REFINING SIMPLEX POINTS FOR SCALABLE ESTIMATION OF THE LEBESGUE CONSTANT

Albert Jiménez-Ramos Abel Gargallo-Peiró Xevi Roca

Barcelona Supercomputing Center, Barcelona, Spain. xevi.roca@bsc.es

ABSTRACT

To estimate the Lebesgue constant, we propose a point refinement method on the d-dimensional simplex. The proposed method features a smooth gradation of the point resolution, neighbor queries based on neighbor-aware coordinates, and a point refinement that algebraically scales as $(d+1)\,d$. Remarkably, by using neighbor-aware coordinates, the point refinement method is ready to automatically stop using a Lipschitz criterion. For different polynomial degrees and point distributions, we show that our automatic method efficiently reproduces the literature estimations for the triangle and the tetrahedron. Moreover, we efficiently estimate the Lebesgue constant in higher dimensions. Accordingly, up to six dimensions, we conclude that the point refinement method is well-suited to efficiently estimate the Lebesgue constant on simplices. In perspective, for a given polynomial degree, the proposed point refinement method might be relevant to optimize a set of simplex points that guarantees a small interpolation error.

Keywords: optimization, simplices, adaptivity, point refinement, Lebesgue constant

1. INTRODUCTION

In approximation theory, one of the key problems is obtaining a set of simplex points for a given polynomial degree that guarantees a small interpolation error. To solve this problem, it is standard to obtain interpolation points featuring a small *Lebesgue constant*. This constant is usually denoted by Λ and defined as

$$\Lambda = \max_{\boldsymbol{x}\in K^d} \sum_{i=1}^{N_p} \left|\phi_i\left(\boldsymbol{x}\right)\right|. \tag{1}$$

It corresponds to the maximum on the d-dimensional simplex K^d of the summation of the absolute values of the lagrangian functions ϕ_i associated with each of the N_p interpolation points, a summation that is Lipschitz because the absolute value terms are Lipschitz, a summation that is called the *Lebesgue function*. The maximum of this function appears in the upper bound of the interpolation error. Specifically, given a point distribution, for any function f, the error of the polynomial interpolation satisfies

$$\left\|f - I\left(f\right)\right\| \leq \left(1+\Lambda\right)\left\|f - p^\star\right\|,$$

where $I\left(f\right)$ denotes the lagrangian interpolator, and $p^\star$ the best polynomial approximation. According to the previous inequality, the smaller the Lebesgue constant, the smaller the bound of the interpolation error. Moreover, the Lebesgue

constant exclusively depends on the position of the interpolation points. For instance, equispaced distribution of points, which lead to larger errors for higher polynomial degrees, feature large values of the Lebesgue constant, yet non-uniform distribution of points, with improved interpolation error, feature sub-optimal values of the Lebesgue constant [1, 2, 3]. Accordingly, to guarantee small interpolation errors, the Lebesgue constant has to be evaluated and thus, it is key to estimate on the simplex the maximum of the Lebesgue function.

To approximate this maximum, it is critical to automatically generate on the d-dimensional simplex a finite number of sample points – exactly the goal of this work. Then, for those points, the approximation is the maximum of the function evaluations. Note that these sample points are used to estimate the Lebesgue constant, but they do not correspond to the interpolation points that define the lagrangians in Eq. (1). To estimate the Lebesgue constant, there are general zeroth-order optimization [4] and Lebesgue-specific [2, 3, 5] methods. These approaches are mainly different because the latter family exploits the structure of the Lebesgue function. Specifically, because the Lebesgue function presents several similar local maxima, specific-purpose methods successfully favor smooth gradations of the point resolution.

Nevertheless, both families of methods share some aspects. They feature the same stopping criterion, add sample points, and have computational costs scaling with the number of points. To stop the maximum approximation process, all the previous methods terminate after a fixed number of iterations. The two families of methods generate sample points statically or dynamically, statically by adding in one shot a grid of points [5], dynamically by adding at each iteration new points [2, 3, 4]. Because the Lebesgue function is evaluated at these sample points, the computational cost of the maximum estimation scales with the number of points. This scaling depends on the method, the polynomial degree, and the simplex dimension. Next, we see how to automatically stop the optimization iterations and the need for neighbor queries and scalable point refinement.

Unfortunately, to automatically stop the optimization iterations, neither the general nor the specific-purpose methods exploit that the Lebesgue function is Lipschitz. Although all the previous methods can improve the estimation of the maximum by increasing the number of iterations, none of them automatically stops when the optimal of the Lebesgue function is sufficiently converged. To measure the convergence and stop the iterations, first- and second-order optimization methods check if the approximated candidate is sufficiently flat, a successful condition that can be emulated if the function is Lipschitz — precisely the case for the Lebesgue function.

To emulate the sufficiently flat condition on a sample point, it is key to query for point neighbors. Thus, using all these points to evaluate the function, a local estimation of the Lipschitz constant is the quotient of the function difference and the distance of the neighbor points. Then, using this constant and a flatness tolerance, the automatic stopping criterion can be incorporated by only evaluating the function. The specific-purpose [2, 3] and the general [4] methods incorporate neither the stopping criterion nor the neighbor queries.

To efficiently estimate the Lebesgue constant in the simplex, it is critical to use scalable point refinement techniques. In this manner, the Lebesgue function can be finely sampled only on the interest regions and coarsely sampled otherwise, an adaptive strategy that reduces the number of needed points [2, 3, 4] . As we said before, the computational cost scales with the number of points, so the local refinement reduces the cost of approximating the Lebesgue constant. Unfortunately, when specific-purpose methods refine the resolution [2, 3, 5], the number of points scales exponentially with the dimensionality. This exponential scaling is affordable for two and three dimensions, but impractical for higher dimensions. Fortunately, some general methods do not scale exponentially with the dimension. Specifically, the DiSimpl [4] method adds only two new points per point refinement, a refinement scaling that is well-suited for higher dimensions.

Summarizing, for more than three dimensions, there is no specific-purpose method to estimate the Lebesgue constant in the simplex. For two and three dimensions, the specific-purpose meth-

ods successfully estimate the Lebesgue constant, but their extensions to arbitrary dimensions do not scale well with the dimensionality. For arbitrary dimensions, optimization methods for general functions scale well with the dimensionality, but they are not specifically devised to estimate the Lebesgue constant. That is, they do not control size gradation. Neither general nor specific-purpose methods feature neighbor queries. Thus, they are not ready to stop automatically when the optimal candidate features sufficient flatness.

To address the previous issues, the main contribution of this work is to propose a new specific-purpose point refinement method. The proposed method features a smooth gradation of the resolution, neighbor queries based on neighbor-aware point coordinates, and a point refinement that scales algebraically with the dimension as $(d+1)d$. The main novelty of the proposed smooth point refinement method is not only that it scales algebraically with the dimension but also that it is ready to use an automatic Lipschitz stopping criterion.

The main application of the proposed point refinement method is to estimate the Lebesgue constant on the simplex. Accordingly, the results check whether the proposed point refinement method reproduces the literature estimations for the triangle and the tetrahedron. Moreover, the results assess whether the method is well-suited for Lebesgue constant approximations on the simplex for mid-range dimensionality.

The rest of the paper is organized as follows. First, in Sect. 2, we review the literature related to this work. Then, in Sect. 3, we describe the system of coordinates that allow the neighbor queries and the core point refinement operations. Next, in Sect. 4, we detail the adaptive minimization method. In Sect. 5, we illustrate with several examples the main features of the presented method. Lastly, in Sect. 6, we present some concluding remarks of this work.

2. RELATED WORK

One of the most immediate estimates of the Lebesgue constant is given by the maximum value of the function in an equispaced grid of points. The size of the sampling determines the accuracy of the estimation at the expense of increasing the number of function evaluations. Alternatively, it is possible to use a sequence of admissible meshes [6] as sampling points [5]. Admissible meshes have the property that the maximum value at this finite set of points bounds the infinity norm of a polynomial of certain degree overall the simplex. Unfortunately, none of these methods is adaptive.

An alternative is to estimate the Lebesgue constant by means of a non-deterministic adaptive method [3]. The method starts with a random sample of points in the simplex. Next, the function is evaluated, and the points are sorted in terms of their function value. Then, new random samples are generated inside boxes centered at the points with the largest values. At each iteration, the box edge-length is halved to capture the maximum more accurately, and thus, a smooth gradation in the sampling resolution is obtained. This process is repeated until a prescribed number of iterations is reached. Finally, the estimate for the Lebesgue constant is the largest value at a sample point.

To compute the Lebesgue constant, an alternative adaptive method [2] named DiTri modifies the DiRect algorithm [7] to work in triangles. The method starts with the evaluation of the function at the centroid of the triangle. Next, the triangle is subdivided using a quadtree strategy. Then, the function is evaluated at the centroid of the three new smaller triangles. At each iteration, the algorithm chooses a set of potentially optimal triangles to refine in terms of their size and the function value at their centroid. After the refinement step, additional elements are refined to ensure a smooth gradation of the element size. When a prescribed number of iterations is reached, the centroid of the triangle with the largest function value determines an estimation of the Lebesgue constant. Remarkably, the method exploits the simplicity of the triangle by uniquely identifying each element with a triplet of integers, and therefore, no explicit mesh connectivity structure is needed.

Similarly to the DiRect algorithm but based on simplices, the method named DiSimpl also considers a Lipschitzian optimization approach [4]. DiSimpl is devised to find the global minimum

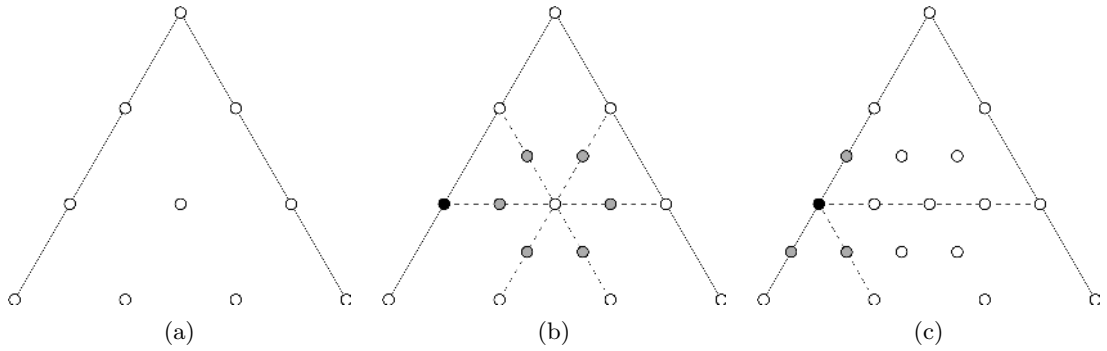


Figure 1: Illustration of the method. (a) Initial sampling. (b) Refining the point at the barycenter by generating six new points (in gray) around it parallel to the triangle edges. (c) To refine the black point we only evaluate the target function at three new points (in gray).

of an arbitrary function whose domain is a hypercube or simplex, and performs particularly well when the function presents symmetries. Initially, the search space is decomposed into simplices. Then, two approaches are considered. In one case, the function is evaluated at the centroid of the simplex, and two hyper-planes cutting the longest edge subdivide the potentially optimal simplex into three smaller simplices. In the other case, the function is evaluated at the vertices of the simplex, and one hyper-plane cutting the longest edge generates two smaller simplices. Interestingly, in any of the approaches, only two function evaluations per refinement are performed. Finally, the algorithm stops when a prescribed number of iterations is reached.

These adaptive methods outperform grid-based methods but they are not devised to estimate the Lebesgue constant in the d-dimensional simplex. The non-deterministic method [3] starts sampling the function in 10 000 points and generates 10 samples per each 2D box. Thus, to keep the same resolution we would need $10^{d/2}$ points inside the d-box. Even in 2D, a considerable amount of approximately 200 000 sample points are required to accurately estimate the Lebesgue constant, and we expect a higher value for higher dimensions. Moreover, the non-deterministic nature of the algorithm makes it difficult to query neighbor points. The quad-tree subdivision-based method [2] is devised to estimate the Lebesgue constant in the triangle. The natural extension of this method to higher dimensions would subdivide the d-simplex into 2^d subelements and, consequently, the number of sample points would increase exponentially. Furthermore, the simplicity of the triangle case to uniquely identify each element could not be exploited. Finally, the DiSimpl algorithm [4] has not been tested against the Lebesgue function and the subdivision strategy becomes complicated in high dimensions. Moreover, the resulting mesh is not conformal and the method does not feature access to neighbor elements.

The method proposed in this work is deterministic and exploits a rational barycentric system of coordinates to uniquely identify each sample point. The method considers a discrete set of directions to refine which are parallel to the simplex edges. Every time a point is refined, we evaluate the function at most $(d+1)\,d$ times. Then, after refining the potentially optimal points, we generate new sample points to ensure a smooth gradation of the sampling resolution. Moreover, the system of coordinates allows accessing to the adjacent points with no need for storing the neighbor structure, which enables a stopping criterion based on the sampling density and the local Lipschitz constant around the extremum.

3. NEIGHBOR-AWARE COORDINATES FOR POINT REFINEMENT

Even though the main application is computing the maximum of the Lebesgue function, we present the method in a minimization framework. In Sect. 3.1, we schematically illustrate the method in 2D. Then, we detail the system of coordinates in Sect. 3.2, and the core refinement operations in Sect. 3.3 and Sect. 3.4.

3.1 Outline

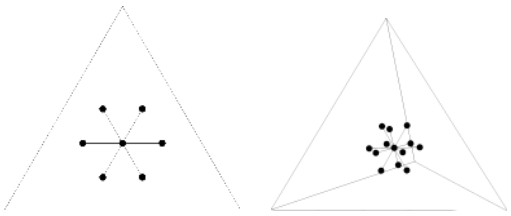
To estimate the minimum of the target function, we propose using neighbor-aware sample points. Consider the set of sample points shown in Fig. 1(a) and assume that the point at the barycenter of the triangle is our minimum candidate. To improve the estimation of the minimum, we refine the sampling around it by generating new sample points parallel to the simplicial edges, see Fig. 1(b). Analogously, if our next minimum candidate is the black point in Fig. 1(b), we generate new sample points at positions parallel to the triangle edges, see Fig. 1(c). However, we only generate three new points (in gray) since one of them already exists. Applying successively this refinement operation to potentially optimal points, we expect to finely sample the target function and find an accurate estimate of the minimum.

3.2 Neighbor-Aware Coordinates

Since we work with simplicial domains, we exploit the barycentric coordinates system. More precisely, we consider an equispaced sampling with $q+1$ points on each edge of the simplex, and we uniquely determine a sample point $\boldsymbol{x} \in K^d$ by a set of rational barycentric coordinates of the form

$$\boldsymbol{x} = \left(\frac{\lambda^1}{2^r q}, \ldots, \frac{\lambda^{d+1}}{2^r q} \right), \tag{2}$$

with $\sum_{i=1}^{d+1} \frac{\lambda^i}{2^r q} = 1$, and non-negative integers r and λ^i, $i = 1, \ldots, d+1$. For each barycentric coordinate, the numerator indicates the position on a uniform grid and the denominator represents the level of refinement on the grid. Thus, the higher the denominator, the higher the resolution of the sampling around the point.



(a) (b)

Figure 2: For a central point (black dot), surrounding stencil points (black dots) for the refinement directions (gray segments) parallel to the simplex edges (gray edges) in (a) 2D and (b) 3D.

Aligned with this system of coordinates, for each simplicial edge we choose a refinement direction. Each refinement direction has two possible orientations: forward and backward. Thus, we consider $n_D = 2n_E$ vectors, where n_E is the number of edges of K^d. Each vector is identified by a pair of integers (i, j) and is written in rational barycentric coordinates as

$$\boldsymbol{u}_{(i,j)} = \frac{1}{q} \left(\boldsymbol{e}_i - \boldsymbol{e}_j \right),$$

with $i, j = 1, \ldots, d+1$, $i \neq j$, and where the $(d+1)$-dimensional vector $\boldsymbol{e}_k$ is a vector with a one in the kth position and zeros elsewhere. The set of direction vectors

$$\mathcal{U} = \{ \boldsymbol{u}_{(i,j)} \text{ for } i, j = 1, \ldots, d+1, i \neq j \}$$

defines a canonical stencil that is used for generating new sample points. More concretely, we generate at most $(d+1)\,d$ new sample points per point refinement. These $(d+1)\,d$ refinement positions provide a reasonable scaling for medium dimensionality while sampling sufficiently fine the neighborhood of the point to refine. In particular, for the equilateral triangle, the six refinement positions are top-left, top-right, left, right, bottom-left, bottom-right, see Fig. 2(a), while for the tetrahedron there are twelve positions, see Fig. 2(b).

The value r in Eq. (2) is strongly related to the resolution of the sampling. Without loss of generality, consider the sample point $\boldsymbol{x} = \left(\frac{1}{2}, \frac{1}{2}\right)$ in the

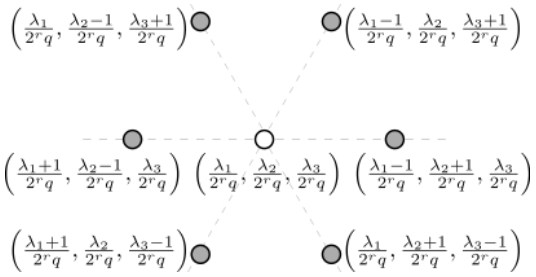


Figure 3: Coordinates of a point of resolution r and its neighbors.

one-dimensional interval $[0, 1]$, and the direction vector $\boldsymbol{u}_{(1,2)} = \left(\frac{1}{2}, \frac{-1}{2}\right)$. The point

$$\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{u}_{(1,2)} = \left(\frac{2}{2}, \frac{0}{2}\right)$$

corresponds to the point zero in cartesian coordinates, while the point $\boldsymbol{z} = \boldsymbol{x} + \frac{1}{2}\boldsymbol{u}_{(1,2)} = \left(\frac{3}{4}, \frac{1}{4}\right)$ is between the points $\boldsymbol{x}$ and $\boldsymbol{y}$. Thus, scaling the vector $\boldsymbol{u}_{(1,2)}$ with a factor of the form 2^{-r}, $r \geq 0$, leads to the generation of closer points along the direction described by the vector $\boldsymbol{u}_{(1,2)}$.

The value of q determines the density of the initial sampling, see Eq. (2). In practice, we favor setting q equal to one. When q is one, the initial grid contains only the simplex vertices as sampling points. That is, the initial number of sampling points is $d + 1$, and thus, it scales linearly with the number of dimensions. Bigger values determine denser initial samplings that might require less adaptation iterations, but the initial number of sampling points scales exponentially with the number of dimensions. This initial offset might be reflected in a larger number of final sampling points required to seek the target function minimum.

We store the points in a hash table built from the rational coordinates. Hence, the point $\left(\frac{\lambda^1}{2^r q}, \ldots, \frac{\lambda^{d+1}}{2^r q}\right)$ and the point $\left(\frac{2^k \lambda^1}{2^{r+k} q}, \ldots, \frac{2^k \lambda^{d+1}}{2^{r+k} q}\right)$ are identified as the same point. Moreover, this system of rational barycentric coordinates allows to easily access to the neighbor points with no need for storing the neighbor structure. As depicted in Fig. 3, coordinates of neighbor points with the same denominator only differ in one unit. Thus, to query if a neighbor exists, we simply add one to one component, subtract one from another component, and search in the set of points. Therefore, there is no need for storing explicit connectivity information.

3.3 Point Refinement

Besides the resolution, it is also useful to classify the points in terms of the completeness of the stencil. On the one hand, we say a point $\boldsymbol{x}$ is *incomplete of resolution* r if the sample point $\boldsymbol{x} + 2^{-r}\boldsymbol{u}_{(i,j)}$ exists, for some $\boldsymbol{u}_{(i,j)} \in \mathcal{U}$. Alternatively, this means that at least one point of the stencil of resolution r centered at $\boldsymbol{x}$ exists. A sample point $\boldsymbol{x}$ may be incomplete in several resolutions, but it is for the highest resolution when the representation of the function around $\boldsymbol{x}$ is more accurate.

On the other hand, if all the points of the stencil exist, we say this point is complete. More precisely, a point $\boldsymbol{x}$ is *complete of resolution* r if the sample point $\boldsymbol{x} + 2^{-r}\boldsymbol{u}_{(i,j)}$ exists, for all $\boldsymbol{u}_{(i,j)} \in \mathcal{U}$. Similarly to the incomplete case, the higher the resolution, the more accurate the representation of the function around the point.

We remark that a complete point of resolution r provides a finer discretization than an incomplete point of resolution r' with $r' \leq r$, since the neighborhood is denser and sampled along all the considered directions. Furthermore, a point may be complete and incomplete at the same time both providing meaningful information. For instance, consider a complete point of resolution r which is also incomplete of resolution $r + 1$. In this case, not only the neighborhood is fully sampled at resolution r, but additional partial information of the function at resolution $r + 1$ is known.

Around an incomplete point, this partial information has to be enhanced to obtain a more accurate representation of the target function. Accordingly, we consider an operation that completes the stencil around an incomplete point. Specifically, to complete an incomplete point of resolution r, we propose to generate all the missing points of the stencil of resolution r. Thus, the resulting point is no longer incomplete at level r.

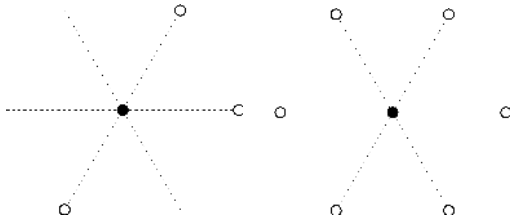


Figure 4: Completing an incomplete sample point. An (a) incomplete point of resolution r becomes (b) complete of resolution r.

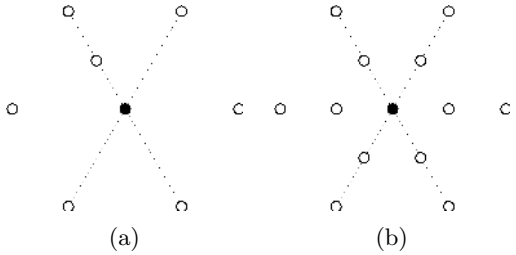


Figure 5: Refining a complete sample point. A (a) complete point of resolution r becomes (b) complete of resolution $r+1$.

In Fig. 4, for the two-dimensional case, we illustrate the completion step for an incomplete point of resolution r. Since three points of the stencil exist, Fig. 4(a), we only generate the remaining missing points to complete the stencil. Once completed, the point becomes complete of resolution r, Fig. 4(b).

When the information gathered from a complete point indicates that there is a minimum nearby, we should sample the function in a smaller neighborhood to capture it. Thus, we need a point refinement operation. Refining a complete point of resolution r consists in generating all the points of the stencil of resolution $r+1$. Thus, the point becomes complete of resolution $r+1$. We highlight that if the point is incomplete of level $r+1$, we only generate those points needed to complete the stencil of resolution $r+1$ and, therefore, we avoid repeated function evaluation. In Fig. 5, we illustrate, for the two-dimensional case, the refinement of a complete point of resolution r which is also incomplete of resolution $r+1$. Since one point of the stencil of resolution $r+1$ exists, see Fig. 5(a), we generate five points to complete the stencil. Then, the point becomes complete of resolution $r+1$, see Fig. 5(b).

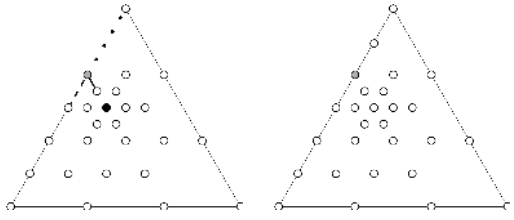


Figure 6: Smooth gradations of the resolutions. (a) The gray point is complete of resolution r (dotted line) and incomplete of resolutions $r+1$ (dashed line) and $r+2$ (solid line). (b) Smooth sampling after refining the gray point until it becomes complete of resolution $r+1$.

3.4 Smooth Gradation

To obtain smooth discretizations of the target function, we need smooth gradations of the resolution of the sampling points. Accordingly, we only consider sampling configurations where the resolution between neighbors differs at most by one unit. More precisely, assume that the finest complete resolution of a point is r, and the highest incomplete resolution is r', $r' > r$. Then, the sampling is smooth if $r' = r+1$. Thus, after completing or refining a point we check if we have a smooth gradation of points. If there is a point such that $r' > r+1$, we smooth it by refining until resolution $r'-1$.

In Fig. 6(a), we show the sampling after refining the black point. We observe that the gray point is complete of resolution r (dotted stencil), but it is also incomplete of resolutions $r+1$ (dashed stencil) and $r+2$ (solid stencil), and therefore, this sampling configuration is non-smooth. To obtain a smooth discretization, we refine the gray point until it becomes complete of resolution $r+1$ by generating one new point, see Fig. 6(b). Now, there is a smooth gradation of the point resolution.

Algorithm 1 Approximating the minimum by sampling.

Input: Function F, Domain K^d
Output: Minimum $\boldsymbol{x}^\star$, $F(\boldsymbol{x}^\star)$
1: **function** ComputeMinimum(F, K^d)
2: $\Sigma \leftarrow$ InitializeSamplePoints(F, K^d)
3: $\boldsymbol{x}^\star \leftarrow$ GetMinimum(Σ)
4: **while** $\boldsymbol{x}^\star$ is not a minimum of F **do**
5: $\Sigma_C \leftarrow$ GetCompletePoints(Σ)
6: $\{\boldsymbol{x}_{C_i}\} \leftarrow$ GetPointsToRefine(Σ_C)
7: RefinePoints(F, Σ, $\{\boldsymbol{x}_{C_i}\}$)
8: $\Sigma_I \leftarrow$ GetIncompletePoints(Σ)
9: $\{\boldsymbol{x}_{I_j}\} \leftarrow$ GetPointsToComplete(Σ_I)
10: CompletePoints(F, Σ, $\{\boldsymbol{x}_{I_j}\}$)
11: SmoothSampling(F, Σ)
12: $\boldsymbol{x}^\star \leftarrow$ GetMinimum(Σ)
13: **end while**
14: **return** $\boldsymbol{x}^\star$, $F(\boldsymbol{x}^\star)$
15: **end function**

4. ADAPTIVE POINT REFINEMENT

In this section, we first present our adaptive method to estimate the minimum of a function defined in the d-dimensional simplex, see Sect. 4.1. The rational barycentric coordinate system described in Sect. 3 is the core of our method since an explicit point connectivity structure is not needed. Then, in Sect. 4.2, we detail the stopping criterion.

4.1 Algorithm

The adaptive point refinement is detailed in Algorithm 1. Given the function to minimize, F, and the simplicial domain where it is defined, K^d, the first step of the method is to initialize the set of sample points denoted as Σ, Line 2. We remark that function F is an arbitrary target function, yet in our main application it corresponds to minus the Lebesgue function. In the second step, the method gets the first minimum approximation on the initial sample points, Line 3. This initialization allows iterating to seek a better approximation of the minimum until convergence, Line 4. To improve the minimum approximation, the iterative process successively refines and completes the potentially optimal sample points and smooths the gradation of the sampling point resolution.

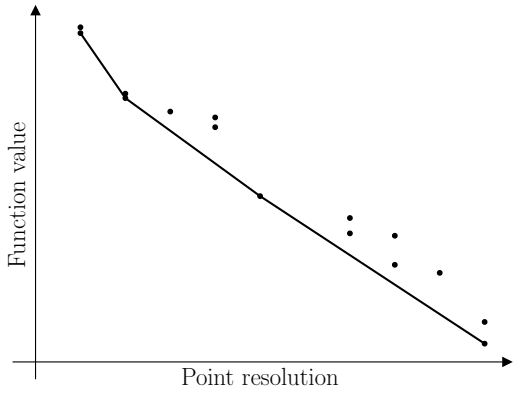


Figure 7: Two-dimensional representation of the complete points in terms of the resolution and function value. The lower boundary of the convex hull determines the points to complete.

First, the method refines the candidate points. To this end, in Line 5, we retrieve the set of complete points Σ_C and choose the points to refine, Line 6. We determine the points to refine in terms of their resolution and function value. More concretely, each complete sample point is represented in a graph by a dot with the horizontal coordinate given by its resolution, and the vertical coordinate given by its function value. If a point $\boldsymbol{x}$ is complete of resolutions $r_1, \ldots, r_k$, with $r_1 < r_2 < \cdots < r_k$, then it is represented by a dot at position $(r_k, F(\boldsymbol{x}))$. In Fig. 7, we show this graph in an intermediate stage of the algorithm. Similarly to the DiRect algorithm [7], we choose the points to refine by exploring multiple Lipschitz constants which, in practice, reduces to computing the lower boundary of the convex hull of this point cloud. Then, in Line 7, we refine the chosen points $\{\boldsymbol{x}_{C_i}\}$.

Second, the method completes the incomplete points. To this end, in Line 8, we obtain the set of incomplete points Σ_I and choose the points to complete, Line 9. Let $\boldsymbol{x} \in \Sigma_I$ be an incomplete point of resolutions $r_1, \ldots, r_k$, with $r_1 < r_2 < \cdots < r_k$, which is either not complete or complete with finest resolution r, $r < r_1$. Incomplete points provide information about the function in a local sense since they have been sam-

pled along a particular direction only. In contrast, complete points have been sampled along all the directions and, therefore, global information is known. Since we prefer to have first a big picture of the function landscape before focusing on the higher-resolution detail, we represent the incomplete point $\boldsymbol{x}$ by a dot with coordinates $(r_1, F(\boldsymbol{x}))$ instead of $(r_k, F(\boldsymbol{x}))$. Then, we obtain a point cloud similar to the one shown in Fig. 7. The lower part of the convex hull of this representation of Σ_I determines the points $\{\boldsymbol{x}_{I_j}\}$ to be completed. Finally, in Line 10, we complete these points.

Third, the method smooths the gradation of the sampling point resolution. Specifically, in Line 11, we generate the points needed to ensure the sampling is smooth, see Sect. 3.4, and retrieve the minimum point $\boldsymbol{x}^\star$ from the sampling Σ, Line 12. These steps are repeated until the point $\boldsymbol{x}^\star$ is a minimum, see Line 4. The details on the stopping criterion are to be described in Sect. 4.2. Finally, the algorithm returns the minimum point and the function value at the minimum, Line 14.

We highlight that the function is evaluated only in the generation of new sample points, that is, in the refinement, completion, and smoothing steps. Further, in the point data structure, we store the point coordinates and the function value, so it can be immediately obtained when needed avoiding repeated calculations. To easily access to the neighbor points, the point data structure also contains an updated list of the complete and incomplete resolutions.

4.2 Stopping Criterion

In zeroth-order minimization, it is standard to stop seeking a minimum when a fixed number of iterations is reached or when the minimum approximation is numerically close to a known minimum value. In our case, only the value of the function is known, yet the sample structure allows obtaining an indicator of the flatness of the function. Accordingly, we can consider a stopping criterion accounting for the function flatness as in first- and second-order optimization methods. The user specifies spatial and functional tolerances, and the method automatically stops when a minimum below these thresholds is found.

The spatial tolerance controls the resolution of the sampling in the neighborhood of the minimum sample point. Specifically, given a spatial tolerance δ, there exists a resolution R such that the distance between a point $\boldsymbol{x}$ and the point $\boldsymbol{x} + 2^{-r}\boldsymbol{u}_{(i,j)}$ is smaller than δ for all $r \geq R$ and vector $\boldsymbol{u}_{(i,j)} \in \mathcal{U}$. Note that a complete point of resolution r, $r \geq R$, satisfies this criterion.

The functional tolerance ε is used to assess the flatness of the function around a point in terms of an estimate of the local Lipschitz constant. Specifically, consider a complete point $\boldsymbol{x}$ of resolution r, and denote by $\boldsymbol{y}$ the neighbor along the direction (i,j), $\boldsymbol{y} = \boldsymbol{x} + 2^{-r}\boldsymbol{u}_{(i,j)}$. We estimate the Lipschitz constant of resolution r around $\boldsymbol{x}$ along the direction (i,j) as

$$\tilde{K}_{(i,j)}(\boldsymbol{x}) = \frac{F(\boldsymbol{x}) - F(\boldsymbol{y})}{d(\boldsymbol{x}, \boldsymbol{y})},$$

where $d(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\|_2$ is the distance between points $\boldsymbol{x}$ and $\boldsymbol{y}$. Note that we allow negative Lipschitz constant estimations. In particular, $\tilde{K}_{(i,j)}(\boldsymbol{x})$ is negative if and only if $F(\boldsymbol{x}) < F(\boldsymbol{y})$. Moreover, the magnitude of the Lipschitz constant is strongly related to the flatness of the function around $\boldsymbol{x}$. Thus, the point $\boldsymbol{x}$ is a minimum candidate if $\tilde{K}_{(i,j)}(\boldsymbol{x})$ is negative and

$$\left|\tilde{K}_{(i,j)}(\boldsymbol{x})\right| < \varepsilon,$$

for all the directions $\boldsymbol{u}_{(i,j)} \in \mathcal{U}$.

In Algorithm 1, at the end of loop, there exists a sample point $\boldsymbol{x}^\star$ such that $F(\boldsymbol{x}^\star) \leq F(\boldsymbol{y})$, for all $\boldsymbol{y} \in \Sigma$. Then, in Line 4, we check if the point $\boldsymbol{x}^\star$ is complete of resolution r, $r \geq R$, and the local estimates of the Lipschitz constant along all the possible directions for resolution r are less than ε. If so, we assume that the neighborhood of $\boldsymbol{x}^\star$ has been sufficiently sampled and the function is sufficiently flat there. Thus, the point $\boldsymbol{x}^\star$ is considered an estimate of the function minimum and the algorithm stops.

Alternatively, it is also possible to limit the number of iterations. This limit allows the user to obtain an approximation of the minimum before the tolerance-based stopping criterion is satisfied. In both cases, the algorithm returns the sample point $\boldsymbol{x}^\star$ with the smallest function value.

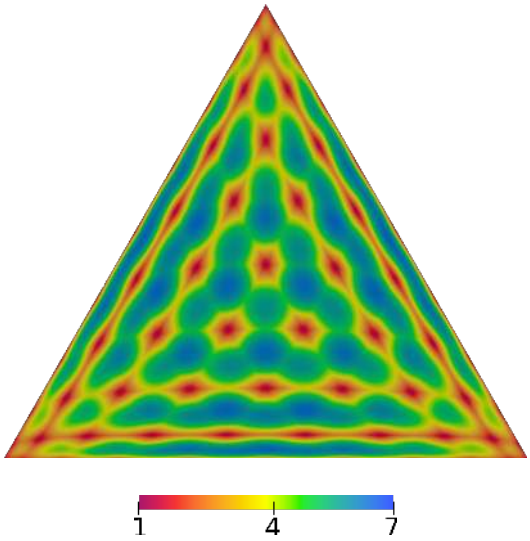


Figure 8: Lebesgue function of the warp-and-blend nodal distribution of polynomial degree 10 in the triangle [3].

5. RESULTS: ESTIMATION OF THE LEBESGUE CONSTANT

The main application of the method presented in Sect. 4 is the estimation of the Lebesgue constant in the d-dimensional simplex. The Lebesgue constant is used to assess the interpolation capabilities of a nodal distribution and is defined as the maximum of the Lebesgue function, see Eq. (1). Due to the absolute value, the function is non-differentiable and, hence, a zeroth-order method is required to compute the maximum.

In Fig. 8, for a triangle of polynomial degree 10, we show the Lebesgue function for a warp-and-blend nodal distribution [3]. Since this nodal family is symmetric, the Lebesgue function is symmetric, too. Consequently, it is enough to find the maximum inside the sextant of the triangle. More precisely, we consider the symmetric tile of the d-dimensional simplex determined by the points with barycentric coordinates $\left(\lambda^1, \ldots, \lambda^{d+1}\right)$, $\sum_{i=1}^{d+1} \lambda^i = 1$, such that $\lambda^i \geq \lambda^j$ if $i \geq j$.

	Equispaced		Warp-and-blend	
p	Λ_{Eq}	# points	Λ_{WB}	# points
2	1.67	219	1.67	221
3	2.27	302	2.11	292
4	3.47	280	2.66	283
5	5.45	280	3.12	483
6	8.75	424	3.70	404
7	14.34	356	4.27	378
8	24.01	409	4.96	668
9	40.92	533	5.74	611
10	70.89	397	6.67	685
11	124.53	427	7.90	497
12	221.41	538	9.36	747
13	397.70	422	11.47	735
14	720.69	412	13.97	1142
15	1315.89	599	17.65	885

Table 1: Number of sample points needed to estimate the Lebesgue constant using the equispaced distribution, Λ_{Eq}, and the warp-and-blend distribution [3], Λ_{WB}, of polynomial degree $p = 2, \ldots, 15$ as interpolation set in the triangle.

5.1 Verification in 2D and 3D

To verify the estimated values of the Lebesgue constant found using our method, we compare our results with those reported in [3]. In Table 1, we report the value of the Lebesgue constant for the equispaced and the warp-and-blend distribution [3] for several polynomial degrees p, $p = 2, \ldots, 15$, in the triangle. The initial sampling is composed of the three vertices of the domain, $q = 1$. We set $\delta = 10^{-4}$ and $\varepsilon = 10^{-3}$ for the stopping criterion, and in all cases the minimum is found before the limit of 50 iterations is reached. We also list the number of sample points needed. In general, as the polynomial degree increases, also does the number of points. This is so because, for high polynomial degrees, the basins of the Lebesgue function that contain the minima are smaller and deeper and, consequently, more sample points are needed to capture the minimum with the same precision. In spite of this fact, we remark that our method is able to compute a good estimate of the Lebesgue constant using less than 1200 sample points, yet the values coincide with those reported in [3] up to the second decimal place.

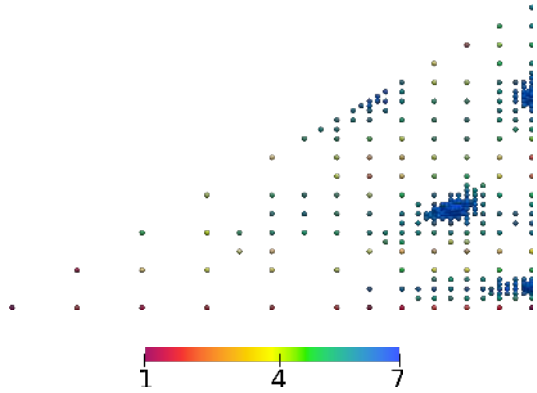


Figure 9: Final sampling used to capture the maximum of the Lebesgue function of the warp-and-blend nodal distribution of polynomial degree 10 in the triangle [3].

In Fig. 9, we show the final sampling used to capture the maximum of the Lebesgue function associated with the warp-and-blend distribution of polynomial degree 10 represented in Fig. 8. Since this function features triangle symmetry, the search space is simply the sextant. We remark that regions with higher values, blueish areas in Fig. 8, present a finer sampling in Fig. 9. We also see that there are three local minima with similar function values, yet the global minimum is the one in the interior of the domain.

In Table 2, we report the maximum value and the number of sample points needed to estimate the Lebesgue constant for the equispaced and the warp-and-blend distribution [3] for several polynomial degrees p, $p = 2, \ldots, 15$, in the tetrahedron. We use the same initial sampling and the same tolerances $\delta = 10^{-4}$ and $\varepsilon = 10^{-3}$ for the stopping criterion, and in all cases the minimum is found before the limit of 50 iterations is reached. As in the two-dimensional case, more sample points are required to estimate the Lebesgue constant of higher polynomial degrees. We highlight that the values coincide with those reported in [3] up to the second decimal place, and only 3519 points are needed to compute an estimate of the Lebesgue constant for the warp-and-blend distribution of polynomial degree 15. In contrast, using an admissible mesh of $3519^{(1/3)} \approx 16$ points per line, the estimated value of the Lebesgue constant is 211.07.

	Equispaced		Warp-and-blend	
p	Λ_{Eq}	# points	Λ_{WB}	# points
2	2.00	398	2.00	398
3	3.02	565	2.93	635
4	4.88	536	4.07	722
5	8.09	581	5.32	990
6	13.66	690	7.01	1040
7	23.38	675	9.21	1671
8	40.55	751	12.54	854
9	71.15	708	17.02	1651
10	126.20	779	24.36	2412
11	225.99	798	36.35	1644
12	408.15	853	54.18	1707
13	742.69	860	84.62	2594
14	1360.49	843	135.75	2635
15	2506.95	926	217.71	3519

Table 2: Number of sample points needed to estimate the Lebesgue constant using the equispaced distribution, Λ_{Eq}, and the warp-and-blend distribution [3], Λ_{WB}, of polynomial degree $p = 2, \ldots, 15$ as interpolation set in the tetrahedron.

5.2 Performance Comparison in 2D

To check the performance, we compare the results of our method with the results of our implementation of the DiTri algorithm [2]. For both methods, we compute in a triangle the Lebesgue constant for the warp-and-blend symmetric nodal distribution of polynomial degree 10. To do so, we report, at the end of each iteration, the number of sample points and the relative error of the maximum estimation. In Fig. 10, we show the evolution of our method, in blue, and the DiTri algorithm, in red. We observe that both methods show similar evolution. Moreover, to capture the maximum with a relative error below 10^{-4}, both methods need less than 200 sample points. Note that we do not consider the non-deterministic method [3] because the initial sampling already consists of 10 000 sample points.

Although the evolution of both methods is similar in 2D, our method scales better in higher dimensions. We highlight that to refine a point using our method, we generate at most 6 new sample

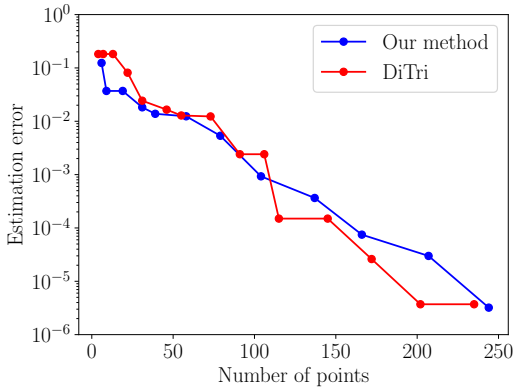


Figure 10: Error in the estimation of the Lebesgue constant in terms of the number of sample points using our method (blue) and DiTri [2] (red) for the warp-and-blend distribution of polynomial degree 10 in the triangle.

points, while DiTri always requires 3 new sample points. This difference is almost irrelevant in 2D and, consequently, the two curves follow a similar trend. However, this would not be the case in higher dimensions since we generate at most $(d+1)\,d$ new sample points per refinement, while an extension to higher dimensions of DiTri would require $2^d - 1$ new sample points. Hence, for each method, the number of sample points scales differently with the dimension d, exponentially for an extension to arbitrary dimensions of DiTri, quadratically for our method.

Moreover, since we only need a point data structure, the refinement, completion, and smoothing operations are implemented for arbitrary dimensions and no dimension-specific considerations are required. Finally, the system of rational barycentric coordinates allows easily accessing the adjacent points with no need for storing the neighbor structure, which enables using a stopping criterion based on the flatness of the function.

5.3 Results in 4D, 5D, and 6D

The values reported in Sect. 5.1 for 2D and 3D coincide with the ones found in the literature [3]. Thus, we believe that our method is capable of estimating the Lebesgue constant accurately using a moderate amount of sample points. In Table 3, we show our estimation Λ_{Eq} of the Lebesgue constant of the equispaced nodal distribution of polynomial degree p in the d-simplex, $p = 6, \ldots, 10$, $d = 4, \ldots, 6$. We also show the number of required sample points. As expected, we observe that the values increase with the polynomial degree and the dimension.

As an alternative to our method, we can use an admissible mesh [6] to estimate the Lebesgue constant. Since in dimension $d = 4$ we provided an estimate using at most 1572 sample points, we approximate the Lebesgue constant using an admissible mesh of approximately $1572^{1/4}$ points per line. Analogously, in 5D and 6D, we compute an estimate using approximately $2030^{1/5}$ and $2466^{1/6}$ points per line, respectively. In Table 3, we denote by $\tilde{\Lambda}_{\text{Eq}}$ the maximum function value at this grid of sample points. We observe that with the same number of points, our method captures a higher value and, therefore, it is more suitable to estimate the Lebesgue constant in small and moderate dimensionality.

6. CONCLUDING REMARKS

To estimate the Lebesgue constant on the simplex, we have proposed a new specific-purpose point refinement method. The proposed method features a smooth gradation of the resolution, neighbor queries based on neighbor-aware coordinates, and a point refinement that algebraically scales with dimensionality. Remarkably, by using neighbor-aware coordinates, the point refinement method is ready to automatically stop using a Lipschitz criterion.

In mid-range dimensionality, we conclude that the point refinement is well-suited to automatically and efficiently estimate the Lebesgue constant on simplices. Specifically, for different polynomial degrees and point distributions, our results efficiently have reproduced the literature estimations for the triangle and the tetrahedron. Moreover, we have adaptively estimated the Lebesgue constant up to six dimensions.

In perspective, for a given polynomial degree, the proposed point refinement might be relevant to obtaining a set of simplex points that guarantees

Degree	Dimension 4			Dimension 5			Dimension 6		
p	# points	Λ_{Eq}	$\tilde{\Lambda}_{\text{Eq}}$	# points	Λ_{Eq}	$\tilde{\Lambda}_{\text{Eq}}$	# points	Λ_{Eq}	$\tilde{\Lambda}_{\text{Eq}}$
6	1126	19.22	19.05	2030	25.49	19.90	1807	32.63	25.45
7	1075	34.08	33.51	1545	46.54	34.43	1790	61.00	50.46
8	1033	60.86	55.75	1578	85.24	65.06	2466	114.13	96.97
9	1175	109.43	90.72	1677	156.62	126.31	2252	213.76	180.24
10	1572	198.08	150.71	1667	288.82	241.51	2381	400.93	323.42

Table 3: Estimation of the Lebesgue constant of the equispaced distribution of polynomial degree $p = 6, \ldots, 10$ in the d-simplex, $d = 4, \ldots, 6$. Number of sample points needed to compute our estimation Λ_{Eq}, and approximation using an admissible mesh $\tilde{\Lambda}_{\text{Eq}}$.

a small interpolation error. That is, it efficiently estimates the Lebesgue constant, an estimation that is helpful in two ways. First, to assess the quality of a given set of interpolation points. Second, to evaluate the Lebesgue constant when optimizing the interpolation error for the point distribution as a design variable. We also think the method might be well-suited to seek optima in the simplex for functions behaving as the Lebesgue function.

ACKNOWLEDGMENTS

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 715546. This work has also received funding from the Generalitat de Catalunya under grant number 2017 SGR 1731. The work of the second author has been partially supported by Grant IJC2020-045140-I from MCIN/AEI/10.13039/501100011 033 and "European Union NextGenerationEU/PRTR". The work of the third author has been partially supported by the Spanish Ministerio de Economía y Competitividad under the personal grant agreement RYC-2015-01633.

References

[1] Angelos J.R., Kaufman Jr E.H., Henry M.S., Lenker T.D. "Optimal nodes for polynomial interpolation." *Approximation theory VI*, vol. 1, 17–20, 1989

[2] Roth M.J. *Nodal configurations and Voronoi tessellations for triangular spectral elements*. Ph.D. thesis, 2005

[3] Warburton T. "An explicit construction of interpolation nodes on the simplex." *Journal of engineering mathematics*, vol. 56, no. 3, 247–262, 2006

[4] Paulavičius R., Žilinskas J. "Simplicial Lipschitz optimization without Lipschitz constant." *Simplicial Global Optimization*, pp. 61–86. Springer, 2014

[5] Briani M., Sommariva A., Vianello M. "Computing Fekete and Lebesgue points: simplex, square, disk." *Journal of Computational and Applied Mathematics*, vol. 236, no. 9, 2477–2486, 2012

[6] Calvi J.P., Levenberg N. "Uniform approximation by discrete least squares polynomials." *Journal of Approximation Theory*, vol. 152, no. 1, 82–100, 2008

[7] Jones D.R., Perttunen C.D., Stuckman B.E. "Lipschitzian optimization without the Lipschitz constant." *Journal of optimization Theory and Applications*, vol. 79, no. 1, 157–181, 1993