k(x)$ within each simplex, a good initial guess of the solution is valuable. To generate such an initial guess analytically, consider the result of a simplified optimization problem obtained by implementing piecewise linear interpolation of the datapoints at the vertices of the simplex Δ_i^k, together with the local uncertainty function $e_i^k(x)$. Following this approach, we rewrite the coordinates of a point inside the simplex Δ_i^k as a linear combination of its vertices:

$$x = X_i\, w,$$

where X_i is an $n \times (n+1)$ matrix whose columns are the coordinates of the $n+1$ vertices of the simplex Δ_i^k, and w is an $(n+1)$-vector with components w_j that form a partition of unity; that is,

$$\sum_{j=1}^{n+1} w_j = 1, \qquad w_j \geq 0 \quad j = 1, 2, \ldots, n+1,$$

which may be written compactly in matrix form as

$$\left[1 \ldots 1\right] w = 1, \qquad -I w \leq 0. \tag{8}$$

In each simplex Δ_i^k, we thus minimize a new search function $\bar{s}_i^k(w)$ defined as

$$\begin{aligned} \bar{s}_i^k(w) &= Y_i\, w - K \left[R_i^2 - (X_i\, w - z_i^k)^T (X_i\, w - z_i^k)\right] \\ &= K\, w^T X_i^T X_i\, w + (Y_i - 2K(z_i^k)^T X_i)\, w \\ &\quad + K\,[(z_i^k)^T z_i^k - R_i^2]. \end{aligned} \tag{9}$$

where Y_i is an $1 \times (n+1)$ row vector whose elements are the function values at the $n+1$ vertices of the simplex Δ_i^k. Minimization of (9), subject to the constraints (8), can be performed exceptionally quickly using convex quadratic programming. This optimization gives a vector of weights w_0, which defines the initial guess for the local minimization of the function $s_i^k(x)$ within the simplex Δ_i^k. Since we have analytic expressions for the gradient and Hessian of $s_i^k(x)$, and a good initial guess of its minimum, we can apply either a Trust Region method, or Newton's method with Hessian modification, in order to find quickly the minimum of $s_i^k(x)$. Newton's method is a line search algorithm with a descent direction derived based on both the gradient and the Hessian of the function; because of the nonconvexity of $s_i^k(x)$, Hessian modification is required to ensure convergence. The Hessian modification that has been used in our numerical code is modified Cholesky factorization [13], and the line search algorithm used is a backtracking line search algorithm (Algorithm 3.1 in [27]). Convergence of this local optimization algorithm is proved in [27].

3.4 Convergence of Algorithm 2

Before analyzing the convergence properties of Algorithm 2, we establish a useful lemma.

Lemma 6 *Assume that the function of interest $f(x)$ and the interpolating function $p^k(x)$ at step $k > 0$ of Algorithm 2 are continuously twice differentiable functions with bounded Hessians. Denote $\lambda_{max}(\cdot)$ as the maximum eigenvalue of its argument, and K is chosen as follow*

$$K > \lambda_{max}(\nabla^2 f(x) - \nabla^2 p^k(x))/2 \tag{10}$$

for all x located in the feasible domain L. Then, there is a point $\tilde{x} \in L$ for which

$$s^k(\tilde{x}) \leq f(x^*), \tag{11}$$

where x^* is a global minimizer of $f(x)$.

Proof Consider Δ_i^k as a simplex in Δ^k which includes x^*. Since the uncertainty function e_i^k is defined for all x inside the simplex Δ_i^k as

$$e_i^k(x) = (r_i^k)^2 - \left(x - z_i^k\right)^T \left(x - z_i^k\right),$$

the Hessian of the uncertainty function e_i^k is simply

$$\nabla^2 e_i^k(x) = -2\, I.$$

Now define a function $G(x)$ for all $x \in L$ such that

$$G(x) = p^k(x) - K\, e^k(x) - f(x), \tag{12}$$

and thus

$$\nabla^2 G(x) = \left(\nabla^2 p^k(x) - \nabla^2 f(x)\right) + 2\,K\,I. \tag{13}$$

By choosing K according to (10), the function $G(x)$ is strictly convex inside the closed simplex that includes x^*; thus, the maximum value of $G(x)$ is located at one of its vertices (see, e.g. Theorem 1 of [16]). Moreover, by construction, the value of $G(x)$ at the vertices of this simplex is zero; thus, $G(x^*) \leq 0$, and therefore $s^k(x^*) \leq f(x^*)$. □

Lemma 6 allows us to establish the convergence of Algorithm 2.

Theorem 2 *At step $k \geq 0$ of Algorithm 2, assume that S^k is the set of available datapoints, that the function of interest $f(x)$ and the interpolating function $p^k(x)$ are continuously twice differentiable functions, and that L_p is a Lipschitz constant of $p^k(x)$. Assume also that K satisfies (10). Define x^* and x_k as the global minimizers of $f(x)$ and the search function $s^k(x)$ at step k, respectively, and r^k_{max} as the maximum circumradius of the triangulation Δ^k; then*

$$0 \leq \min_{z \in S^k} f(z) - f(x^*) \leq \epsilon_k \quad \textit{where} \tag{14a}$$

$$\epsilon_k = (L_f + L_p + 2Kr^k_{max}) \cdot \min_{i<k} \|x_i - x_k\|. \tag{14b}$$

Proof Select that i, with $i < k$, such that $\delta = \|x_i - x_k\|$ is minimized. By the Lipschitz norms of $p^k(x)$ and (6), we have

$$\begin{aligned} &\|p^k(x_i) - p^k(x_k)\| \leq L_p\,\delta, \\ &\text{and} \quad \|e^k(x_i) - e^k(x_k)\| \leq 2\,r^k_{\max}\,\delta. \end{aligned}$$

Noting that $s^k(x) = p^k(x) - Ke^k(x)$, we have

$$\|s^k(x_i) - s^k(x_k)\| \leq (L_p + 2\,K r^k_{\max})\,\delta.$$

Since x_i is one of the evaluation points at the k-th step, at this point the value of the uncertainty function $e^k(x_i)$ is zero, and the values of the interpolant $p^k(x_i)$ and the function $f(x_i)$ are equal. That is,

$$s^k(x_i) = p^k(x_i) - K\, e^k(x_i) = f(x_i). \tag{15}$$

Since x_k is taken to be the global minimum of $s^k(x)$ and K satisfies (10), based on Lemma 6, $s^k(x_k) \leq f(x^*)$; Thus,

$$\begin{aligned} f(x^*) \geq s^k(x_k) &\geq s^k(x_i) - (L_p + 2\,K\,r^k_{\max})\,\delta \\ &= f(x_i) - (L_p + 2\,K r^k_{\max})\,\delta \\ &\geq [f(x_k) - L_f\,\delta] - (L_p + 2\,K\,r^k_{\max})\,\delta, \\ \Rightarrow \quad f(x^*) &\geq f(x_k) - (L_f + L_p + 2\,K\,r^k_{\max})\,\delta. \end{aligned}$$

Remark 3 Algorithm 2 begins from a set of initial datapoints, and computes one new datapoint at each iteration. In this setting, (14) provides a (sometimes conservative) **termination certificate** that guarantees that a desired degree of convergence ϵ_{des} has been attained. Algorithm 2 is simply marched in k until $\epsilon_k \leq \epsilon_{\text{des}}$, where ϵ_k is defined in (14b).

Note that, if $r^k_{\max}$ is bounded, $\epsilon_k \to 0$ in the limit in which $k \to \infty$. Thus, there is a finite k for which $\epsilon_k \leq \epsilon_{\text{des}}$.

Remark 4 The existence of a bound L_p on the Lipschitz norm of $p^k(x)$ is required. The Lipschitz norm L_p of the interpolation $p^k(x)$ is, in general, cumbersome to compute. Subject to the above stated assumptions, a simpler way to prove convergence of (and, to certify a termination criterion for) Algorithm 2 that ensures that (14a) is attained for some $\epsilon_k \leq \epsilon_{\text{des}}$ is to calculate ε_k using linear interpolation, for which it can be shown that $L_p \leq L_f$, thereby replacing (14b) with

$$\epsilon_k = (2\,L_f + 2Kr^k_{\max}) \cdot \min_{y \in S^k} \|x_k - y\|.$$

Remark 5 In addition to the required assumptions of a bound L_p for the Lipschitz norm of $p^k(x)$, and existence of K which (10) holds, a bound for the maximum circumradius $r^k_{\max}$ of the triangulation Δ^k is also required. In general, algorithm 2 cannot guaranty this property. A slight but technically important change to Algorithm 2 is presented in the next section to address this issue.

4 Bounding the circumradii

In the previous section, we established that Algorithm 2 converges to the global minimum of the cost function under two assumptions: (a) the underlying function of interest $f(x)$ is Lipschitz and twice differentiable with bounded Hessian, and (b) the maximum circumradii of the simplicies in the triangulations are bounded as the algorithm proceeds. Without assumption (a), or something like it, not much can be done to assure global convergence, beyond requiring that the grid becomes everywhere dense as the number of function evaluations approaches infinity (see [38]). Assumption (b), however, is problematical, as Algorithm 2 doesn't itself ensure this condition. In this section, we make a small but important technical adjustment to Algorithm 2 to ensure that condition (b) is satisfied.

Distributing points in $\mathbb{R}^n$ such that the resulting Delaunay triangulation of these points has a bounded maximum circumradius is a common problem in 2D and 3D mesh generation (see [20,21,32]). Applying known strategies for this problem to the present application is challenging, however, due to the incremental nature of the point generation, the interest in $n > 3$, the large number and nonuniform distribution of points generated, and the possibly sharp corners of the feasible domain itself. In this section, we thus develop a new method for solving this problem that meets these several challenges.

The feasible domain L considered is defined in (1a); we assume for the remainder of this section that the problem is n-dimensional, is bounded by m constraints which result in M vertices of the feasible domain, that all redundant constraints have been eliminated [see step (B) of Algorithm 1], and that all equality constraints implied by the condition $Ax \leq b$ have been projected out of the problem [see step (C) of Algorithm 1].

Definition 5 Consider P as a point in L, $a_i^T x \leq b_i$ as a constraint which is not **active** (that is, equality) at P, and H_i as the $(n-1)$-dimensional hyperplane given by equality in this constraint. The **feasible constraint projection** of P onto H_i is the point P_L defined as the outcome of the following procedure:

0. Set $k = 1$ and $P_L^1 = P$.

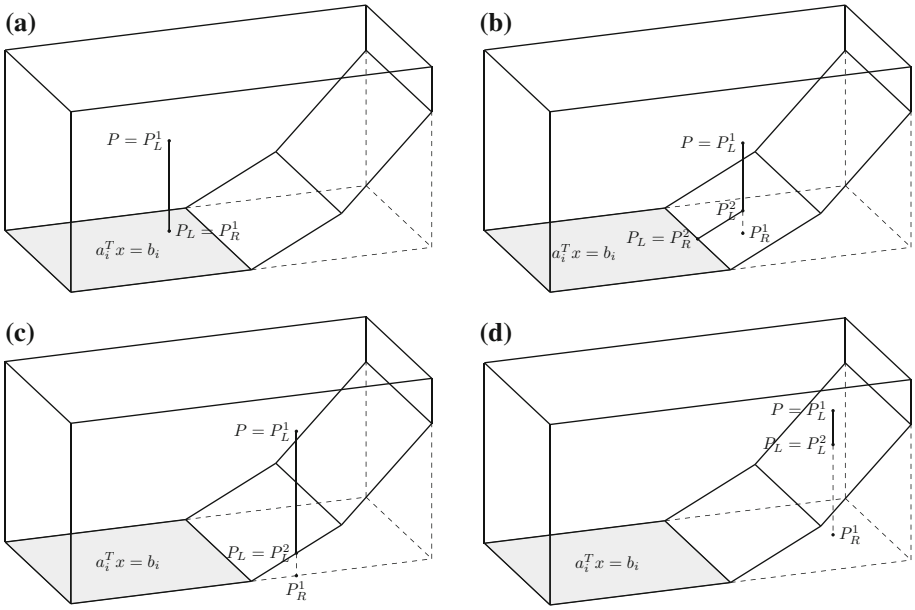

Fig. 2 Feasible constraint projections of various points P onto the constraint $a_i^T x \leq b$. **a** Complete, termination at step 3 of iteration $k = 1$, **b** complete, termination at step 3 of iteration $k = 2$, **c** incomplete, termination at step 1 of iteration $k = 2$, **d** incomplete, termination at step 2 of iteration $k = 2$

1. Define m_a^k as the number of active constraints at P_L^k, and H_L^k as the hyperplane (with dimension less than or equal to n) implied by this set of constraints. If $m_a^k = n - 1$, set $P_L = P_L^k$ and exit.
2. If $m_a^k > 0$ and there is no vertex of L that is contained within both H_L^k and H_i, set $P_L = P_L^k$ and exit.
3. Obtain P_R^k as the point which is contained within both H_L^k and H_i, and has minimum distance from P_L. If $P_R^k \in L$, set $P_L = P_R^k$ and exit.
4. Otherwise, set P_L^{k+1} as the intersection of the line segment from P_L^k to P_R^k with the boundary of L, increment k, and repeat from step 1.

If the above-described procedure exits at step 3, and thus $P_L \in H_i$, then P_L is said to be a **complete** feasible constraint projection; otherwise (that is, if the procedure exits at step 1 or 2), P_L is said to be an **incomplete** feasible constraint projection. The procedure described above will terminate after some $\overline{k}$ steps, where $\overline{k} < n$; the P_L^k for $k < \overline{k}$ are referred to as **intermediate** feasible constraint projections.

Some examples of complete and incomplete feasible constraint projections are given in Fig. 2.

Lemma 7 *Consider the P_L^k as the intermediate feasible constraint projections at each step $k \leq \overline{k}$ of a feasible constraint projection (see Definition 5), which exits at iteration $\overline{k}$, of some point $P \in L$ onto the hyperplane H_i defined by the constraint $a_i^T x \leq b_i$. For any point V which lies within the intersection of H_L^k and H_i,*

$$\frac{\|P - V\|}{\|P - P_T\|} \leq \frac{\|P_L^k - V\|}{\|P_L^k - P_T^k\|} \tag{16}$$

for $k \leq \bar{k}$, where P_T and P_T^k are the projections of P and P_L^k on H_i.

Proof In the procedure described in Definition 5, at each step k, if P_R^k is in L; then $P_L^{k+1} = P_R^k$. By construction, V is in the intersection of H_L^k and H_i, and P_R^k is the point in the intersection of H_L^k and H_i that has minimum distance from P_L^k; it thus follows that $\overrightarrow{P_L^k P_R^k}$ is perpendicular to $\overrightarrow{P_R^k V}$. Since P_L^{k+1} is a point on the line between P_L^k and P_R^k, it follows that $\overrightarrow{P_L^{k+1} P_R^k}$ is also perpendicular to $\overrightarrow{P_R^k V}$. Thus,

$$\begin{aligned} \|P_L^k - V\|^2 &= \|P_L^k - P_R^k\|^2 + \|V - P_R^k\|^2, \\ \|P_L^{k+1} - V\|^2 &= \|P_L^{k+1} - P_R^k\|^2 + \|V - P_R^k\|^2, \\ \|P_L^{k+1} - P_R^k\| &\leq \|P_L^k - P_R^k\|, \\ \frac{\|P_L^k - V\|}{\|P_L^k - P_R^k\|} &\leq \frac{\|P_L^{k+1} - V\|}{\|P_L^{k+1} - P_R^k\|}. \end{aligned} \tag{17}$$

Since P_L^k, P_L^{k+1} and P_R^k are collinear and P_R^k is on the hyperplane $a_i^T x = b_i$, it follows that

$$\frac{\|P_L^{k+1} - P_R^k\|}{\|P_L^k - P_R^k\|} = \frac{\|P_L^{k+1} - P_T^{k+1}\|}{\|P_L^k - P_T^k\|}. \tag{18}$$

Combining (17) and (18) over several steps k and noting that $P_L^1 = P$, (16) follows. □

Definition 6 Consider P as a point in a set of points S in the convex polyhedra L, $a_i^T x = b_i$ as a constraint which is not active at P, and H_i as the hyperplane defined by this constraint. The points P_L^k are taken as the intermediate feasible constraint projections of P onto H_i (see Definition 5, which we take as exiting at iteration $\bar{k}$). With respect to some parameter $r > 1$, the point P is said to be **poorly situated** with respect to the constraint $a_i^T x \leq b_i$ if the feasible constraint projection is complete and, for any point $V \in S$ which is in both H_L^k and H_i,

$$\frac{\|P_L^k - V\|}{\|P_L^k - P_R^k\|} > r, \quad \forall\ 1 \leq k \leq \bar{k}; \tag{19}$$

otherwise, the point P is said to be **well situated** with respect to the constraint $a_i^T x \leq b_i$. The set of data points S is a **well-situated set** if, for all pairs of points $P \in S$ and constraints $a_i^T x \leq b_i$ defining L, P is well-situated with respect to the constraint $a_i^T x \leq b_i$.

Remark 6 It is easy to verify that the set of vertices of the feasible domain L is itself a well-situated set for any $r > 1$.

The important property of a well-situated set S is that the maximum circumradius of the Delaunay triangulation of S is bounded by r [the factor used in (19), which will be considered further at the end of Sect. 4] and some parameters related to L. These geometric parameters are identified in Definitions 7 and 8, and existence of a bound for the maximum circumradius is proved in Theorem 3, based on Lemmas 9 and 10.

Definition 7 Consider $A_a(V)$ as the set of active constraints at a vertex V of a feasible domain L with redundant constraints eliminated [see step (B) of Algorithm 1], and all equality constraints removed [see step (C) of Algorithm 1]. Since there are no redundant constraints, $A_a(V)$ includes exactly n constraints which are linearly independent. Consider $a_i^T x \leq b_i$ as

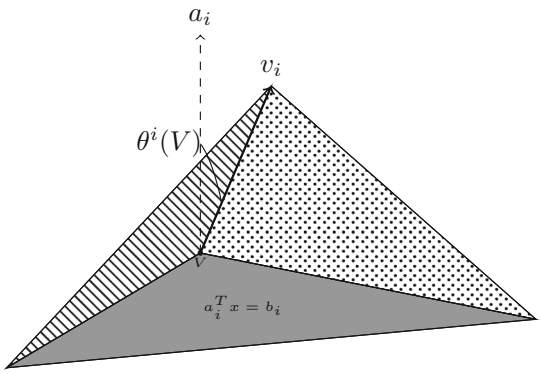

Fig. 3 The skewness of a vertex V with respect to constraint $a_i^T x \leq b_i$ in $n = 3$ dimensions is defined as $S^i(V) = 1/\cos(\theta^i(V))$, where $\theta^i(V)$ is the angle indicated

a constraint in $A_a(V)$, and $A_a^i(V)$ as the set of all active constraints at V except $\{a_i^T x \leq b_i\}$. It can be observed that the space defined by $A_a^i(V)$ is a ray from V. In other words, each point in this one-dimensional space can be written as $V + \alpha\, v_i$, where α is a positive real number. Defining $\theta^i(V)$ as the angle between a_i and v_i, the skewness of the vertex V with respect to the constraint $a_i^T x \leq b_i$ is defined as $S^i(V) = 1/\cos(\theta^i(V))$. The skewness of the feasible domain L, denoted $S(L)$, is the maximum of $S^i(V)$ over all vertices V and active constraints $a_i^T x \leq b_i$ at V.

Remark 7 If the feasible domain L is defined by simple box constraints ($a \leq x \leq b$), then the skewness of all vertices with respect to all active constraints (and, thus, the skewness of the domain L itself) is equal to 1. In $n = 2$ dimensions, the skewness of each vertex is simply $S^i(V) = 1/|\sin(\theta)|$, where θ is the angle of vertex V. In $n = 3$ dimensions, the value of $\theta^i(V)$ is illustrated in Fig. 3. Note that, in general, $S^i(V) \geq 1$.

Lemma 8 *Consider V as a vertex of L, and $A_a(V)$ as the set of constraints active at V. Consider some point $x \in L$ at which the set of active constraints, denoted $A_{a_1}(V)$, is a proper subset of $A_a(V)$. Denote $a_i^T x \leq b_i$ as a constraint in $A_a(V)$ which is not in $A_{a_1}(V)$. Define x_1 as the projection of x onto the hyperplane defined by $a_i^T x = b_i$, and define x_2 as the projection of x onto the hyperplane defined by the union of $a_i^T x = b_i$ and $A_{a_1}(V)$. Then,*

$$1 \leq \frac{\|x - x_2\|}{\|x - x_1\|} \leq S^i(V) \leq S(L). \tag{20}$$

Proof Consider x_3 as the point in the hyperplane defined by $A_a^i(V)$ with minimum distance from x, where $A_a^i(V)$ is identified in Definition 7. By construction, $\|x - x_3\| \geq \|x - x_2\|$; note that x_3 is also in the hyperplane defined by $A_{a_1}(V)$. Moreover, according to Definition 7, $\|x - x_3\|/\|x - x_1\| = S^i(V)$. □

Definition 8 For each pair of vertices V of L and constraints $a_i^T x \leq b_i$ not active at V, V_L is defined as the projection of V onto the hyperplane defined by $a_i^T x = b$. Define $L_1 = \max_{x,y \in L} \|x - y\|$ as is the **diameter** of L. The **aspect ratio** $\kappa(L)$ is taken as the maximum value of $L_1/\|V - V_L\|$ for all pairs of vertices V and constraints $a_i^T x \leq b_i$ not active at V (Fig. 4).

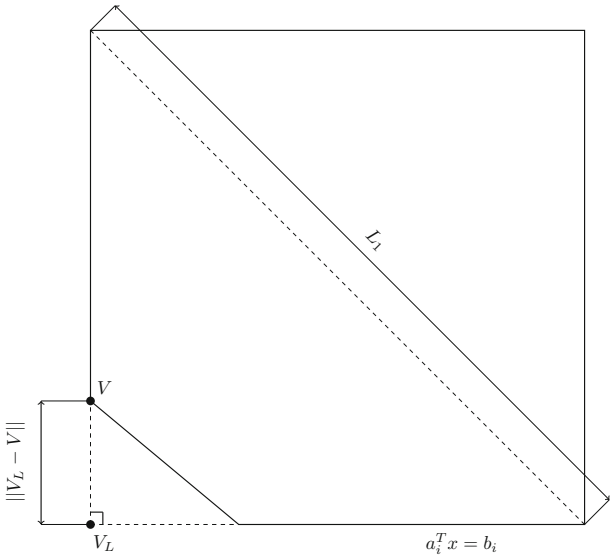

Fig. 4 This figure represent the aspect ratio of the convex polyhedra L. Note that the vertex V and constraint $a_i^T x = b_i$ has minimum distance from all pair of vertex and constraints, and L_1 is the diameter of the polyhedra L. The aspect ratio is equal to $\kappa(L) = \frac{L_1}{V - V_T}$ where $\|V - V_T\|$ and L_1 are shown

Lemma 9 *Consider S as a set of feasible points in L (including the vertices of L) which is well-situated (see Definition 6) with factor r. Then, for each pair of points $P \in S$ and constraints $a_i^T x \leq b_i$ not active at P, there is a point $V \in S$ such that $a_i^T V = b_i$ and*

$$1 \leq \frac{\|P - V\|}{\|P - P_T\|} \leq \max\{rS(L), \kappa(L)\}, \tag{21}$$

where P_T is taken as the projection of P onto the hyperplane H_i, and $S(L)$ and $\kappa(L)$ are the skewness and aspect ratio of the feasible domain L.

Proof Consider P_L as the feasible constraint projections of P onto H_i (see Definition 5, which we take as exiting at iteration $\bar{k}$). According to this procedure, there are three possible termination conditions, each of which is considered below.

First, consider the case in which the feasible constraint projection is terminated at step 2. In this case, $P_L = P_L^{\bar{k}}$, and there is no vertex of L that is contained within both $H_L^{\bar{k}}$ and H_i. Therefore, the point in intersection of the feasible domain L and the hyperplane $H_L^{\bar{k}}$ which has minimum distance from the hyperplane H_i is a vertex of L, denoted V. Thus,

$$\|V - V_T\| \leq \|P_L - P_{L^T}\|, \tag{22}$$

where P_{L^T}, P_T^k and V_T are the projections of P_L, P_L^k and V onto H_i; thus, $P_{L^T} = P_T^{\bar{k}}$. Further, at each step of the procedure of feasible constraint projection (Definition 5), the distance of the point considered to H_i is reduced. Thus,

$$\|P_L - P_{L^T}\| = \|P_L^{\bar{k}} - P_T^{\bar{k}}\| \leq \|P - P_T\|. \tag{23}$$

Using (22) and (23),

$$\|V - V_T\| \leq \|P - P_T\| \tag{24}$$

On the other hand, $\|P - V\| \leq L_1$, where L_1 is the diameter of the feasible domain L. Thus,

$$\frac{\|P - V\|}{\|P - P_T\|} \leq \frac{L_1}{\|V - V_T\|} \leq \kappa(L). \tag{25}$$

Next, consider the case in which the feasible constraint projection is terminated at step 1. In this case, the number of active constraints at P_L is $m_a = n - 1$, and $P_L = P_L^{\bar{k}}$. If there is no vertex of L that is contained within both $H_L^{\bar{k}}$ and H_i, the situation is similar to the previous case, and (25) again follows. On the other hand, if there is a vertex V of L which is in both $H_L^{\bar{k}}$ and H_i, then it follows from Lemma 7 that

$$\frac{\|P - V\|}{\|P - P_T\|} \leq \frac{\|P_L - V\|}{\|P_L - P_{L^T}\|}. \tag{26}$$

Note that $A_a(P_L)$ is a proper subset of $A_a(V)$ which does not include the constraint $a_i^T x \leq b_i$, and V is the only point which is in both $H_L^{\bar{k}}$ and H_i, as the intersection of n linear independent hyperplanes is a unique point. Therefore, via Lemma 8 (taking $x = P_L$ and thus, by construction, $x_1 = P_{L^T}$ and $x_2 = V$),

$$\frac{\|P_L - V\|}{\|P_L - P_{L^T}\|} \leq S^i(V) \leq S(L) \tag{27}$$

Using (26) and (27)

$$\frac{\|P - V\|}{\|P - P_T\|} \leq S(L) \leq rS(L). \tag{28}$$

Finally, consider the case in which the feasible constraint projection is terminated at step 3, and thus the process is complete, and $P_L = P_R^{\bar{k}}$. Since P is well situated with respect to the constraint $a_i^T x \leq b_i$ (see Definition 6), there is a $k \in \{1, 2, \ldots, \bar{k}\}$ and a point $V \in S$ which is in both H_L^k and H_i such that

$$\frac{\|P_L^k - V\|}{\|P_L^k - P_R^k\|} \leq r. \tag{29}$$

Since V is in both H_L^k and H_i, there is a vertex of L, denoted W, which is in both H_L^k and H_i. Moreover, P_L^k is not in H_i, and $A_a(P_L^k)$ is a proper subset of $A_a(W)$. Denote again P_T^k as the projection of P_L^k on the hyperplane H_i. Via Lemma 8 (taking $x = P_L^k$ and thus, by construction, $x_1 = P_T^k$ and $x_2 = P_R^k$), it follows that

$$\frac{\|P_L^k - P_R^k\|}{\|P_L^k - P_T^k\|} \leq S^i(W) \leq S(L). \tag{30}$$

Combining (29) and (30),

$$\frac{\|P_L^k - V\|}{\|P_L^k - P_T^k\|} \leq rS(L)$$

Thus, via Lemma 7, it follows that

$$\frac{\|P - V\|}{\|P - P_T\|} \leq \frac{\|P_L^k - V\|}{\|P_L^k - P_T^k\|} \leq rS(L). \tag{31}$$

□

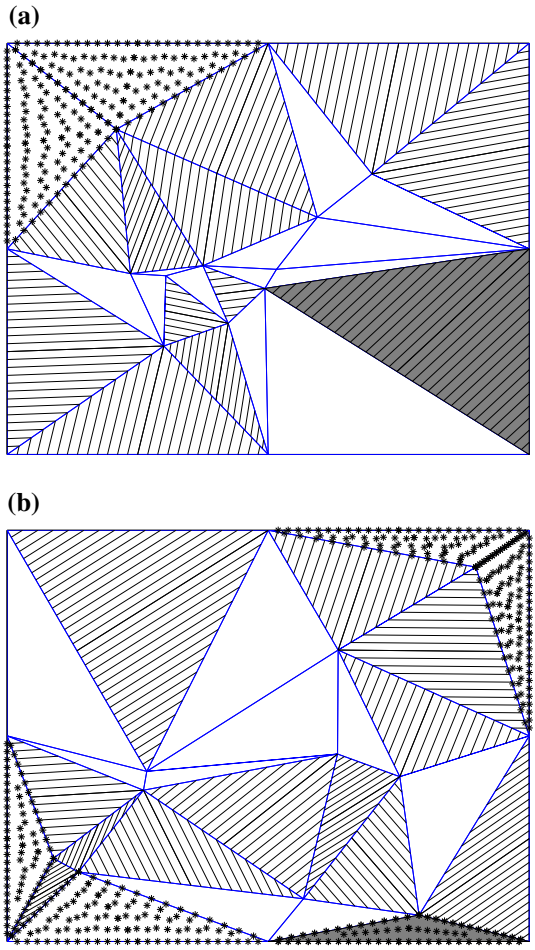

Fig. 5 The position of candidate simplices for a representative *triangulation*. *Hatched triangles* are interior candidate simplices, and *triangles filled* with *stars* are boundary candidate simplices. The *dark shaded area* is the maximal simplex. **a** An interior candidate simplex is maximal, **b** a boundary candidate simplex is maximal

Lemma 9 allows us to show that the maximum circumradius of a Delaunay triangulation of a well-situated set of points is bounded. In order to do this, some additional lemmas and definitions are helpful.

Definition 9 A simplex Δ_x in a Delaunay triangulation Δ^k of a set of points S (including the vertices of L) which has the maximum circumradius among all simplices is called a **maximal** simplex. Note that a given triangulation might have more than one maximal simplices.

A simplex Δ_x is called a **candidate** simplex (Figs. 4) and (Fig. 5) if either (a) the circumcenter of Δ_x is inside or on the boundary of Δ_x, or (b) an $n-1$ dimensional face p of this simplex forms part of the boundary of L corresponding to equality in the constraint $a_i^T x \leq b_i$, and the circumcenter of Δ_x violates this constraint. Case (a) is denoted an **interior** candidate simplex, and case (b) is denoted a **boundary** candidate simplex.

Lemma 10 *There is a maximal simplex in a Delaunay triangulation Δ^k of a set of points S (including the vertices of L) which is a candidate simplex.*

Proof Consider Δ_x as a maximal simplex of a Delaunay triangulation Δ^k. If Δ_x is a candidate simplex, then the lemma is true. Otherwise, define p as an $n-1$ dimensional face of this simplex, lying within an $n-1$ dimensional hyperplane H, in which V, the vertex of Δ_x that is not in H, and Z, the circumcenter of Δ_x, are on opposite sides of H, and none of the $n-1$ dimensional boundaries of L lie within H.

Then there is a simplex Δ_x^1 which is a neighbor of Δ_x that shares the face p. Define V' as the vertex of Δ_x^1 which is not in H, and Z' as the circumcenter of Δ_x^1. Since the triangulation is Delaunay

$$\|Z - V\| \leq \|Z - V'\| \text{ and } \|Z' - V'\| \leq \|Z' - V\|. \tag{32}$$

Define Z_T as the projection of both Z and Z' on H (by construction, they have the same projection), and V_T and V_T' as the projections of V and V' on H, respectively. By the assumption, Z and V' are on one side of H, and V is on the other side. Thus, (32) implies

$$\begin{aligned} &\|V_T - Z_T\|^2 + [\|Z_T - Z\| + \|V - V_T\|]^2 \leq \\ &\|V_T' - Z_T\|^2 + [\|V' - V_T'\| - \|Z_T - Z\|]^2. \end{aligned} \tag{33}$$

Moreover, regardless of the position of Z', we may write

$$\begin{aligned} &\|Z' - V'\|^2 \geq \\ &\quad \|V_T' - Z_T\|^2 + [\|V' - V_T'\| - \|Z_T - Z'\|]^2, \\ &\|Z' - V\|^2 \leq \\ &\quad \|V_T - Z_T\|^2 + [\|V' - V_T'\| + \|Z_T - Z'\|]^2; \end{aligned}$$

thus, due to (32),

$$\begin{aligned} &\|V_T' - Z_T\|^2 + [\|V' - V_T'\| - \|Z_T - Z'\|]^2 \\ &\quad \leq \|V_T - Z_T\|^2 + [\|V - V_T\| + \|Z_T - Z'\|]^2. \end{aligned} \tag{34}$$

Adding (33) and (34) gives

$$\begin{aligned} &\|Z_T - Z\|\|V - V_T\| - \|V' - V_T'\|\|Z_T - Z'\| \\ &\quad \leq \|V - V_T\|\|Z_T - Z'\| - \|V' - V_T'\|\|Z_T - Z\|, \end{aligned}$$

and thus

$$\begin{aligned} &\|Z_T - Z\|[\|V - V_T\| + \|V' - V_T'\|] \\ &\quad \leq \|Z_T - Z'\|[\|V - V_T\| + \|V' - V_T'\|], \\ &\|Z - Z_T\| \leq \|Z' Z_T\|. \end{aligned} \tag{35}$$

Define W as a common vertex of Δ_x and Δ_x', noting that W must be in H. Since $Z - Z_T$ and $Z' - Z_T$ are perpendicular to $Z_T - W$, (35) gives

$$\|Z - W\| \leq \|Z' - W\|. \tag{36}$$

It may be shown analogously that, if (32) is a strict inequality, then (36) is a strict inequality as well. Further, since Δ_x is a maximal simplex, $\|Z - W\| \geq \|Z' - W\|$; thus, the inequality (36) must be an equality, and Δ_x and Δ_x' are both maximal. We may also conclude that[3] (32) must also be an equality, which implies that Z and Z' are, in fact, the same point.

[3] The logic for this conclusion is as follows: if (i) $a \leq b$ and (ii) $a < b \rightarrow c < d$, then, if $c = d$, then $a = b$.

Now define F as the polygon which is equal to the union of those simplices of Δ^k whose circumcenter is Z; note that all of the simplices that make up F are maximal. If Z is inside F, the simplex which includes Z is an interior candidate simplex. If Z is not inside F then, by the above reasoning, any boundary of F which Z is on the opposite side of must also be a boundary of L, and the simplex in F which shares this boundary is a boundary candidate simplex. □

Theorem 3 *Consider Δ^k as an n dimensional Delaunay triangulation of a set of well-situated points $S \in L$ (including the vertices of L), with factor of r. Then*

$$R_{max} \leq L_2 r_1^{n-1} \quad \textit{where} \quad r_1 = \max\{r S(L), \kappa(L)\},$$

where R_{max} is the maximum circumradius, L_2 is the maximum edge length in all simplices, and $S(L)$ and $\kappa(L)$ are the skewness and aspect ratio of L.

Proof This theorem is shown by induction on n, the dimension of the problem. For $n = 1$, the circumcenter of any simplex is located in L, and the lemma is trivially satisfied. Assuming the theorem is true for the $n-1$ dimensional case, we now show that the theorem is also true for the n dimensional case.

Consider Δ_x as a maximal simplex of the n dimensional Delaunay triangulation Δ^k, with circumcenter Z, which is also a candidate simplex (see Lemma 10). If Δ_x is an interior candidate simple, it includes its circumcenter, and the lemma is trivially satisfied, since Z is located in L. Otherwise, Δ_x is a boundary candidate simplex, and there is a constraint $a_i^T x \leq b_i$ bounding L which is active at n vertices of Δ_x, and Z violates this constraint. Denote H_i as the $n-1$ dimensional hyperplane which contains this constraint.

Consider P as the vertex of Δ_x which is not in H_i. Define Z_T and P_T as the projections of Z and P onto H_i, and W as a vertex of Δ_x which is in H_i; then

$$\begin{aligned}
&\|Z - P\| = \|Z - W\|, \\
&[\|Z - Z_T\| + \|P - P_T\|]^2 + \|Z_T - P_T\|^2 \\
&\quad = \|Z - Z_T\|^2 + \|Z_T - W\|^2 \\
&2\|Z_T - Z\|\|P - P_T\| + \|P - P_T\|^2 \\
&\quad = \|Z_T - W\|^2 - \|Z_T - P_T\|^2
\end{aligned} \tag{37}$$

By construction Z_T is the circumcenter of an $n-1$ dimensional simplex Δ_x^1 which includes all vertices of Δ_x except P. Note that restriction of Δ^k onto the hyperplane H_i is itself an $n-1$ dimensional Delaunay triangulation. In other words, for any point $V \in S$ that is in H_i,

$$\begin{aligned}
\|Z_T - V\| &\geq \|Z_T - W\| \\
\|P_T - V\| &\geq \|Z_T - V\| - \|Z_T - P_T\| \\
\|P_T - V\| &\geq \|Z_T - W\| - \|Z_T - P_T\| \\
\|Z_T - W\|^2 - \|Z_T - P_T\|^2 &= (\|Z_T - W\| + \|Z_T - P_T\|)(\|Z_T - W\| - \|Z_T - P_T\|)
\end{aligned}$$

According to equation (37), $\|Z_T - W\| \geq \|Z_T - P_T\|$; it thus follows from the above equations that

$$\|Z_T - W\|^2 - \|Z_T - P_T\|^2 \leq 2\|Z_T - W\|\|P_T - V\|. \tag{38}$$

Combining (37) and (38), we may write

$$\|Z_T - Z\|\|P - P_T\| \leq \|Z_T - W\|\|P_T - V\| \tag{39}$$

Furthermore, by construction, $Z_T - Z$ and $P - P_T$ are perpendicular H_i to ; therefore,

$$\begin{aligned}
\|Z - W\|^2 &\leq \|Z_T - W\|^2[1 + \frac{\|V - P_T\|^2}{\|P - P_T\|^2}] \\
\|P - V\|^2 &= \|V - P_T\|^2 + \|P - P_T\|^2 \\
\|Z - W\| &\leq \|Z_T - W\| \frac{\|P - V\|}{\|P - P_T\|}
\end{aligned} \tag{40}$$

On the other hand, since S is well-situated with factor of r, according to Lemma 9, there is a point V in S which is in H_i, and

$$\frac{\|P - V\|}{\|P - P_T\|} \leq r_1. \tag{41}$$

Note that, if S is well situated with factor of r, the subset of points of S which lie within H_i, denoted S_i, are also well situated with factor of r. Since $\|Z_T - W\|$ is the circumradius of the $n-1$ dimensional simplex Δ_x^1 of the Delaunay triangulation of S_i, by the inductive hypothesis,

$$\|Z_T - W\| \leq L_2 r_1^{n-2}$$

Applying (40) and (41), it thus follows from the above condition that

$$\|Z - W\| \leq L_2 r_1^{n-1}$$

Since $\|Z - W\|$ is equal to the maximum circumradius of Δ^k, the theorem is proved. □

We now use Theorem 3 to perform a slight modification of Algorithm 2 in a way that ensures the set of datapoints remains well situated, with factor r, as the iteration proceeds. In this way, a bound for the maximum circumradius of the Delaunay triangulations generated by the algorithm is assured.

Algorithm 2 is initialized with the vertices of L. By Remark 6, this set of points is well situated. Algorithm 2 then (a) adds to this initial set of points a number of user specified points of interest, and then (b) adds (at step 5) a new datapoint [selected carefully, as described in the algorithm] to the existing set of datapoints at each iteration, until convergence. We now modify Algorithm 2 such that each time a new datapoint P is added, in both steps a and b above, an adjustment Q to the location of point P is made, if necessary, in order to ensure that set of datapoints remains well situated. This adjustment is performed as follows.

Algorithm 3 *Assume S is a well-situated set of points, and P is a candidate point to be added to this set (after adjustment, if necessary).*

0. *Set Q=P.*
1. *Find a constraint $a_i^T x \leq b_i$ for which P is not in a well situated position. If none can be found, stop the algorithm, and return Q.*
2. *Replace Q by the feasible constraint projection of Q on $a_i^T x \leq b_i$ (see Definition 5).*
3. *Repeat from step 1 until the algorithm stops.*

Note that $A_a(Q)$ includes the active constraints of P. At each step of the above algorithm, an additional constraint is added. Thus, the above algorithm stops after at most $n-1$ iterations.

In Lemma 11, we show that Algorithm 2 still converges, even if we add Q instead of P at each iteration.

Lemma 11 *Consider S as a well-situated set of points in L, and Q as the outcome of Algorithm 3 from input P. Then,*

$$\min_{x\in S} \|P - x\| \leq \rho \cdot \min_{y\in S} \|Q - y\|, \tag{42}$$

$$\rho = \left[2r_1^2 \left(1 - \frac{1}{r^2}\right)\right]^{-\frac{n-1}{2}}, \tag{43}$$

$$r_1 = \max\{rS(L), \kappa(L)\}. \tag{44}$$

Proof Consider $y \in S$ as a point which minimizes $\|Q - y\|$. If a constraint $a_i^T x \leq b_i$ exists in $A_a(Q)$ which is not active at y, since S is well-situated, according to Lemma 9, there is a point $y^1 \in S$ such that $a_i^T x \leq b_i$ is active at it, and

$$\frac{\|y - y^1\|}{\|y - y_T\|} \leq r_1, \tag{45}$$

where y_T is the projection of y on the hyperplane $a_i^T x = b_i$. By construction,

$$\begin{aligned}
\|y - Q\|^2 &= \|y - y_T\|^2 + \|y_T - Q\|^2,\\
\|y - Q\| &\geq \frac{\|y - y_T\| + \|y_T - Q\|}{\sqrt{2}},\\
\|y^1 - Q\| &\leq \|y^1 - y_T\| + \|y_T - Q\|,\\
\frac{\|y - Q\|}{\|y^1 - Q\|} &\geq \frac{1}{\sqrt{2}} \frac{\|y - y_T\| + \|y_T - Q\|}{\|y^1 - y_T\| + \|y_T - Q\|}.
\end{aligned} \tag{46}$$

Using (45) and (46), we have:

$$\frac{\|y - Q\|}{\|y^1 - Q\|} \geq \frac{1}{\sqrt{2}\, r_1}. \tag{47}$$

Using (47), recursively over the $k \leq n - 1$ binding constraints at point Q, we will derive that there is a point $y^k \in S^k$ in which $A_a(Q) \subseteq A_a(y^k)$, and

$$\frac{\|y - Q\|}{\|y^k - Q\|} \geq \left(\frac{1}{\sqrt{2}\, r_1}\right)^{n-1}. \tag{48}$$

Take $V = y^k$; then we will show that

$$\frac{\|P - V\|}{\|Q - V\|} \leq \left(1 - \frac{1}{r^2}\right)^{-\frac{n-1}{2}}. \tag{49}$$

According to Algorithm 3, Q is derived by a series of successive complete feasible constraint projections of a point P onto various constraints of L which are active at Q. Assume that m feasible constraint projections are performed in during the process of Algorithm 3, and $\{P^1, P^2, \ldots, P^{m+1}\}$ is the series of points which are generated by Algorithm 3. In this way, $P^1 = P$, $P^{m+1} = Q$, and P^i for $1 < i \leq m + 1$ is the feasible-constraint-projection of P^{i-1} onto a constraint of L demoted by $a_i^T x \leq b_i$.

Define $P_L^{i,j}$ as the intermediate feasible constraint projection of P^i onto constraint $a_i^T x \leq b_i$, at step j, and $H_L^{i,j}$ as the hyperplane (with dimension less than or equal to n) implied by $A_a(P_L^{i,j})$. Then denote $P_R^{i,j}$ as a point in the intersection of H_i and $H_L^{i,j}$, that has minimum distance from $P_L^{i,j}$. (This is similar to P_R^k in Definition 5).

Since the point P^i is in a poorly situated position with respect to the constraint $a_i^T x \leq b_i$, and $A_a(P_L^{i,j}) \subseteq A_a(Q)$, and therefore V is in both H_i and $H_L^{i,j}$,

$$\frac{\|P_L^{i,j} - V\|}{\|P_L^{i,j} - P_R^{i,j}\|} > r$$
$$\|P_L^{i,j} - V\|^2 = \|P_L^{i,j} - P_R^{i,j}\|^2 + \|P_R^{i,j} - V\|^2$$
$$\frac{\|P_L^{i,j+1} - V\|}{\|P_L^{i,j} - V\|} \geq \frac{\|P_R^{i,j} - V\|}{\|P_L^{i,j} - V\|} \geq \sqrt{1 - \frac{1}{r^2}}$$

Moreover, at each iteration of the feasible constraint projection, a linearly independent constraint is added to the set of active constraints, therefore, step 3 of the procedure of feasible boundary projection could happened at most $n-1$ times. Thus, (49) is satisfied which shows (42) when $A_a(Q) \subseteq A_a(V)$. Furthermore, by using (48) and (49), (42) is satisfied when $A_a(Q) \nsubseteq A_a(V)$. □

Theorem 4 *Algorithm 2, with the adjustment described in Algorithm 3, will converge to the global minimum of the feasible domain L if the parameter K satisfies (10), and $p^k(x)$ is Lipschitz with a single Lipschitz constant L_p for all steps k.*

Proof Consider S^k as the set of datapoints at step k, x_k as the global minimizer of $s^k(x)$, and x_k' as the outcome of Algorithm 3 for input x_k. Denote δ_k and δ_k^1 as follows:

$$\delta_k^1 = \min_{y \in S^k} \|x_k - y\|,$$
$$\delta_k = \min_{y \in S^k} \|x_k' - y\|.$$

Note that $S^{k+1} = S^k \cup \{x_k'\}$.

$$0 \leq \min_{z \in S^k} f(z) - f(x^*) \leq (L_p + 2K r_{\max}^k)\delta_k^1, \tag{50}$$

where $r_{\max}^k$ is the maximum circumradius of a Delaunay triangulation for S^k, and L_p is the Lipschitz constant of $p^k(x)$.

Since S^k is a well-situated set with factor of r, according to Theorem 3,

$$r_{\max}^k \leq L_2 r_1^{n-1}. \tag{51}$$

where r_1 and L_2 are constants. Moreover, via Lemma 11,

$$\delta_k^1 \leq \rho \delta_k, \tag{52}$$

where ρ is a constant defined in (43). Thus, using (50), (51) and (52), it follows that

$$0 \leq \min_{z \in S^k} f(z) - f(x^*) \leq \epsilon_k$$
$$\epsilon_k = \rho \left(L_p + 2K L_2 r_1^{n-1}\right) \delta_k. \tag{53a}$$

Since the feasible domain L is bounded, $\delta_k \to 0$ as $k \to \infty$. Thus, Algorithm 2, with the adjustment described in Algorithm 3 incorporated, will achieve $\epsilon_k \leq \epsilon_{\text{des}}$ in finite k, where ϵ_k defined in (53a) for any specified $\epsilon_{\text{des}} > 0$. □

Remark 8 The parameter $r > 1$ represents a balance between two important tendencies of Algorithm 2, with the adjustment described in Algorithm 3. For the $r \to 1$, many feasible constraint projections are performed, and thus many datapoints are computed on the boundary of F; as a result, a restrictive bound on the maximum circumradius of the triangulation is available. On the other hand, as r is made large, fewer feasible constraint projections are performed, and thus fewer datapoints are computed on the boundary of F; as a result, the bound on the maximum circumradius of the triangulation is less restrictive. A good balance between these two competing objectives seems to be given by $r = c\sqrt{n}$ where c is an $O(1)$ constant; note that Algorithm 2 is recovered in the $r \to \infty$ limit.

5 Adapting K

The tuning parameter K in Algorithms 2 and 3 specifies the trade-off between global exploration, which is emphasized for large K, and local refinement, which is emphasized for small K. In this section, we develop a method to adjust the tuning parameter K at each iteration k in such a way as to maximally accelerate local refinement while still assuring global convergence.

The method builds on the fact that, if there exists an $\tilde{x}$ such that $p^k(\tilde{x}) - K\, e^k(\tilde{x}) \leq f(x^*)$ where $f(x^*)$ is a global minimum of $f(x)$ at each step k of Algorithm 2, then (11) is sufficient to establish convergence in Theorems 2 and 4, and (10) may be relaxed. Furthermore, it is not necessary to choose constant value for K in Algorithm 2, instead we may adapt K^k at each step k in such a way that $K^k \geq 0$ is bounded and $p^k(\tilde{x}) - K^k\, e^k(\tilde{x}) \leq f(x^*)$ at each step k of Algorithm 2.

If y_0 is a known lower bound for $f(x)$ over the feasible domain L, then if we choose K^k adaptively at each step of Algorithm 2 such that

$$0 \leq K^k \leq K_{\max}, \tag{54a}$$

$$\exists\, \tilde{x} \in L \quad p^k(\tilde{x}) - K^k\, e(\tilde{x}) \leq y_0, \tag{54b}$$

then the Algorithm 2 will converge to a global minimizer.

Note that reduced values of K^k accelerate local convergence. Thus, at each step k, we seek the smallest value of K^k which satisfies (54). The optimal K^k can be found simply as follows

$$K^k = \min \frac{p^k(x) - y_0}{e^k(x)}. \tag{55}$$

It is trivial to verify that the x which minimizes (55) also minimizes the corresponding search function $p^k(x) - K^k\, e^k(x)$. Thus, an alternative definition of the search function is $s_a^k(x) = \frac{p^k(x) - y_0}{e^k(x)}$, which has a same minimizer as $s^k(x) = p^k(x) - K^k e^k(x)$ for the optimal value of K^k given in (55).

If at some step k, the solution of (55) is negative, we take K^k at that step as zero, and the adaptive search function as $s_a^k(x) = p^k(x)$.

The new search function $s_a^k(x) = \frac{p^k(x) - y_0}{e^k(x)}$ is defined piecewise, similar to the original search function. Thus, we have to solve several optimization problems with linear constraints in order to minimize $s_a^k(x)$ in L. Following similar reasoning as in Lemma 5, we can relax these constraints: for each simplex Δ_i^k, we can instead minimize $s_{a,i}^k(x) = \frac{p^k(x) - y_0}{e_i^k(x)}$ in the intersection of the circumsphere Δ_i^k and the feasible domain L.

Again in order to minimize $s^k_{a,i}(x)$ for each simplex, a good initial guess is required. In each simplex Δ^k_i, a minimizer generally has a large value of $e^k_i(x)$; therefore, the projection of the simplex's circumcenter onto the simplex itself is a good initialization point for searching for the minimum of $s^k_{a,i}(x)$. This initial point for each simplex is denoted $\hat{x}^k_i$.

As before, with this initial guess for the minimum of $s^k_{a,i}(x)$ in each simplex, we can find the global minimum using a Newton method with Hessian modification. We thus need the gradient and Hessian of the function $s^k_{a,i}(x)$:

$$\nabla\left(\frac{p^k(x)-y_0}{e^k_i(x)}\right)=\frac{\nabla\left(p^k(x)\right)}{e^k_i(x)}-(p(x)-y_0)\frac{\nabla e^k_i(x)}{e^k_i(x)^2}$$

$$\begin{aligned}\nabla^2\left(\frac{p^k(x)-y_0}{e^k_i(x)}\right)&=\frac{\nabla^2\left(p^k(x)\right)}{e^k_i(x)}\\&-\frac{\left(\nabla p^k(x)\right)\left(\nabla e^k_i(x)\right)^T}{e^k_i(x)^2}-\frac{\left(\nabla e^k_i(x)\right)\left(\nabla p(x)\right)^T}{e^k_i(x)^2}\\&+\left(p^k(x)-y_0\right)\left(-\frac{\nabla^2 e^k_i(x)}{e^k_i(x)^2}+2\frac{\nabla e^k_i(x)\nabla e^k_i(x)^T}{e^k_i(x)^3}\right)\end{aligned}$$

The algorithm for optimization with adaptive K can be formalized as follows:

Algorithm 4 *This algorithm is identical to Algorithm 2 except for step 3.c and 3.d, which at step k initially defines the local search functions (upon which the global search function is built) as*

$$s^k_{a,i}(x)=\frac{p^k(x)-y_0}{e^k_i(x)}, \tag{56}$$

and at step 3.d, the minimizer of $s^k_{a,i}(x)$ is calculated instead of $s^k_i(x)$. but then, if a point x is encountered during this search for which $p^k(x)<y_0$, subsequently redefines the global search function for step k as $s^k_a(x)=p^k(x)$

Remark 9 Newton's method doesn't always converge to a global minimum. Thus, the result of the search function minimization algorithm at step k, x_k, is not necessarily a global minimizer of $s^k_a(x)$. However, the following properties are guaranteed:

$$\text{if } s^k_a(x)=p^k(x),\quad \text{then } p^k(x_k)\le y_0;$$

$$\text{if } s^k_a(x)=\frac{p^k(x)-y_0}{e^k(x_k)},\quad \text{then} \tag{57a}$$

$$s^k_a(x_k)\le s^k_a\left(\hat{x}^k_j\right)\ \ \forall\Delta^k_j\in\Delta^k. \tag{57b}$$

Recall that $\hat{x}^k_j$ is the maximizer of $e^k_j(x)$ in $\Delta^k_j(x)$.

In the following theorem we prove the convergence of Algorithm 4 to the global minimum of $f(x)$. Convergence is based on the conditions in (57); note that global minimization of the search function $s^k_a(x)$ at each iteration k is not required.

Theorem 5 *Algorithm 4, with the modification described in Algorithm 3 incorporated, will converge to the global minimum of the feasible domain L if $f(x)$ and $p^k(x)$ are twice differentiable functions with bounded Hessian, and all $p^k(x)$ are Lipschitz with the same Lipschitz constant.*

Proof Define S^k, $r^k_{\max}$, and L^k_2 as the set of datapoints, the maximum circumradius of Δ^k, and the maximum edge length of Δ^k, respectively, where Δ^k is a Delaunay triangulation of S^k. Define x_k as the outcome of Algorithm 4, which at step k satisfies (57), and define x'_k as the outcome of Algorithm 3 from input x_k. According to Algorithm 3, $S^{k+1} = S^k \cup \{x'_k\}$. Since $f(x)$ and $p^k(x)$ are twice differentiable with bounded Hessian, constants K_f and K_{pf} exist such that

$$K_f \geq \lambda_{\max}\left(\nabla^2 f(x)\right)/2, \tag{58a}$$

$$K_{pf} \geq \lambda_{\max}\left(\nabla^2\left(f(x) - p^k(x)\right)\right)\Big/2. \tag{58b}$$

Define L_p as a Lipschitz constant of $p^k(x)$ for all steps k of Algorithm 2. Define $y_1 \in S^k$ as the point which minimizes $\delta = \min_{x\in S^k} \|x - x_k\|$.

We will now show that

$$\begin{aligned} &\min_{z\in S^k} f(z) - f(x^*) \leq \bar{\varepsilon}_k, \\ &\bar{\varepsilon}_k = \sqrt{2 r^k_{\max} K_f L_p L^k_2 \delta} + \left[2 r^k_{\max} \max\left\{K_{pf}, K_f\right\} + L_p\right]\delta, \end{aligned} \tag{59}$$

where x^* is a global minimizer of $f(x^*)$.

During the iterations of Algorithm 4, there two possible cases for $s^k_a(x)$. The first case is when $s^k_a(x) = p^k(x)$. In this case, via (57a), $p^k(x_k) \leq y_0$, and therefore $p^k(x_k) \leq f(x^*)$. Since $y_1 \in S^k$, it follows that $p^k(y_1) = f(y_1)$. Moreover, L_p is a Lipschitz constant for $p^k(x)$; therefore,

$$\begin{aligned} p^k(y_1) - p^k(x_k) &\leq L_p\,\delta, \\ f(y_1) - p^k(x_k) &\leq L_p\,\delta, \\ f(y_1) - f(x^*) &\leq L_p\,\delta, \\ \min_{z\in S^k} f(z) - f(x^*) &\leq L_p\,\delta, \end{aligned}$$

which shows that (59) is true in this case.

The other case is when $s^k_a(x) = \frac{p^k(x) - y_0}{e^k(x)}$. For this case, consider Δ^k_j as a simplex in Δ^k which includes x^*. Define $L(x)$ as the unique linear function for which $L(V_i) = f(V_i)$, where V_i are the vertices of the simplex Δ^k_j. According to Lemma 6 and (58a), there is an $\tilde{x} \in \Delta^k_j$

$$L(\tilde{x}_1) - K_f\, e^k(\tilde{x}_1) \leq f(x^*). \tag{60}$$

Since $L(x)$ is a linear function, it takes its minimum value at one of its vertices; thus,

$$\min_{z\in S^k} f(z) - f(x^*) \leq K_f\, e^k(\tilde{x}_1).$$

Recall $\hat{x}^k_j$ is the point in simplex Δ^k_j which maximizes $e^k_j(x)$ inside the closed simplex Δ^k_j; therefore, $e^k(\tilde{x}_1) \leq e^k(\hat{x}^k_j)$, and

$$\min_{z\in S^k} f(z) - f(x^*) \leq K_f\, e^k\left(\hat{x}^k_j\right). \tag{61}$$

On the other hand, according to Lemma 4, $2\,r^k_{\max}$ is the Lipschitz norm for the uncertainty function, $y_1 \in S^k$, and thus $e^k(y_1) = 0$; hence,

$$e^k(x_k) \leq 2\, r^k_{\max}\,\delta.$$

If $e^k(\hat{x}_j^k) \leq e^k(x_k)$,

$$\min_{z \in S^k} f(z) - f(x^*) \leq 2\, r_{\max}^k\, K_f \delta;$$

therefore, (59) is satisfied. Otherwise,

$$\frac{f(x^*) - y_0}{e^k\left(\hat{x}_j^k\right)} \leq \frac{f(x^*) - y_0}{e^k(x_k)}. \tag{62}$$

Using (57b) and (62), it follows:

$$\frac{p^k(x_k) - f(x^*)}{e^k(x_k)} \leq \frac{p^k\left(\hat{x}_j^k\right) - f(x^*)}{e^k\left(\hat{x}_j^k\right)}. \tag{63}$$

According Lemma 6 and (58b), there is an $\tilde{x}_2 \in \Delta_j^k$ such that

$$p^k(\tilde{x}_2) - K_{pf}\, e^k(\tilde{x}_2) \leq f(x^*).$$

Since $\tilde{x}_2 \in \Delta_j^k$ and L_p is a Lipschitz constant for $p^k(x)$, it follows that

$$\begin{aligned} p^k\left(\hat{x}_j^k\right) - L_p L_2^k - K_{pf}\, e^k\left(\hat{x}_j^k\right) &\leq f(x^*), \\ \frac{p^k\left(\hat{x}_j^k\right) - f(x^*)}{e^k\left(\hat{x}_j^k\right)} &\leq K_{pf} + \frac{L_p L_2^k}{e^k\left(\hat{x}_j^k\right)}. \end{aligned} \tag{64}$$

Using (63), (64), and the fact that L_p and $2\, r_{\max}^k$ are Lipschitz constants for $p^k(x)$ and $e^k(x)$, it follows:

$$\begin{aligned} p^k(x_k) - f(x^*) &\leq \left[K_{pf} + \frac{L_p L_2^k}{e^k\left(\hat{x}_j^k\right)} \right] e^k(x_k), \\ p^k(y_1) - f(x^*) &\leq \left[2 r_{\max}^k \left(K_{pf} + \frac{L_p L_2^k}{e^k\left(\hat{x}_j^k\right)} \right) + L_p \right] \delta. \end{aligned}$$

Note that $y_1 \in S^k$; thus, $p^k(y_1) = f(y_1) \geq \min_{z \in S^k} f(z)$, and

$$\min_{z \in S^k} f(z) - f(x^*) \leq \left[2 r_{\max}^k \left(K_{pf} + \frac{L_p L_2^k}{e^k\left(\hat{x}_j^k\right)} \right) + L_p \right] \delta. \tag{65}$$

Using (61) and (65), it follows:

$$\begin{aligned} \min_{z \in S^k} f(z) - f(x^*) \leq & \left[2 r_{\max}^k\, K_{pf} + L_p \right] \delta \\ & + \frac{2 r_{\max}^k\, K_f L_2^k L_p}{\min_{z \in S^k} f(z) - f(x^*)} \delta. \end{aligned}$$

Since x^* is a global minimizer of $f(x)$, $\min_{z\in S^k} f(z) \geq f(x^*)$, and therefore

$$\left[\min_{z\in S^k} f(z) - f(x^*)\right]^2 \leq 2r_{\max}^k K_f L_p L_2^k \delta + \left[2r_{\max}^k K_{pf} + L_p\right]\left[\min_{z\in S^k} f(z) - f(x^*)\right]\delta.$$

Apply the quadratic inequality,[4] and the triangular inequality, it follows:

$$\min_{z\in S^k} f(z) - f(x^*) \leq \sqrt{2r_{\max}^k K_f L_p L_2^k \delta} + \left[2r_{\max}^k K_{pf} + L_p\right]\delta.$$

The above equation implies that (59) is true for this case too. Thus, (59) is true for all possible cases.

Moreover, by construction, S^k is a well situated set; thus, by Theorem 3,

$$r_{\max}^k \leq L_2^k r_1^{n-1}, \tag{66}$$

Furthermore, via Lemma 11,

$$\delta \leq \rho\, \delta_k, \tag{67}$$

where ρ is defined as in (43), and $\delta_k = \min_{x\in S^k} \|x_k' - x\|$. Note that the L_k^2 are bounded, and define L_2 as an upper bound for the L_k^2 for all k. Using (59), (66) and (67), it follows that,

$$\begin{aligned} &0 \leq \min_{z\in S^{\hat{k}}} f(z) - f(x^*) \leq \epsilon_k, \quad \text{where} \\ &\epsilon_k = A\delta_k + B\delta_k, \\ &A = \sqrt{2\rho r_{\max}^k K_f L_p L_2^k}, \\ &B = \rho\left[2r_{\max}^k \max\left\{K_{pf}, K_f\right\} + L_p\right]. \end{aligned}$$

Since the feasible domain is bounded, $\delta_k \to 0$ as $\hat{k} \to \infty$. Moreover, A and B are two constants; therefore, $\epsilon_k \to 0$ as $k \to \infty$; thus, the global convergence of Algorithm 4, with Algorithm 3 incorporated, is assured. □

Remark 10 Globally minimizing the search function $s_a^k(x)$ at each step k is not required in Algorithm 4; it is enough to have (57) to guarantee convergence. In contrast, the search function $s^k(x)$ must be globally minimized in order to guarantee convergence of Algorithm 2.

Since performing Newton's method for minimizing the search function is not required for global convergence, in practice, we will perform Newton's method only in those simplices whose initial points $\hat{x}_j^k$ have small values for the adaptive search function $s_a^k(x)$. In general, performing Newton iterations in more simplices reduces the number of function evaluations required for convergence, but increases the cost of each optimization step.

5.1 Using an inaccurate estimate of y_0

It was shown in Theorem 5 that, using Algorithm 4 with a lower bound y_0 for the function, convergence to a global minimum of the function is guaranteed. However, it is observed

[4] If $A, B, C > 0$, and $A^2 \leq AB + C$ then $A \leq B + \sqrt{C}$.

in Sect. 7 that, if y_0 is not an accurate lower bound for the global minimum, the speed of convergence is reduced. In this subsection, we study the behavior of Algorithm 4, when the estimated value of y_0 is slightly *larger* than the actual minimum of the function of interest.

Theorem 6 *Assume that $f(x)$ and $p^k(x)$ are twice differentiable functions with bounded Hessian, and all $p^k(x)$ are Lipschitz with the same Lipschitz constant. Then, for any small positive $\varepsilon > 0$, there is a finite iteration k of Algorithm 4, with the modification described in Algorithm 3 incorporated, such that a point $z \in S^k$ exists for which:*

$$f(z) - \max\left\{f\left(x^*\right), y_0\right\} \leq \varepsilon, \tag{68}$$

where $f(x^*)$ is the global minimum of $f(x)$.

Proof To prove this theorem, we use the notations defined in the proof of Theorem 5. Note that, if $y_0 \leq f(x^*)$, the theorem is true based on Theorem 5. Otherwise, similar to (59), we will show that

$$\min_{z \in S^k} f(z) - y_0 \leq \sqrt{2r^k_{\max} K_f L^k_2 L_p \delta} + \left[L_p + 2r^k_{\max} K_{pf}\right]\delta. \tag{69}$$

As stated previously, during the iterations of Algorithm 4, there two possible cases for $s^k_a(x)$. The first case is when $s^k_a(x) = p^k(x)$. In this case, via (57a), $p^k(x_k) \leq y_0$. Since $y_1 \in S^k$, it follows that $p^k(y_1) = f(y_1)$. Moreover, L_p is a Lipschitz constant for $p^k(x)$; therefore,

$$\begin{aligned}
&p^k(y_1) - p^k(x_k) \leq L_p\,\delta,\\
&f(y_1) - p^k(x_k) \leq L_p\,\delta,\\
&f(y_1) - y_0 \leq L_p\,\delta,\\
&\min_{z \in S^k} f(z) - y_0 \leq L_p\,\delta,
\end{aligned}$$

which shows that (69) is true in this case.

The other case is when $s^k_a(x) = \frac{p^k(x) - y_0}{e^k(x)}$. In this case, (57b) is satisfied. Now, similar to the proof of Theorem 5, define Δ^k_j as a simplex in Δ^k which includes a global minimizer x^*. Following the same reasoning as in the proof of Theorem 5, (61) is true. Moreover, $y_0 \geq f(x^*)$, and thus

$$\min_{z \in S^k} f(z) - y_0 \leq K_f\, e^k\left(\hat{x}^k_j\right).$$

In addition, if $\min_{z \in S^k} f(z) \leq y_0$, the theorem is trivial, otherwise, the above equation may be modified to

$$\frac{1}{e^k\left(\hat{x}^k_j\right)} \leq \frac{K_f}{\min_{z \in S^k} f(z) - y_0}. \tag{70}$$

Note that (64) is true in this case too. Using (57b), $y_0 \geq f(x^*)$, and the Lipschitz properties of $p^k(x)$ and $e^k(x)$, it follows that

$$\min_{z \in S^k} f(z) - y_0 \leq \left[2r^k_{\max}\left(K_{pf} + \frac{L_p L^k_2}{e^k\left(\hat{x}^k_j\right)}\right) + L_p\right]\delta. \tag{71}$$

Using (70), (71), and $\min_{z\in S^k} f(z) \geq y_0$, it follows that

$$\left[\min_{z\in S^k} f(z) - y_0)\right]^2 \leq 2r_{\max}^k K_f L_p L_2^k \delta + \left[2r_{\max}^k K_{pf} + L_p\right]\left[\min_{z\in S^k} f(z) - f(x^*)\right]\delta.$$

It follows from the above equation that (69) is true for all possible cases. Note that (66) and (67) are true in this theorem too; thus, with similar reasoning, it follows:

$$0 \leq \min_{z\in S^{\hat{k}}} f(z) - y_0 \leq \epsilon_k, \quad \text{where}$$
$$\epsilon_k = A\sqrt{\delta_k} + B\delta_k,$$
$$A = \sqrt{2\rho L_2^2 r_1^{n-1} K_f L_p},$$
$$B = \rho\left[L_p + 2L_2 r_1^{n-1} K_{pf}\right].$$

where $\delta_k = \min_{x\in S^k} \|x'_k - x\|$. Since the feasible domain is bounded, $\delta_k \to 0$ as $k \to \infty$; thus, $\epsilon_k \to 0$ as $k \to \infty$. □

Remark 11 Theorem 6 shows that if an inaccurate guess for the lower bound y_0 of the function $f(x)$ is used, Algorithm 4 will converge to a point whose function value is equal to or below the estimate y_0. In other words, Algorithm 4 will first converge to a point whose function value is less than y_0, and then perform only local refinements thereafter, taking $s^k(x) = p^k(x)$ for later iterations.

6 Parallelization

Parallel computing is one of the most powerful tools of modern numerical methods. In expensive optimization problems, performing an optimization of the type discussed here on a single CPU is relatively slow. The algorithms presented above only accommodate serial computations, with one function evaluation performed in each step.

The present algorithms are intended for problems with expensive function evaluations. Other than these function evaluations, the most expensive part of the present algorithms are the search function minimizations. Constructing the Delaunay triangulation is also somewhat expensive in high-dimensional problems; however, this is made negligible in comparison with the other parts of the algorithm when an Incremental method is used to update the Delaunay triangulation from one step to the next.

The search function minimization allows for straightforward parallel implementation. As described before, we must minimize the local search functions $s_i^k(x)$ [or, in Algorithm 4, $s_{a,i}^k(x)$] within each simplex of the Delaunay triangulation at step k; this task is embarrassingly parallel.

The other expensive part (which will be the most expensive part in many applications) is the function evaluations themselves. We may modify Algorithms 2 and 4 to perform n_p function evaluations in parallel. To accomplish this, we need to identify n_p new points to evaluate at each step.

During Algorithm 2 or 4, x_k is derived by minimizing $s^k(x) = p^k(x) - Ke^k(x)$ or $s_a^k(x) = \frac{p^k(x)-y_0}{e^k(x)}$. Note that, at each step, the uncertainty function $e^k(x)$ is independent from

the interpolation $p^k(x)$ and the function values themselves. Thus, we can modify $e^{k+1}(x)$ without performing the cost function evaluation at x_k. This idea may be implemented as follows

Algorithm 5 *In this algorithm, a modification for Algorithm 2 is presented is which, at each step k, identifies n_p new points to evaluate in parallel at each step. Note that this approach extends immediately to Algorithm 4.*

0. *Take the set of initialization points S^0 as all M of the vertices of the feasible domain L together with one or more user-specified points of interest on the interior of L. Evaluate the function $f(x)$ at each of these initialization points in parallel. Perform a Delaunay triangulation for S^0. Set $k = 0$.*
1. *Calculate (or, for $k > 0$, update) an appropriate interpolating function $p^k(x)$ through all points in S^k.*
2. *Calculate x_k^0 as the minimizer of $s^k(x)$ (see step 3 and 4 of Algorithm 2). This task may be done in parallel for each simplex.*
3. *Replace x_k^0 with the outcome of Algorithm 3 from input x_k^0, then take $S_1^k = S^k \cup \{x_k^0\}$.*
4. *For each substep $i \in \{1, 2, \ldots, n_p - 1\}$, do the following:*
   a. *Incrementally calculate the Delaunay triangulation for data points for S_i^k in order to derive the new uncertainty function $e^{k_i}(x)$.*
   b. *Derive x_k^i as a global minimizer of $s^{k_i}(x) = p^k(x) - K e^{k_i}(x)$.*
   c. *Calculate $\delta_i^k = \min_{y \in S_i^k} \|x_k^i - y\|$, and $\delta_0^k = \min_{y \in S^k} \|x_k^0 - y\|$.*
   d. *If $\delta_i^k \leq c\delta_0^k$ for some c with $0 < c \leq 1$, replace x_k^i with a global minimizer of $e^{k_i}(x)$.*
   e. *Replace x_k^i with the outcome of Algorithm 3 from input x_k^i.*
   f. *Take $S_{i+1}^k(x) = S_i^k \cup \{x_k^i\}$.*
5. *Take $S^{k+1} = S_{n_p}^k$, and evaluate the function at $\left\{x_k^0, x_k^1, \ldots, x_k^{n_p-1}\right\}$ in parallel.*
6. *Repeat from step 1 until $\delta_0^k \leq \delta_{des}$.*

Note that minimizing $s^{k_i}(x)$ for $0 < i < n_p$ is a relatively easy task, since $s^{k_i}(x) = s^{k_{i-1}}(x)$ in most of the simplices, and the incremental implementation of the Delaunay triangulation can be used to flag the indices of those simplices that have been changed by adding x_i^k to $S_{i-1}^k(x)$ (see [36]).

Remark 12 It is easy to show that the modification proposed in Algorithm 5 preserves the convergence properties of Algorithms 2 and 4. The parameter c at step 4.d plays a significant role in the convergence rate; large c forces more of the function evaluations to be related strictly to global exploration, whereas small c allows function evaluations to potentially get dense in regions away from the global minimum. Intermediate values of c are thus preferred, as discussed further in Sect. 7.5.

7 Results

To evaluate the performance of our algorithms, we applied them to the minimization of the some representative test functions (see Appendix.A [26]):

– **One dimension**

Weierstrass function: [5] Taking $N \gg 1$ ($N = 300$ in the simulations performed here),

$$f(x) = \sum_{i=0}^{N} \frac{1}{2^i} \cos(3^i \pi\, x) \tag{72}$$

– **Two dimensions**
Parabolic function:

$$f(x,\ y) = x^2 + y^2 \tag{73}$$

Schwefel fuction: Defining $c = 418.982887$,

$$f(x,\ y) = c - x \sin\left(\sqrt{|x|}\right) - y \sin\left(\sqrt{|y|}\right) \tag{74}$$

Rastrigin function: Defining $A = 2$,

$$\begin{aligned} f(x,\ y) = 2\,A + x^2 + y^2 + \\ -A\cos(2\pi\, x) - A\cos 2\pi\, y \end{aligned} \tag{75}$$

Rosenbrock function: Defining $p = 10$,

$$f(x,\ y) = (1 - x^2) + p\,(y - x^2)^2 \tag{76}$$

– **Higher dimensions**
Rastrigin function: Defining $A = 2$,

$$f(x_i) = A\,n + \sum_{i=1}^{n} \left[x_i^2 - A\cos\left(2\pi\, x_i\right)\right] \tag{77}$$

Rosenbrock function: Defining $p = 10$,

$$f(x_i) = \sum_{i=1}^{n-1} \left[(1 - x_i)^2 + p\left(x_{i+1} - x_i^2\right)^2\right] \tag{78}$$

Except where otherwise stated, each simulation was initialized solely with function evaluations at each vertex of the feasible domain; that is, no user-specified points were used during the initialization. What follows is a description of the results of the simulations performed.

Also, unless otherwise stated, each test result reported incorporates Algorithm 3 into Algorithms 2 and 4, with parameter $r = 2\sqrt{n}$.

7.1 Nondifferentiable 1D case

The Weierstrass function is a classical example of a nondifferentiable function, for which derivative-based optimization approaches are ill-suited. Note that the proofs of convergence of the present algorithms, as developed above, don't even apply in this case; however, it is seen that the algorithms developed converge quickly regardless.

We sought a global minimum of this test function over the domain $[-2, \pi]$. For this test function (only), the set of initial data points was taken as $S^0 = \{-2, 0.5, \pi\}$. The result using Algorithm 2 with $K = 0$ is illustrated in Fig. 6a, the result using Algorithm 2 with $K = 100$

[5] The parameters of the Weierstrass function used in this paper do not satisfy the condition assuring nondifferentiability everywhere that Weierstrass originally identified; however, according to [15], these parameters indeed assure nondifferentiability of the Weiertrass function everywhere as $N \to \infty$.

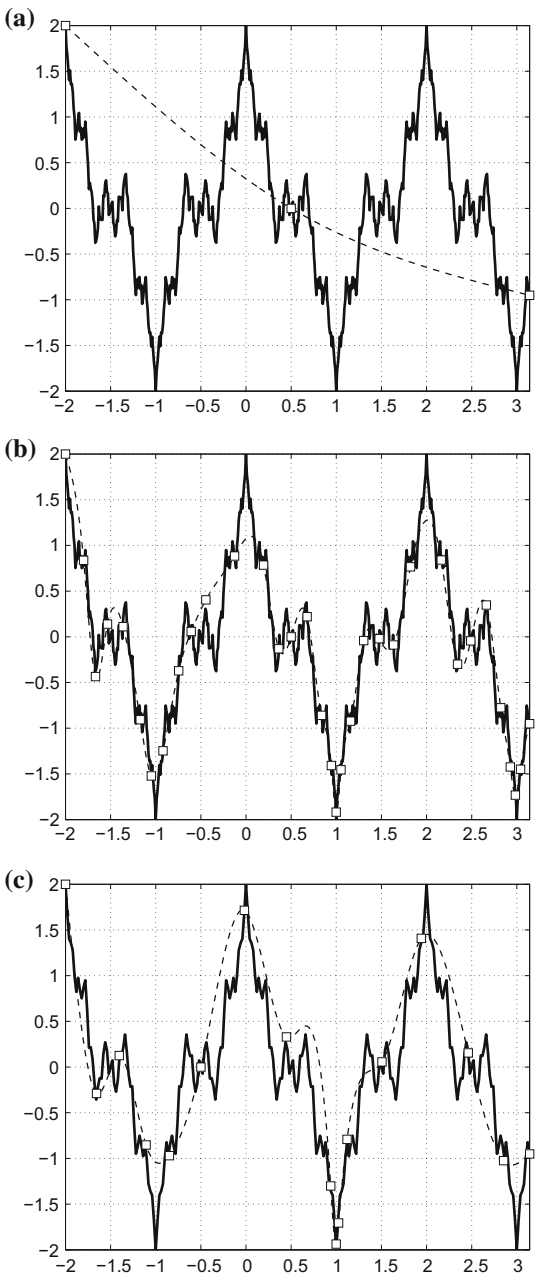

Fig. 6 Implementation of Algorithms 2 and 4 for the Weierstrass function (72). Actual function (*solid*), function evaluations (*squares*), and interplant after the final function evaluation (*dashed*). **a** Algorithm 2 with $K = 0$. **b** Algorithm 2 with $K = 100$. **c** Algorithm 4 with $y_0 = -2$

is illustrated in Fig. 6b, and the result using Algorithm 4 (with adaptive K, taking $y_0 = -2$, which is the known lower bound of $f(x)$) is illustrated in Fig. 6c. The optimizations were terminated when $\min_{x \in S^k} \|x_k - x\| \leq 0.01$.

It is seen in the $K = 0$ case that the optimization routine terminated prematurely, and the global minimum of the problem was not identified; it is thus seen that the global exploration

part of the algorithm (for $K > 0$) is important. It is seen in the $K = 100$ case that global convergence was achieved, and that 34 function evaluations were required for convergence. It is seen in the adaptive K case that global convergence was achieved more rapidly, requiring only 17 function evaluations for convergence.

7.2 Box constraints

7.2.1 2D parabola

The first 2D test function considered is a parabola (73). This simplistic test was made to benchmark the effectiveness of the algorithm on a trivial convex optimization problem. The function considered has a global minimizer at $(0,\ 0)$ in the feasible domain $x_i \in [-\pi,\ 4]$ (the center of the domain is shifted away from the origin to make the minimum nontrivial to find). Again, the optimizations were terminated when $\min_{x \in S^k} \|x_k - x\| \leq 0.01$. For this problem, for both small K ($K = 0.3$, see Fig. 7a) and larger K ($K = 1$, see Fig. 7b), global convergence was achieved; 16 function evaluations were required for convergence of this simple function with $K = 0.3$, and 29 function evaluations were required for convergence with $K = 1$. For the larger value of K, a bit more global exploration is evident. Taking $K = 0$ in the present case again results in premature termination of the optimization algorithm, at the very first step, and the global minimum in not identified

Given exact knowledge of y_0, the behavior of Algorithm 4 for this simple test function is remarkable, and the algorithm converges in only 6 function evaluations (that is, two function evaluation after evaluating the function at the vertices of L). As shown in Fig. 7d, taking y_0 as a bound on the minimum, $y_0 = -0.1$, the algorithm requires a few more iterations (19 function evaluations are needed). In this case, Algorithm 4 actually performs a function evaluation very near the global minimizer within the first two iterations, similar to the case when $y_0 = 0$; however, the algorithm continues to explore a bit more, until it confirms that no other minima with reduced function values exist near this point.

7.2.2 2D Schwefel

The second 2D test function considered is the Schwefel function (74), which is characterized by nine local minima over the domain considered, $x_i \in [0,\ 500]$, with the global minimizer at $(420.968746,\ 420.968746)$, and global minimum of $f(x^*) = 0$. The optimizations were terminated when $\min_{x \in S^k} \|x_k - x\| \leq 1$. As shown in Fig. 8a, for $K = 0.03$, the algorithm fails to converge to the global minimum, as not enough global exploration is performed. As shown in Fig. 8b, taking $K = 0.2$, and thus performing more global exploration, the algorithm succeeds in finding the global minimum after 87 function evaluations. As shown in Fig. 8c, using adaptive K based on accurate knowledge of global minimum $y_0 = 0$, the result is similar to the $K = 0.3$ case, but only 36 function evaluations are performed; convergence is seen to be especially rapid once the neighborhood of the global minimum was identified. As shown in Fig. 8e, using adaptive K based on a bound on the global minimum, $y_0 = -20$, the algorithm continues to explore a bit more, now requiring 64 function evaluations for convergence. As shown in Fig. 8f, using adaptive K based on an inaccurate guess of the bound on the global minimum, $y_0 = 20$, convergence is similar to the case with $y_0 = 0$ (Fig. 8d), with actually a bit faster convergence (now requiring 26 function evaluations for convergence), because global exploration is suspended (that is, K is driven to zero) once function values below y_0 are discovered, with the algorithm thereafter focusing solely on local refinement; recall from Theorem 6 that, in this case, the algorithm may stop any time

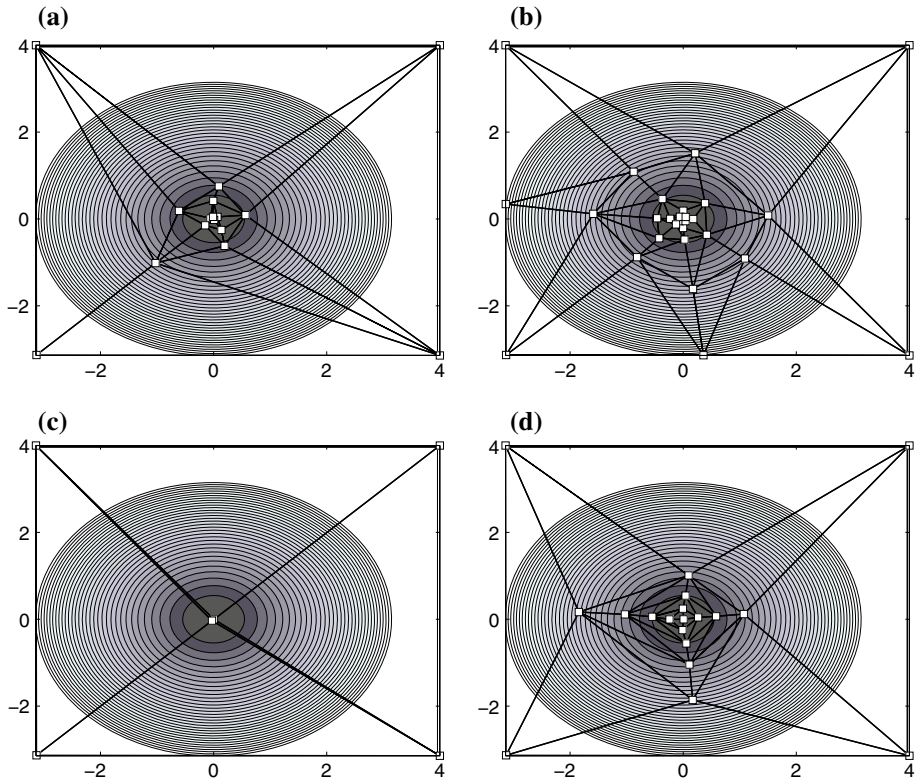

Fig. 7 Location of function evaluations using Algorithms 2 and 4 applied to the 2D parabola (73). **a** Algorithm 2 with $K = 0.3$. **b** Algorithm 2 with $K = 1$. **c** Algorithm 4 with $y_0 = 0$. **d** Algorithm 4 with $y_0 = -0.1$

after function values below y_0 are discovered, and convergence to a neighborhood of the global minimum is not assured.

7.2.3 2D and 3D rastrigin

The third test function considered is the Rastrigin function. We first consider the 2D case (75), which is characterized by 36 local minima over the domain considered, $x_i \in [-2, \pi]$, with the global minimum at the origin. The results for this test function are presented in Figs. 9a–c, and are similar to the Schewefel test case. Algorithm 2 fails to converge to the global minimum when $K = 10$, which is too small for this problem, and more extensive global exploration was performed when $K = 20$, in which case convergence was achieved in 63 function evaluations. More efficient convergence was obtained when Algorithm 4 (adaptive K) was used with an accurate value of $y_0 = 0$, requiring only 30 function evaluations for convergence.

In Fig. 10, we consider the 3D case (77), which is characterized by 216 local minima over the domain considered, $x_i \in [-2, \pi]$, with the global minimum at the origin. We applied Algorithm 4 with an accurate value of $y_0 = 0$, and terminated when $\min_{x \in S^k} \|x_k - x\| \leq 0.01$. During the first iteration after the initialization, a point was obtained with function value close

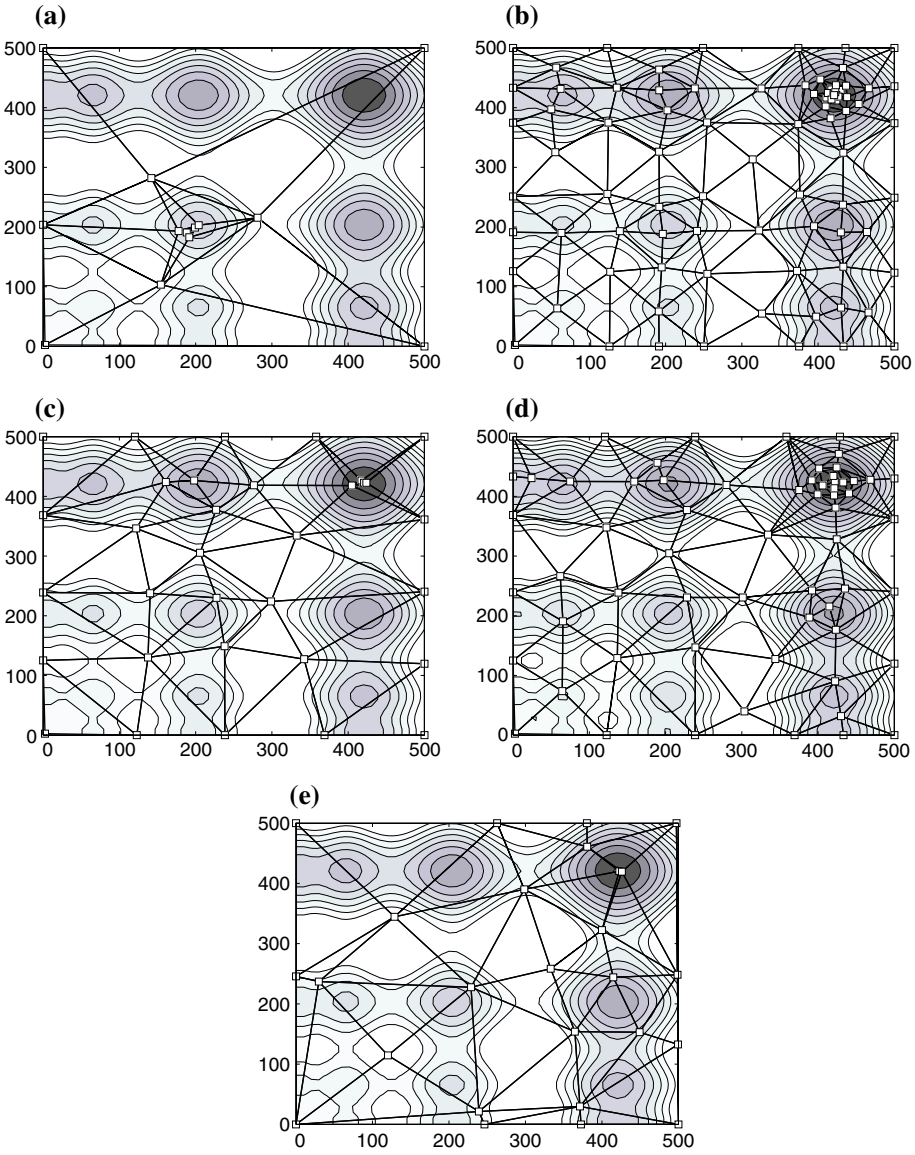

Fig. 8 Location of function evaluations in Algorithms 2 and 4 on the 2D Schwefel function (74). **a** Algorithm 2 with $K = 0.03$. **b** Algorithm 2 with $K = 0.2$. **c** Algorithm 4 with $y_0 = 0$. **d** Algorithm 4 with $y_0 = -20$. **e** Algorithm 4 with $y_0 = 20$.

to the global minimum; however, several more iterations were required until the algorithm stopped after 50 iterations (i.e., including the vertices of L, 58 function evaluations).

7.2.4 2D and 3D rosenbrock

The next test function considered is the Rosenbrock function. We first consider the 2D case (76), which is characterized by a single minimum over the domain considered, $x_i \in [-2\ 2]$,

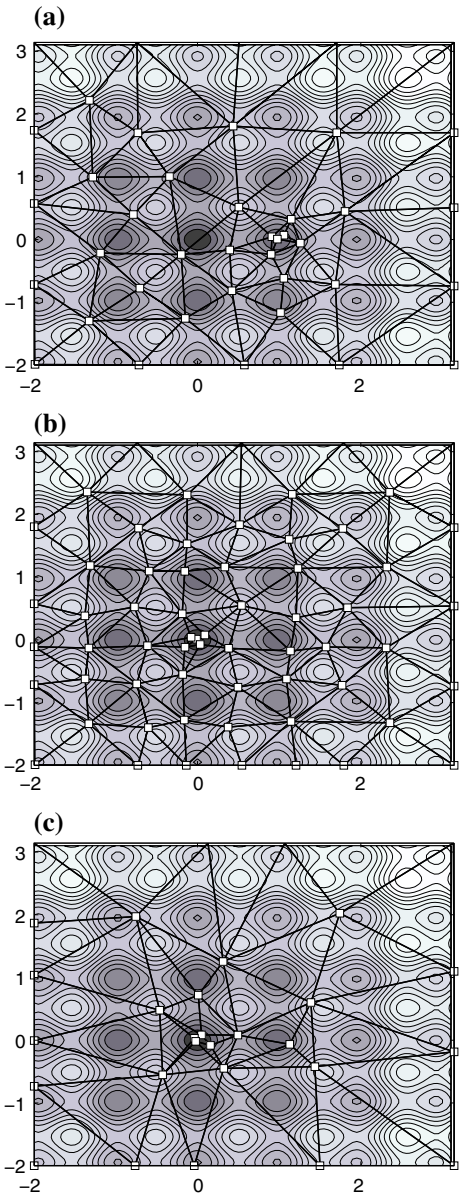

Fig. 9 Location of function evaluations in Algorithms 2 and 4 on the 2D Rastrigin function (75). **a** Algorithm 2 with $K = 10$. **b** Algorithm 2 with $K = 20$. **c** Algorithm 4 with $y_0 = 0$

with the global minimum at $(1,\ 1)$, and a challenging, nearly flat valley along the curve $y = x^2$ where the global minimum lies. In this test function, the optimizations were terminated when $\min_{x \in S^k} \|x_k - x\| \leq 0.01$; since the valley is so flat in this case, the accuracy of the converged solution is a strong function of the termination criterion, and significantly relaxing this criterion leads to inaccurate results. As shown in Fig. 11a, for $K = 5$, the algorithm fails to converge to the global minimum, as not enough global exploration is performed. As shown in Fig. 11b, a larger value of the tuning parameter, $K = 20$, facilitates more thorough global exploration over the domain, with the function evaluations concentrating along the valley,

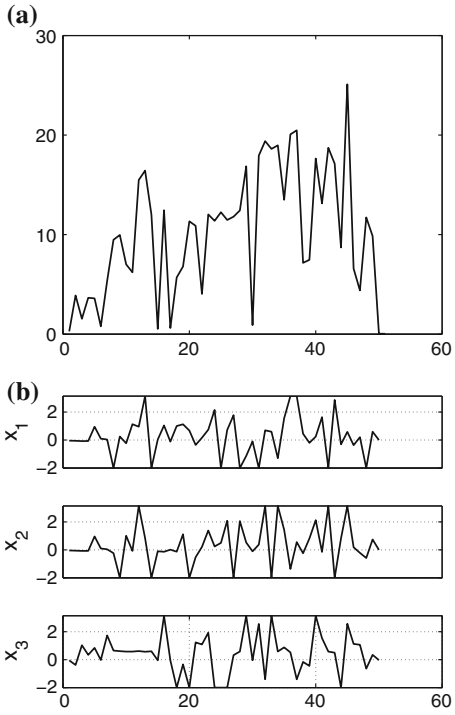

Fig. 10 Implementation of Algorithm 4 with $y_0 = 0$ on the 3D Rastrigin function (77). **a** Function values at the evaluation points. **b** Coordinates of the evaluation points

and convergence is achieved in 70 function evaluations. As shown in Fig. 11c, applying Algorithm 4 (adaptive K) using an accurate value of $y_0 = 0$ focused the function evaluations even better along the valley of the function, and convergence is achieved in only 34 function evaluations.

In Fig. 12, we consider the 3D case (78), which is again characterized by a single minimum over the domain considered, $x_i \in [-2\,2]$, with the global minimum at (1, 1, 1), and a challenging region near the 3D curve $x_3 = x_2^2 = x_1^4$ where the function is nearly "flat". We applied Algorithm 4 with an accurate value of $y_0 = 0$, and terminated when $\min_{x \in S^k} \|x_k - x\| \leq 0.01$. Similar to the 2D case, after a brief exploration of the feasible domain, the algorithm soon concentrates function evaluations near the $x_3 = x_2^2 = x_1^4$ curve where the reduced function values lie, as shown in Fig. 12b, and convergence is achieved in 76 iterations.

7.3 General linear constraints

The above tests were all performed using simple box constraints (1b), $a \leq x \leq b$. We now test the performance of Algorithm 4 with $y_0 = 0$ when more general linear constrains (1a) are applied, $Ax \leq b$.

We first consider the 2D Rastrigin function (75) with the following linear constraints:

$$-2 \leq x, \tag{79a}$$
$$x \leq \pi, \tag{79b}$$
$$-2 \leq y, \tag{79c}$$
$$y \leq \pi, \tag{79d}$$

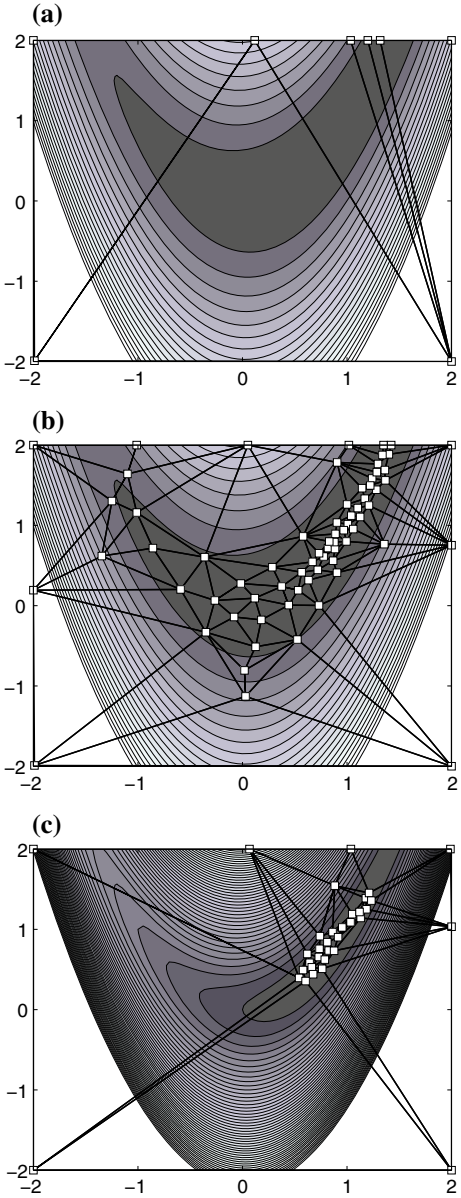

Fig. 11 Location of function evaluations in Algorithms 2 and 4 on the 2D Rosenbrock function (76). **a** Algorithm 2 with $K = 5$. **b** Algorithm 2 with $K = 20$. **c** Algorithm 4 with $y_0 = 0$

$$x \leq y, \tag{79e}$$

$$x + y \leq 1. \tag{79f}$$

During the initialization step, after finding the vertices of L, it is determined that (79b), (79c) and (79d) are redundant. Thus, the feasible domain (in general, a convex polyhedron) is actually a simplex in this case, bounded by (79a), (79e), and (79f). The global minimum in this case lies on the constraint boundary (79e); as shown in Fig. 13, Algorithm 4 converges after initially exploring the feasible domain with 17 function evaluations.

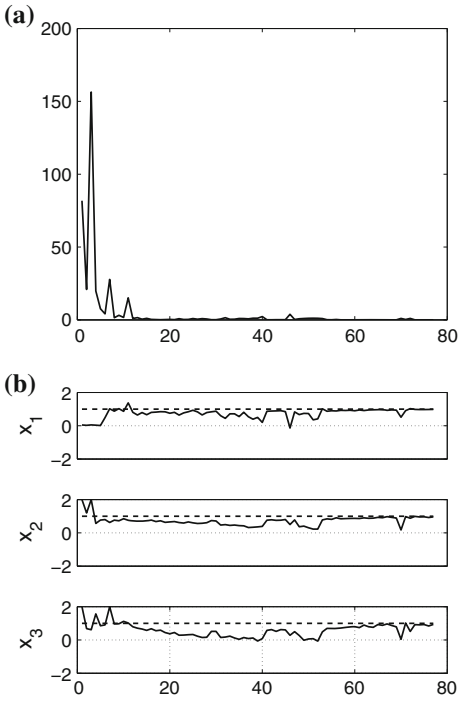

Fig. 12 Implementation of Algorithm 4 with $y_0 = 0$ on the 3D Rosenbrock function (78). **a** Function values at the evaluation points. **b** Coordinates of the evaluation points

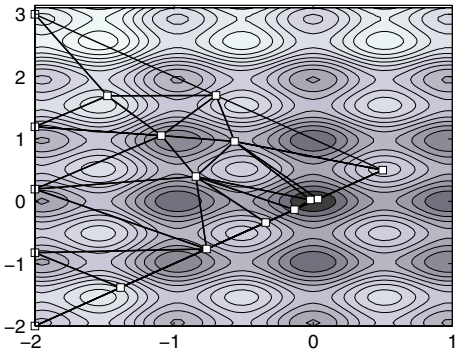

Fig. 13 Rastrigin function in 2D with linear constraints

Next, consider the 2D Rosenbrock function (76) with the following linear constraints:

$$-2 \leq x \leq 2, \tag{80a}$$
$$-2 \leq y \leq 2, \tag{80b}$$
$$-2.2 \leq x + y, \tag{80c}$$
$$x + y \leq 2.2. \tag{80d}$$

During the initialization step, it is determined that none of the constraints are redundant, since each constraint is active at exactly two vertices. As shown in Fig. 14, the feasible domain in this case is a convex polygon with six vertices. The global minimum in this case lies near, but not on, the constraint boundary (80d). As expected, the results are quite similar to the

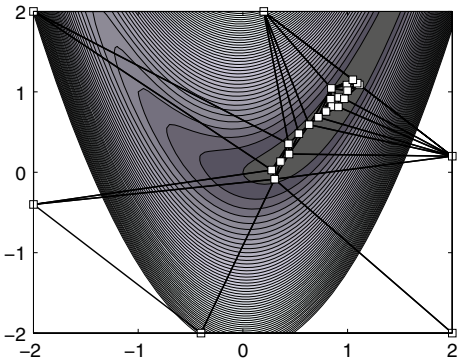

Fig. 14 Rosenbrock function in 2D with linear constraints

case with box constraints (see Fig. 11), and the global minimum is found with 27 function evaluations.

Finally, we considered the 3D Rastrigin and Rosenbrock functions, (77) and (78), with the following linear constraints:

$$-2 \leq x_i \leq 2 \quad \text{for } 1 \leq i \leq 3, \tag{81a}$$

$$x_1 + x_2 + x_3 \leq 3, \tag{81b}$$

$$x_1 - x_2 - x_3 \leq 0. \tag{81c}$$

During the initialization step, it is determined that the constraint $x_1 \leq 2$ is the only redundant constraint, since each of the other constraints is active at at least three of the vertices. The feasible domain in this case is a convex polyhedron with 10 vertices; it turns out that the constraint in (81) is active at six vertices, so one of the faces of this polyhedron is a hexagon. As shown in Figs. 15 and 16, the behavior of Algorithm 4 is similar to the corresponding tests with box constraints discussed previously. Note, of course, that all function evaluations performed by Algorithm 4 are evaluated at feasible points.

7.4 Feasible constraint projections

We now explore the role of the feasible boundary projections introduced in Definition 5, and incorporated into Algorithm 3, on the convergence of Algorithm 2 with $K = 1$, focusing specifically on the impact of the r parameter, taking $r = 1.05$, $r = 5$ and $r = 30$. We perform this test using the 2D parabolic function (73) subject to the following linear constraints:

$$x \leq 0.1, \tag{82a}$$

$$-1.1 \leq y, \tag{82b}$$

$$y - x \leq 0.5. \tag{82c}$$

The location of the function evaluations for different values of r is shown in Fig. 17.

Recall that $1 < r < \infty$, with the $r \to \infty$ limit suppressing all feasible constraint projections. It is observed that, for small values of r, the algorithm tends to explore more on the boundaries of the feasible domain, and for large values of r, the triangulation is more irregular, with certain function evaluations clustered in a region far from the global minimum. Intermediate values of r are thus preferred.

Figure 18 plots the maximum circumradius $r^k_{\max}$ of the Delaunay triangulation Δ^k, as well as the upper bound for $r^k_{\max}$, during the optimization using Algorithm 2 with Algorithm 3

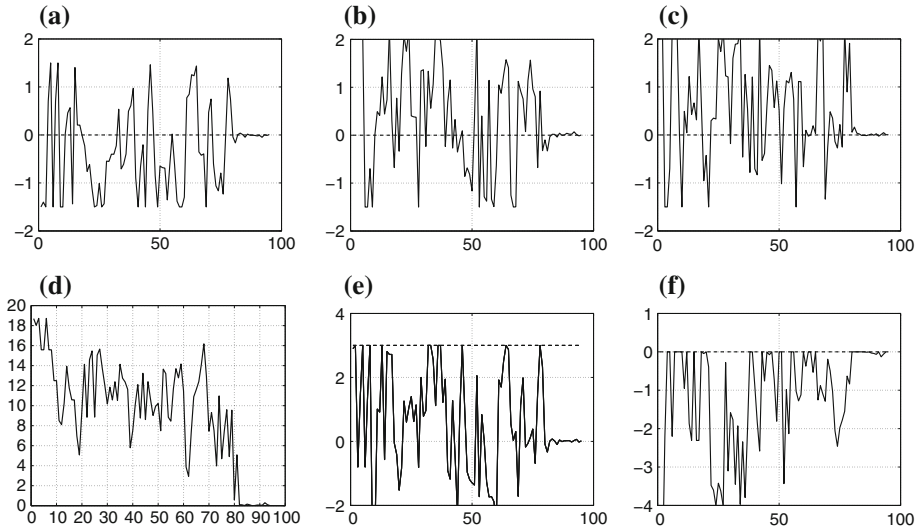

Fig. 15 Implementation Algorithm 4 on the 3D Rastrigin function (77) with linear constraints. The *first row* shows the location of the function evaluations, and the *second row* shows the actual function values, as well as the "slack" distance of the function evaluations from the two constraints of L that are not simple bound constraints. **a** x_1 coordinate, **b** x_3 coordinate, **c** Cost function values, **d** $x_1 + x_2 + x_3$, **e** $x_1 - x_2 - x_3$

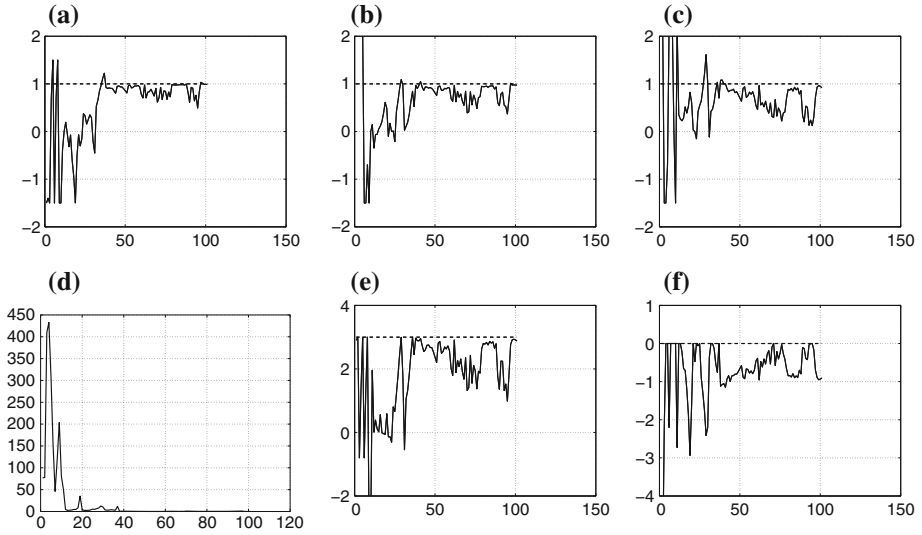

Fig. 16 As in Fig. 15 for the 3D Rosenbrock function (78). **a** x_1 coordinate, **b** x_2 coordinate, **c** x_3 coordinate, **d** Cost function values, **e** $x_1 + x_2 + x_3$, **f** $x_1 - x_2 - x_3$

incorporated using different values of r. The maximum circumradius $r^k_{\max}$ is seen to be reduced when smaller values of r are used; however, the cases with $r = 1.05$ and $r = 5$ are not noticeably different in this respect. Another important observation is that the bound on the maximum circumradius, given by (51), is also reduced when smaller values of r are used; however, this bound is seen to be quite conservative in this example.

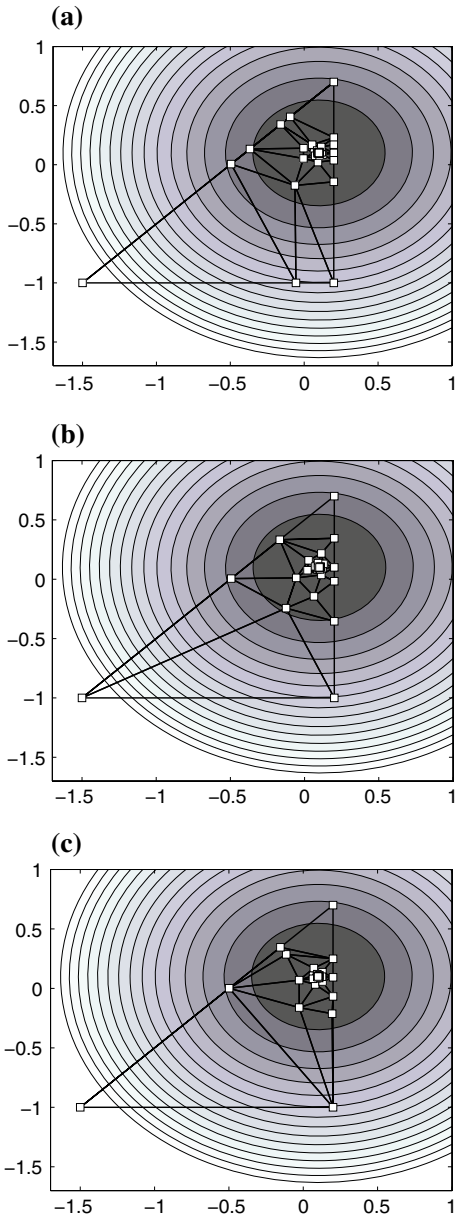

Fig. 17 The location of function evaluations by Algorithm 2, with Algorithm 3 incorporated, for different values of r. The cost function is simple quadratic function whose minimizer is an interior point within a feasible domain given by a *right triangle*. **a** Implementation for $r = 1.05$. **b** Implementation for $r = 5$. **c** Implementation for $r = 30$

7.5 Parallel performance

We now test the performance of the constant-K version of Algorithm 5 on the Weierstrass function (72), over the domain $[-2, \pi]$ using $n_p = 3$ processors, taking $K = 15$ and, in turn, $c = 0, c = 0.5$ and $c = 1$. The optimizations were terminated when $\min_{x \in S^k} \|x_k - x\| \leq 0.01$. Algorithm 5 fails to converge to the global minimum when $c = 0$, as multiple function evaluations are performed at a single step k that are close to each other in this case, which

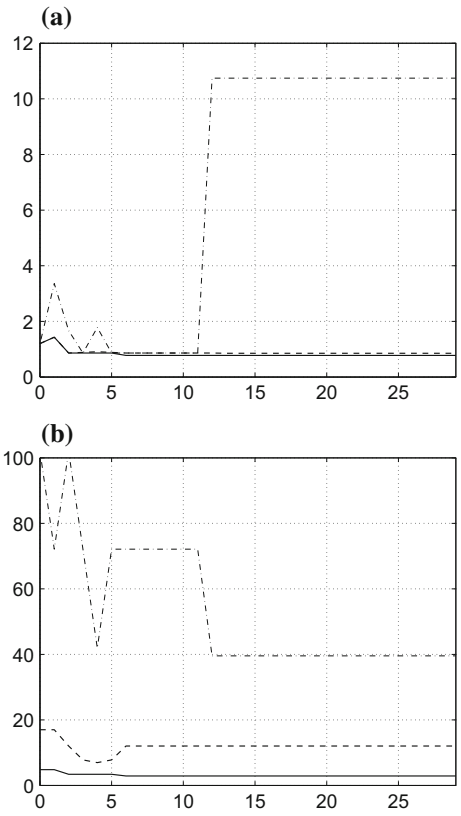

Fig. 18 The actual value and the theoretical bound for the maximum circumradius $r^k_{\max}$ of Δ^k, as a function of k, for the optimizations illustrated in Fig. 17. *Solid line*, *dashed line*, and *dot–dashed line* are the results for $r = 1.05$, $r = 5$ and $r = 30$ respectively. **a** Actual value of $r^k_{\max}$. **b** Theoretical upper bound for $r^k_{\max}$

causes premature termination of the algorithm. Algorithm 5 converges to the global minimum for $c = 0.5$ in 18 function evaluations, and for $c = 1$ in 21 function evaluations; it is thus seen that intermediate values of c are preferred. In the $c = 0.5$ case, after the initialization, 6 iterations were executed, with 7 function evaluations performed in parallel during each iteration.

Testing Algorithm 2 with $K = 15$ on the same problem, it is seen that fewer (in this case, 13) function evaluations are required by the serial version of the algorithm, as the location of each new function evaluation is based on the trends revealed in all of the previous function evaluations. However, the total number of iterations that need to be executed in this case is increased from 6 to 10, thus demonstrating the benefit of the parallel algorithm when performing function evaluations in parallel is possible (Fig. 19).

8 Conclusions

In this paper, we have presented a new response surface method for derivative-free optimization of nonconvex functions within a feasible domain L bounded by linear constraints. The paper developed five algorithms:

- Algorithm 1 showed how to initialize the problem, identifying the vertices of L, eliminating the redundant constraints, and projecting the equality constraints out of the problem.

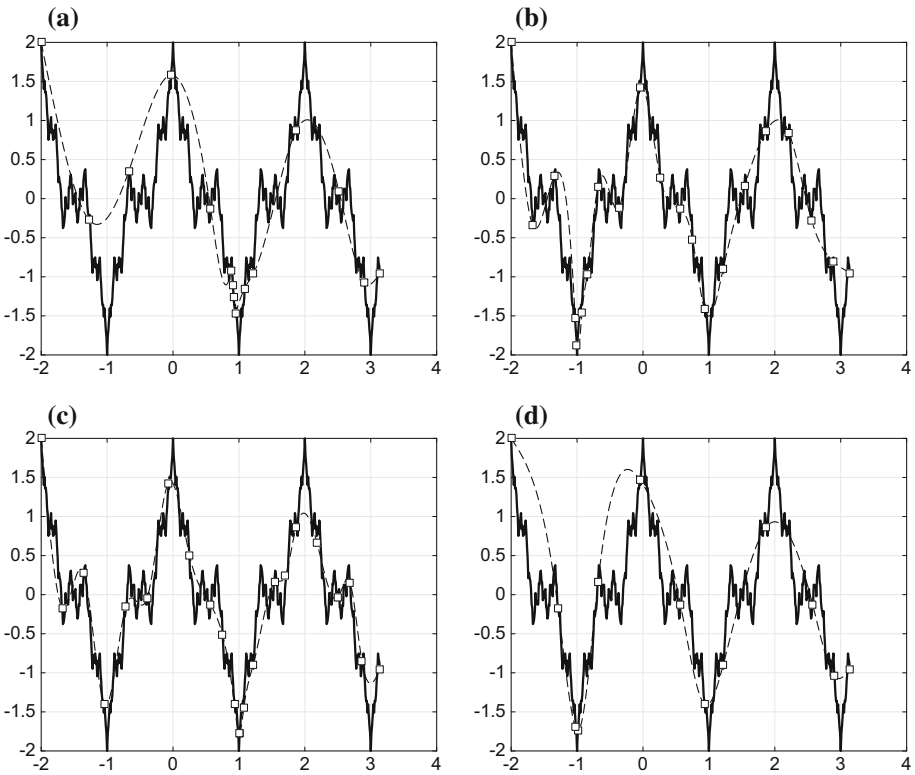

Fig. 19 Comparison of Algorithm 5, for various values of c, and Algorithm 2, all with $K = 15$, on the Weierstrass function considered previously. Evaluated points (*squares*), function of interest (*solid line*), interpolation at last step (*dashed line*). **a** Parallel Algorithm with $c = 0, n_p = 3$. **b** Parallel Algorithm with $c = 0.5$, $n_p = 3$. **c** Parallel Algorithm with $c = 1$, $n_p = 3$. **d** Serial Algorithm

- Algorithm 2 presented the essential strawman form of the method. It uses any well-behaved interpolation function of the user's choosing, and a synthetic piecewise-quadratic uncertainty function built on the framework of a Delaunay triangulation. A search function given by a linear combination of the interpolation and a model of the uncertainty is minimized at each iteration. The search function itself is piecewise smooth, and may in fact be nonconvex within certain simplices of the Delaunay triangulation. A valuable feature of the algorithm is that global minimization of the search function within each simplex is, in fact, not required at each iteration; convergence to the global minimum can be guaranteed even if the algorithm only locally minimizes the search function within each simplex at each iteration. Unfortunately, this simple algorithm contains an important technical flaw: it does not ensure that the triangulation remains well behaved as new datapoints are added.
- Algorithm 3 showed how to correct the technical flaw of Algorithm 2 by performing feasible constraint projections, when necessary, to ensure that the triangulation remains well behaved, with bounded circumradii, as new datapoints are added.
- Algorithm 4 showed how to use an estimate for the lower bound of the function to maximally accelerate local refinement while still ensuring convergence to the global minimum.

- Algorithm 5 showed how to efficiently parallelize the function evaluations on n_p processors at each step of the algorithm.

Four adjustable parameters were identified in the above algorithms, and their effects quantified in numerical tests:

- Algorithm 2 introduced a tuning parameter $K > 0$, which governs the balance between global exploration and local refinement. For sufficiently large K applied to smooth functions (that is, Lipschitz, twice-differentiable, and bounded Hessian), convergence to a neighborhood of the global minimum is guaranteed in a finite number of iterations. For larger values of K, exploration becomes essentially uniformly over L.
- Algorithm 3 introduced a tuning parameter $r > 1$ which controls how frequently feasible constraint projections are performed. Small values of r leads to function evaluations accumulating on the boundaries of L, and a more uniform triangulation with reduced maximum circumradius. Large values of r leads to fewer function evaluations on the boundaries of L, and less uniform triangulations. Intermediate values of r are thus preferred.
- Algorithm 4 uses a tuning parameter y_0, which is an estimate for the global minimum. Convergence of Algorithm 4 was found to be remarkably rapid when y_0 was an accurate estimate of the global minimum, both for smooth functions, and even certain nonsmooth functions, like Weierstrass, characterized by exploitable trends of the global shape of function. (For problems without such exploitable trends, the algorithm was well behaved, exploring essentially uniformly over the feasible domain.) When y_0 is less than the global minimum, convergence of Algorithm 4 to a global minimizer is guaranteed, though more global exploration is typically performed in the process. When y_0 is greater than the global minimum, convergence of Algorithm 4 to a value less than or equal to y_0 is guaranteed, with some local refinement performed thereafter.
- Algorithm 5 uses a tuning parameter c which controls how much closer evaluation points are allowed to get during the parallel substeps of the iteration. Global convergence is guaranteed for $0 < c \leq 1$. Small values of c allow function evaluations to get dense far from the global minimum during the parallel substeps, and slows convergence. Large values of c force the algorithm to focus primarily on global exploration, again slowing convergence. Intermediate values of c are thus preferred.

The algorithms described above were tested in Matlab, and Python and C++ implementations of these algorithms are currently being developed; for more information regarding availability of these codes, please contact the authors via email. Of course, as with any derivative-free optimization algorithm, there is a curse of dimensionality, and optimization in only moderate dimensional problems (i.e., $n < 10$) is expected to be numerically tractable; a key bottleneck of the present code as the dimension of the problem increases is the overhead required with the enumeration of the triangulation. The parallel version of the algorithm is expected to be particularly efficient in cases requiring substantial global exploration; in cases focusing primarily on local refinement, the speed up provided by performing function evaluations in parallel is anticipated to be reduced.

In Part 2 of this work, we extend the algorithms developed here to convex domains bounded by arbitrary convex constraints. In Part 3 of this work, we extend the algorithms developed here to approximate function evaluations, each of which is obtained via statistical averaging over a finite number of samples. The numerical results presented in the present paper are intended to exhibit a proof of concept of the behavior of the algorithms developed herein; future papers will consider more practical applications, and compare to other leading global optimization algorithms available in the literature.

Acknowledgments The authors gratefully acknowledge funding from AFOSR FA 9550-12-1-0046, from the Cymer Center for Control Systems & Dynamics, and from Leidos corporation in support of this work.

Appendix: Polyharmonic splines

The algorithms described above require the gradient and Hessian of the interpolant being used to facilitate Newton-based minimizations of the search function. Since our numerical tests all implement the polyharmonic spline interpolation formula, we now derive analytical expressions of the gradient and Hessian in this case.

The polyharmonic spline interpolation $p(x)$ of a function $f(x)$ in $\mathbb{R}^n$ is defined as a weighted sum of a set of radial basis functions $\varphi(r)$ built around the location of each evaluation point, plus a linear function of x:

$$p(x) = \sum_{i=1}^{N} w_i\, \varphi(r) + v^T \begin{bmatrix} 1 \\ x \end{bmatrix},$$
$$\text{where} \quad \varphi(r) = r^3 \quad \text{and} \quad r = \|x - x_i\|. \tag{83}$$

The weights w_i and v_i represent N and $n+1$ unknowns, respectively, to be determined through appropriate conditions. First, we match the interpolant $p(x)$ to the known values of $f(x)$ at each evaluation point x_i, i.e. $p(x_i) = f(x_i)$; this gives N conditions. Then, we impose the orthogonality conditions $\sum_i w_i = 0$ and $\sum_i w_i x_{ij} = 0, \quad j = 1, 2, \ldots, n$. This gives $n+1$ additional conditions. Thus,

$$\begin{bmatrix} F & V^T \\ V & 0 \end{bmatrix} \begin{bmatrix} w \\ v \end{bmatrix} = \begin{bmatrix} f(x_i) \\ 0 \end{bmatrix} \quad \text{where}$$
$$F_{ij} = \varphi(\|x_i - x_j\|) \quad \text{and}$$
$$V = \begin{bmatrix} 1 & 1 & \ldots & 1 \\ x_1 & x_2 & \ldots & x_N \end{bmatrix}. \tag{84}$$

The gradient and Hessian of $p(x)$ may now be written as follows:

$$\nabla p(x) = \nabla \left(\sum_{i=1}^{N} w_i \|x - x_i\|^3 + v^T \begin{bmatrix} 1 \\ x \end{bmatrix} \right)$$
$$= 3 \sum_{i=1}^{N} w_i \|x - x_i\| (x - x_i) + \bar{v},$$

where $\bar{v} = [v_2, v_3, \ldots, v_{n+1}]^T$, and

$$\nabla^2 p(x) = \nabla^2 \left(\sum_{i=1}^{N} w_i \|x - x_i\|^3 + v^T \begin{bmatrix} 1 \\ x \end{bmatrix} \right)$$
$$= 3 \sum_{i=0}^{N} w_i \left(\frac{(x - x_i)(x - x_i)^T}{\|x - x_i\|} + \|x - x_i\| I_{n \times n} \right).$$

Note that the calculation of the weights of a polyharmonic spline interpolant requires the solution of a $(N + n + 1) \times (N + n + 1)$ linear system. This system is not diagonally dominant, and does not show an easily-exploitable sparsity pattern facilitating fast factorization

techniques. Nevertheless, since our algorithm adds only one point to the set of N evaluation points at each iteration, we can avoid the solution of the new linear system from scratch, and instead implement a rank-one update at each iteration as follows. First, for the set of initial points, we calculate the inverse $A = \left[\begin{smallmatrix} F & V^T \\ V & 0 \end{smallmatrix}\right]$. This step is somewhat time consuming, but reduces the computations required in subsequent steps. Using Matrix Inversion Lemma, we then update the inverse of A with the new information given at each step as follows:

$$A_{N+1}^{-1} = \begin{bmatrix} A_N & b^T \\ b & 0 \end{bmatrix}^{-1} = \begin{bmatrix} A_N^{-1} + A_N^{-1} b^T b A_N^{-1}/c & -A_N^{-1} b^T/c \\ -b A_N^{-1}/c & 1/c \end{bmatrix}, \tag{85}$$

where b is a vector of length $n+1$ defined as $b = \left[\begin{smallmatrix} 1 & x_{N+1} \end{smallmatrix}\right]^T$, and $c = -b A_N^{-1} b^T$ is a scalar. Multiplication of A_{N+1}^{-1} in (85) with the vector $\left[\begin{smallmatrix} f(x_i) & 0 & f(x_{N+1}) \end{smallmatrix}\right]^T$ gives the vector of weights in an unordered fashion, i.e. $\left[\begin{smallmatrix} w_i & v_i & w_{N+1}) \end{smallmatrix}\right]^T$. Therefore, before adding the new function evaluation in the following iteration and performing the next rank-one update, it is necessary to permute the matrix A_{N+1}^{-1}, given by

$$A_{N+1}^{-1} = \begin{bmatrix} F & V^T & \varphi(\|x_{N+1} - x_i\|)^T \\ V & 0 & \left[\begin{smallmatrix} 1 & x_{N+1} \end{smallmatrix}\right] \\ \varphi(\|x_{N+1} - x_i\|) & \left[\begin{smallmatrix} 1 \\ x_{N+1} \end{smallmatrix}\right] & 0 \end{bmatrix}^{-1},$$

such that the desired 2×2 block form at the next iteration is recovered:

$$\begin{aligned} P A_{N+1}^{-1} P^T &= \begin{bmatrix} F_+ & V_+^T \\ V_+ & 0 \end{bmatrix}^{-1} \\ &= \begin{bmatrix} F & \varphi(\|x_{N+1} - x_i\|)^T & V^T \\ \varphi(\|x_{N+1} - x_i\|) & 0 & \left[\begin{smallmatrix} 1 & x_{N+1} \end{smallmatrix}\right] \\ V & \left[\begin{smallmatrix} 1 \\ x_{N+1} \end{smallmatrix}\right] & 0 \end{bmatrix}^{-1}. \end{aligned}$$

After this permutation, it is possible to apply the Matrix Inversion Lemma (85) at the following step.

Remark 13 Another fast method to find the coefficients of radial basis functions is described in [35]. Since the present algorithms build the dataset incrementally, the method described above is less expensive in the present case.

As mentioned earlier, variations of Kriging interpolation are often used in Response Surface Methods, such as the Surrogate Management Framework, for derivative-free optimization. DACE (see [8]) is one of the standard packages used for numerically computing the Kriging interpolant. Figure 20a and b compare of the polyharmonic spline interpolation method described above and the Kriging interpolation method computed using DACE, as applied to the test function $f(r) = r * \sin 1/r$, where $r^2 = x^2 + y^2$ with $N = 1004$ data points. The data points used in this example are the 4 corners of a square domain, and 1000 random-chosen points clustered within a small neighborhood of the center of the square, which highlights the numerical challenge of performing interpolation when grid points begin to cluster in a particular region, which is common when a response surface method for derivative-free optimization approaches convergence. Figure 20a and b plot the difference between the real value of f and the value of the corresponding interpolants.

An observation which motivated the present study is that, in such problems, the Kriging interpolant is often spurious in comparison with other interpolation methods, such as polyharmonic splines. Note that various methods have been proposed to regularize such spurious

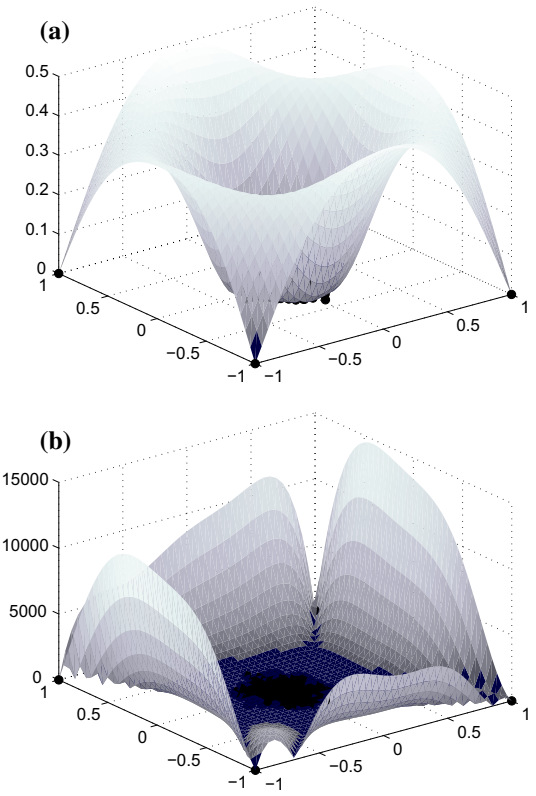

Fig. 20 The difference between the actual function, $f(r) = r * \sin 1/r$, where $r^2 = x^2 + y^2$, and its interpolant for two different interpolation strategies when 1000 function evaluations are clustered near the *center* of a *square* domain. **a** The error of the polyharmonic spline interpolation interpolant (83). **b** The error of the Kriging interpolant with a Gaussian model for the corrrelation, computed using DACE

interpolations in the presence of clustered datapoints, such as combining interpolants which summarize global trends with interpolants which account for local fluctuations. Our desire in the present effort was to develop a robust response surface method that can implement any good interpolation strategy, the selection of which is expected to be somewhat problem dependent.

References

1. Alexandria, D.A.: Convex Polyhedra. Springer, Berlin (2005)
2. Balinski, M.L.: An algorithm for finding all vertices of convex polyhedral sets. J. Soc. Ind. Appl. Math. **9**(1), 72–88 (1961)
3. Belitz, P., Bewley, T.: New horizons in sphere-packing theory, part II: lattice-based derivative-free optimization via global surrogates. J. Glob. Optim. **56**(1), 61–91 (2013)
4. 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. Struct. Optim. **17**(1), 1–13 (1999)
5. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press, Cambridge (2004)
6. Hornus, S., Boissonnat, J.D.: An Efficient Implementation of Delaunay Triangulations in Medium Dimensions, [research report] RR-6743 (2008)
7. Boissonnat, J.D., Devillers, O., Hornus, S.: Incremental construction of the Delaunay triangulation and the Delaunay graph in medium dimension. In: Proceedings of the Twenty-Fifth Annual Symposium on Computational Geometry. ACM (2009)
8. Nielsen, H.B., Lophaven, S.N., Sndergaard, J., DACE, A.: A matlab kriging toolbox. In: Technical Report, Technical University of Denmark, Version 2.0, 1 Aug (2002)

9. Dwyer, R.A.: A faster divide-and-conquer algorithm for constructing delaunay triangulation. Algorithmica **2**(1–4), 137–151 (1987)
10. Dwyer, R.A.: Higher-dimensional Voronoi diagram in linear exprected time. Discrete Comput. Geom. **6**(1), 343–367 (1991)
11. Dwyer, R.A.: The expected number of k-faces of Voronoi diagram. Comput. Math. Appl. **26**(5), 13–19 (1993)
12. George, P.L., Borouchaki, H.: Delaunay Triangulation and Meshing: Application to Finite Element. Hermes, Paris (1998)
13. Gill, P.E., Murray, M.: Newton-type methods for unconstrained and linearly constrained optimization. Math. Progr. **7**(1), 311–350 (1974)
14. Gutmann, H.M.: A radial basis function method for global optimization. J. Glob. Optim. **19**(3), 201–227 (2001)
15. Hardy, G.H.: Weierstrass as non differentiable function. Trans. Amer. Math. Soc. **17**(3), 301–325 (1916)
16. Hoffman, K.L.: A method for globally minimizing concave functions over convex sets. Math. Progr. **20**(1), 22–32 (1981)
17. Jones, D.: A taxonomy of global optimization methods based on response surfaces. J. Glob.Optim. **21**, 345–383 (2001)
18. Krige, D.G.: A Statistical Approach to Some Mine Valuations and Allied Problems at the Witwatersrand. Masters thesis of the University of Witwatersrand, South Africa (1951)
19. Lewis, R.M., Torczon, V., Trosset, M.W.: Direct Search Method: Then and Now, NASA/CR-2000-210125, ICASE Report No.2000-26 (2000)
20. Li, X.Y.: Generating well-shaped d-dimensional Delaunay meshes. Theor. Comput. Sci. **296**(1), 145–165 (2003)
21. Li, X-Y, Teng, S.H.: Generating well-shaped Delaunay meshed in 3D. In: Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete algorithms. SIAM (2001)
22. Manas, M., Nedoma, J.: Finding all vertices of a convex polyhedron. Numer. Math. **12**(3), 226–229 (1968)
23. Matheiss, T.H., Rubin, D.S.: A survey and comparison of methods for finding all vertices of convex polyhedral sets. Math. Oper. Res. **5**(2), 167–185 (1980)
24. Matheron, B.: Principles of geostatistics. Econ. Geol. **58**(8), 1246–1266 (1963)
25. McMullen, P.: The maximum numbers of faces of a convex polytope. Mathematika **17**(02), 179–184 (1970)
26. Yang, X.: Nature-inspired optimization algorithms. Elsevier (2014)
27. Nocedal, J., Wright, S.J.: Numerical Optimization. Springer, Berlin (1999)
28. Powell, M.J.D.: An efficient method for finding the minimum for function of several variables without calculating derivatives. Comput. J. **7**(2), 155–162 (1964)
29. Rosenbrock, H.H.: An automatic method for finding the greatest or least value of a function. Comput. J. **3**(3), 175–184 (1960)
30. Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning. MIT Press, Cambridge (2006)
31. 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)
32. Shewchuk, J.R.: Delaunay refinement algorithms for triangular mesh generation. Comput. Geom. **22**(1), 21–74 (2002)
33. Spendley, W., Hext, G.R., Himsworth, F.R.: Sequential application of simplex designs in optimisation and evolutionary operation. Technometrics **4**(4), 441–461 (1962)
34. Torczon, V.: Multi-Directional Search, A Direct Search Algorithm for Parallel Machines, Ph.D. thesis, Rice University, Houstin, TX (1989)
35. Wahba, G.: Spline models for observational data, Vol. 59. Siam (1990)
36. Watson, D.: Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. Comput. J. **24**(2), 167–172 (1981)
37. Torczon, V.: On the convergence of pattern search algorithms. SIAM J. Optim. **7**(1), 1–25 (1997)
38. Torn, A., Zilinkas, A.: Global Optimization. Springer, New York (1989)
39. http://www.qhull.org. Accessed 31 March 2015
40. http://netlib.org/voronoi/hull.html. Accessed 31 March 2015
41. http://www.cgal.org. Accessed 31 March 2015