Finding the Minimum of a Function

Albert Cohen, Ronald DeVore, Guergana Petrova, and Przemysław Wojtaszczyk

Dedicated to Stanley Osher on the occasion of his 70-th birthday with much admiration. *

February 14, 2013

Abstract

Adaptive query algorithms for finding the minimum of a function f are studied. The algorithms build on the earlier adaptive algorithms given in [5, 7]. The rate of convergence of these algorithms is estimated under various model assumptions on the function f. The first class of algorithms is analyzed when f satisfies a smoothness condition, e.g. $f \in C^r$, and an assumption on its level sets as given in [7]. There is a distinction drawn as to whether or not the algorithm has knowledge of the semi-norm $|f|_{C^r}$. If this information is known, it is rather straightforward to design algorithms with optimal performance and to show that this performance is better than non-adaptive algorithms. A bit more subtle is to build algorithms which are universal in that they do not need to know the semi-norm of f. Universal algorithms are built that have the same asymptotic performance as when the semi-norm is known, save for a logarithm. The second part of this paper studies adaptive algorithms for finding the minimum of a function in high dimension. In this case, additional assumptions are placed on f, of the form given in [3], that have the effect of variable reduction and thereby avoiding the curse of dimensionality. These algorithms are again shown to be asymptotically optimal up to a logarithm factor.

Keywords: computing minima, adaptive methods, tree based algorithms, high dimension
MSC numbers: Primary: 65K99, 68Q25; Secondary 41A25, 90C60

1 Introduction

We discuss strategies for finding the minimum value $m(f) = m(f, \Omega)$ taken by a continuous function f defined on $\Omega := [0,1]^d$. We assume that we are able to query f by asking for its value at any point. The algorithms we study give an adaptive procedure to determine the points to query f. After a typical step of the algorithm, it will have asked for the values $f(\xi_1), \dots, f(\xi_n)$ and then uses these values to estimate $m(f)$. In the simplest case, one could take

$$\hat{m}_n(f) = \min_{1 \le j \le n} f(\xi_j), \tag{1.1}$$

*This research was supported by the Office of Naval Research Contracts ONR N00014-09-1-0107, ONR N00014-11-1-0712, ONR N00014-12-1-0561; by the AFOSR Contract FA95500910500; by the NSF Grants DMS 0915231, DMS 1222715; by the "HPC Infrastructure for Grand Challenges of Science and Engineering" Project, co-financed by the European Regional Development Fund under the Innovative Economy Operational Program; by the Polish NCN grant DEC2011/03/B/ST1/04902, and the ANR project Défi08 ECHANGE. This publication is based on work supported by Award No. KUS-C1-016-04, made by King Abdullah University of Science and Technology (KAUST).

as the approximation to $m(f)$. This choice works when f is in Lip 1. However, when f has higher smoothness, better choices for the estimate $\hat{m}_n(f)$ are given in (2.21) below. We denote by

$$E_n(f) := |\hat{m}_n(f) - m(f)| \tag{1.2}$$

the error produced by such an algorithm. Of course $E_n(f)$ depends on the particular algorithm used and properties of f. In particular, we are interested in when such adaptive methods can perform better than simply querying f at n equally spaced points.

In order to give a bound for the decay of $E_n(f)$ as $n \to \infty$, one needs to assume something about f. A common model for the function f is to assume that it is in $\mathrm{Lip}_{M^*}1$. Then, $n = N^d$ suitably spaced point queries (independent of f) would give an error

$$E_n(f) \leq \sqrt{d}2^{-1}M^*n^{-1/d}. \tag{1.3}$$

Starting with the algorithm in [7] based on graphs, adaptive algorithms have been constructed which perform better than non-adaptive queries provided the function satisfies another condition described by level sets. Namely, the following level set condition was introduced in [7]:

Level Set Condition of Order s: *For any $s > 0$, we say $f \in \mathcal{L}(s, L)$ if*

$$|\{x : \ f(x) \leq \delta + m(f)\}| \leq L\delta^s, \quad \delta > 0, \tag{1.4}$$

where here and later, we denote by $|A|$ the Lebesgue measure of a set $A \subset \mathbb{R}^d$. Notice, that since $|\Omega| = 1$, the condition (1.4) automatically holds for δ large by adjusting L. The previous work in [7, 5] assumes that (1.4) only holds for $\delta \leq \delta_0$ which may enable the introduction of a smaller L, but then all results also involve the constant δ_0. Our results could also be presented in that way, but at the expense of much more complicated statements which we choose not to make.

Let us also say a few important words about the compatibility between smoothness assumptions like $f \in \mathrm{Lip}_{M^*}1$ and level set assumptions like $f \in \mathcal{L}(s, L)$.

Remark 1.1 *If $f \in \mathrm{Lip}_{M^*}1$ then f cannot be in $\mathcal{L}(s, L)$ for any $s > d$. Indeed, if f assumes its minimum at a point $\xi \in \Omega$, then for any $x \in \Omega$ in the ball B_δ of radius δ centered at ξ, we have*

$$|f(x) - f(\xi)| \leq M^*\delta.$$

It follows that

$$c_d\delta^d \leq |B_\delta \cap \Omega| \leq |\{x : f(x) \leq m(f) + M^*\delta\}|, \quad \delta > 0.$$

If, in addition, f has higher smoothness, say C^2, near a minimum ξ which is in the interior of Ω, then $\nabla(f)(\xi) = 0$ and $f(x) - f(\xi) \leq C\delta^2$ on B_δ. The same argument now gives that s cannot exceed $d/2$.

In [7], an algorithm based on graphs was given that shows that whenever $f \in \mathrm{Lip}_{M^*}1$ and f satisfies the level set condition of order s, then

$$E_n(f) = O(n^{-\frac{1}{d-s}}), \quad n \to \infty. \tag{1.5}$$

The above performance results require the knowledge of M^*. The main point of [5] was to study whether estimates of this type can be obtained without this knowledge. We call such algorithms

universal. A universal algorithm was given in [5] for functions f in $\mathrm{Lip}_{M^*}1$, which does not require knowledge of M^*, but still yields an estimate [1]

$$E_n(f) \le C_0 L^{2/d} [M^*]^8 n^{-2/d}, \quad n \ge 1, \tag{1.6}$$

whenever f is in $\mathcal{L}(d/2, L)$. Note that this estimate is not explicity stated in [5] but can be derived from his error and complexity estimates.

In the present paper, we further study adaptive algorithms for obtaining bounds for $m(f)$ with the goal of treating more general smoothness conditions, more general level set parameters s, and also obtaining provably optimal results. Our main technique, which is based on adaptive dyadic partitioning and tree algorithms is conceptually similar to [5], but its implementation and analysis is much simpler.

Our main result, given in §3, constructs a universal algorithm for C^r functions and general level set parameter s. As a special case, we obtain that the factor $[M^*]^8$ appearing in (1.6) can be replaced by $[M^* \log M^*]^2$. Namely, we give a universal algorithm such that whenever the function f is in $\mathrm{Lip}_{M^*}1$ and also in $\mathcal{L}(s, L)$, then we have

$$E_n(f) \le C_0(d, s)[M^* \log M^*]^{\frac{d}{d-s}} \left(\frac{L}{n}\right)^{\frac{1}{d-s}}, \quad n \ge 1. \tag{1.7}$$

We also show in §5 that save for the appearance of the logarithm, this algorithm gives optimal performance not only in the dependence on n but also on M^*. Namely, we prove that for any algorithm (adaptive or not) using n point queriess, we have the lower bound for performance

$$\sup_{f \in \mathcal{L}(s,L) \cap \mathrm{Lip}_{M^*}1} E_n(f) \ge c_0 [M^*]^{\frac{d}{d-s}} \left(\frac{L}{n}\right)^{\frac{1}{d-s}}, \tag{1.8}$$

for an absolute constant c_0 and n sufficiently large. In the case $s = d/2$, the same lower bound was proved in [5] but for a restricted class of algorithms. We do not know whether in (1.7) it is possible to remove this logarithm by another method.

We begin in §2 by deriving algorithms and results which apply to more general smoothness conditions $f \in C^r$ and general level set parameters s that appear in (1.4). We first establish algorithms that require knowledge of r and a bound for the semi-norm $|f|_{C^r}$ but apply to any level set parameter s without knowing the value of s. We call these *core* algorithms since they are at the core of constructing universal algorithms.

We then turn our attention to universal algorithms. We give in §3 a universal algorithm for C^r functions which does not require knowledge of $|f|_{C^r}$ and then prove in §5 that its performance is optimal, again save for a logarithm of $|f|_{C^r}$.

The above algorithms become computationally infeasible when the dimension d is large. One can see this already in the factor $n^{-2/d}$ that appears in (1.6). This is the so-called *curse of dimensionality*. In order to break this curse, in the context of approximating f, it is known that one needs additional structure on f. There are several model classes that have been introduced in high dimensions which both match some application domains and also avoid the curse of dimensionality (see [3, 1, 4, 6]). We build on the model classes introduced in [3] which assume that f either depends only on $\ell << d$ variables or can be approximated by functions of this type. As an example

[1] We denote by c_0, C_0 absolute constants. Otherwise, we indicate the dependence of the constants on the parameters involved, e.g., $C(d)$ means a constant depending only on d. These constants may vary from line to line.

of the results we derive, we mention that we give adaptive algorithms for finding $m(f)$ for $\mathrm{Lip}_{M^*}1$ that do not require knowledge of either M^* or the ℓ active variables but still give the estimate

$$E_n(f) \le C_0[M^* \log M^*]^2 L n^{-2/\ell}, \quad n \ge 1, \tag{1.9}$$

whenever $f \in \mathcal{L}(\ell/2, L)$. Various generalizations of this result are given to incorporate more general smoothness and more general level set parameters s.

2 Core algorithms

We first consider algorithms that apply to any function f but are designed to perform well under certain smoothness conditions imposed on f. These algorithms utilize the given smoothness condition in the design of the algorithm. We call these *core algorithms* since they will also be used in the design of universal algorithms. We begin by discussing a core algorithm that performs well for Lipschitz functions. This simplest case is easiest to understand. This algorithm has the same spirit as those given in [7, 5] but with a different execution based on standard dyadic partitioning and tree algorithms, both of which are commonly used in the design of adaptive algorithms [2] . In the next section, we treat core algorithms which perform well for more general smoothness. Later sections of the paper describe universal algorithms.

2.1 Core algorithm for Lipschitz smoothness

We recall that the space $\mathrm{Lip}_M 1$ is defined as the set of all functions such that $|f(x)-f(y)| \le M|x-y|$ holds for all $x, y \in \Omega$. We let $\mathcal{D}$ denote the set of all dyadic subcubes of Ω and $\mathcal{D}_j$ those dyadic cubes with sidelength 2^{-j}. Each dyadic cube Q has 2^d children and each $Q \neq \Omega$ has one parent. A collection Λ of dyadic cubes is said to be a *tree* if whenever each $Q \neq \Omega$ is in Λ, then its parent is also in Λ.

Given $Q \in \mathcal{D}$, we let c_Q be the center of Q. Our algorithm defines for $k = 0, 1, \dots$ a tree Λ_k and finds the values of f at the centers c_Q of the $Q \in \Lambda_k$. These query values are then used to find an estimate $\hat{m}_k := \hat{m}_k(f)$ to $m(f)$. The value of $\hat{m}_k$ is then used to find the next tree Λ_{k+1}. The tree Λ_{k+1} contains Λ_k.

CORE(1,M) *Let $M > 0$. The algorithm takes as input any continuous function f and does the following steps for $k = 0, 1, \dots$.*
STEP 0: *We define $\Lambda_0 := \Gamma_0(M) := \{\Omega\}$ and compute $f(c_\Omega)$.*

STEP $k+1$: *If $\Gamma_k(M) \subset \mathcal{D}_k$ and Λ_k have been defined and the values $f(c_Q)$ have been computed at each center point c_Q, $Q \in \Lambda_k$, we then compute the value*

$$\hat{m}_k(f) := \min\{f(c_Q) : \ Q \in \Gamma_k\}, \tag{2.1}$$

which is our current estimate for the minimum of f. We define $\tilde{\Gamma}_k$ as the set of all cubes $Q \in \Gamma_k$ for which

$$f(c_Q) - M\sqrt{d}2^{-k-1} \le \hat{m}_k. \tag{2.2}$$

Then, we define $\Lambda_{k+1} := \Lambda_k \cup \Gamma_{k+1}$. The set $\Gamma_{k+1} = \Gamma_{k+1}(M)$ is then defined as the collection of all dyadic cubes $Q \in \mathcal{D}_{k+1}$ which are the children of a cube $Q \in \tilde{\Gamma}_k$.

Remark 2.1 *Let us observe that if $f \in \mathrm{Lip}_M 1$, and $k \geq 0$, then any true minimum of f is taken on one of the cubes in Γ_k. This is clear for $k = 0$ and so assume this is true for a given k. Now if Q is in Γ_k but not in $\tilde{\Gamma}_k$, then the true minimum of f over Q is greater than our current estimate $\hat{m}_k$ for the minimum of f and hence greater than $m(f)$. It follows that the minimum of f on Ω must be taken on one of the cubes $Q \in \tilde{\Gamma}_k$ and hence on one of the cubes in Γ_{k+1}.*

To analyze the performance of this algorithm, we impose additional structure on f through its level sets and prove the following theorem about the performance of the above algorithm.

Theorem 2.2 *Suppose that* **CORE(1,M)** *is run on a function $f \in \mathrm{Lip}_{M^*} 1$ with $M^* \leq M$ Then at step k, the following holds.*
(i) *The estimator $\hat{m}_k$ satisfies*

$$m(f) \leq \hat{m}_k \leq m(f) + M^* \sqrt{d} 2^{-k-1}. \tag{2.3}$$

(ii) *If in addition $f \in \mathcal{L}(s, L)$ for some $s > 0$, then the number n_k of point evaluations at the end of Step k, is at most*

$$n_k \leq 1 + C(d, s) L M^s \begin{cases} 2^{k(d-s)}, & 0 < s < d, \\ k, & s = d \end{cases} \tag{2.4}$$

Here $C(d, s)$ can be replaced by $C(d, s_0)$ if $0 < s \leq s_0 < d$.

Proof: Let us first prove (i). The lower estimate in (2.3) is obvious. Since $M^* \leq M$, we have from Remark 2.1 that the minimum of f on Ω is always taken at a point $\xi^* \in Q^*$ with cube $Q^* \in \Gamma_k$. Since the sidelength of Q^* is 2^{-k}, we have

$$\hat{m}_k \leq f(c_{Q^*}) = f(\xi^*) + f(c_{Q^*}) - f(\xi^*) \leq m(f) + M^* \sqrt{d} 2^{-k-1},$$

as desired.

Now, we prove (ii). Clearly n_k is the same as the cardinality of Λ_k. Since $\#(\Gamma_j) = 2^d \#(\tilde{\Gamma}_{j-1})$, for $j \geq 1$, we have

$$n_k = 1 + 2^d \sum_{j=0}^{k-1} \#(\tilde{\Gamma}_j). \tag{2.5}$$

If $Q \in \tilde{\Gamma}_j$, then for any $x \in Q$,

$$\begin{aligned} f(x) &\leq f(c_Q) + M^* \sqrt{d} 2^{-j-1} \leq \hat{m}_j(f) + M \sqrt{d} 2^{-j-1} + M^* \sqrt{d} 2^{-j-1} && (2.6) \\ &\leq m(f) + M \sqrt{d} 2^{-j-1} + 2M^* \sqrt{d} 2^{-j-1} \leq m(f) + \frac{3M}{2} \sqrt{d} 2^{-j}, && (2.7) \end{aligned}$$

where the first inequality uses that $f \in \mathbf{Lip}_{M^*} 1$, the second inequality uses (2.2), and the third inequality uses (i). Hence, from the level set assumption

$$\#(\tilde{\Gamma}_j) 2^{-jd} \leq |\{x : \ f(x) \leq m(f) + \frac{3M}{2} \sqrt{d} 2^{-j}\}| \leq L [\frac{3M}{2} \sqrt{d} 2^{-j}]^s. \tag{2.8}$$

This gives that

$$n_k \leq 1 + L 2^d [\frac{3M}{2} \sqrt{d}]^s \sum_{j=0}^{k-1} 2^{-j(s-d)}. \tag{2.9}$$

The two bounds given in (2.4) follow by estimating the last sum. □

It is useful to restate the last theorem in terms of accuracy versus the number of computations. We recall that $E_n(f)$ is the error in computing $m(f)$ from n point evaluations.

Corollary 2.3 *Suppose that* **CORE(1,M)** *is run on a function $f \in \mathrm{Lip}_{M^*}1$ with $M^* \le M$ and $f \in \mathcal{L}(s,L)$ for some $0 < s < d$. If after Step k, $n_k \ge 2$ point evaluations have been used, then*

$$E_{n_k}(f) \le C(d,s) M^* M^{\frac{s}{d-s}} \Big(\frac{L}{n_k}\Big)^{\frac{1}{d-s}}, \tag{2.10}$$

where $C(d,s)$ can be replaced by $C(d,s_0)$ as long as $s \le s_0 < d$.

Proof: From (2.4) have that $n_k \le 1 + C(d,s) L M^s 2^{k(d-s)}$ and therefore

$$2^{k(d-s)} \ge C(d,s)^{-1}(n_k - 1) L^{-1} M^{-s} \ge C(d,s)^{-1}(n_k/2) L^{-1} M^{-s}. \tag{2.11}$$

We use this in (i) of the theorem to obtain

$$E_{n_k}(f) \le M^* \sqrt{d} 2^{-k-1} \le C(d,s) M^* M^{\frac{s}{d-s}} \Big(\frac{L}{n_k}\Big)^{\frac{1}{d-s}}, \tag{2.12}$$

as desired. □

As long as $0 < s < d$, we obtain a better error decay, in terms of n, than that given in (1.3) for non-adaptive algorithms. We obtain the best bound if $M = M^*$, in which case the estimate (2.12) reads for $n = n_k$,

$$E_n(f) \le C(d,s) [M^*]^{\frac{d}{d-s}} \Big(\frac{L}{n}\Big)^{\frac{1}{d-s}}. \tag{2.13}$$

This bound then holds for all n because of the monotonicity of $E_n(f)$ and the fact that $n_k \le 2^d n_{k-1}$. As noted in (1.8), this bound was shown to be optimal in [5] for the case $s = d/2$. We show in §5 by example that it is optimal for all s.

Remark 2.4 *We want to make remarks about the* **CORE** *algorithm that will be useful when we construct universal algorithms. If at each step j of the* **CORE(1,M)** *algorithm, we use in place of $\hat{m}_j$ an estimate $\tilde{m}_j$ which is a better estimate for the true minimum, i.e. $m(f) \le \tilde{m}_j(f) \le \hat{m}_j(f)$ then we still have that the set $\tilde{\Lambda}_j \subset \Lambda_j$ and all true minima of f are attained on one of the cubes $Q \in \tilde{\Lambda}_j \cap \mathcal{D}_j$. Hence, this modified algorithm admits the same conclusions as Theorem 2.2.*

2.2 Core algorithms for higher smoothness

We next construct core algorithms with other smoothness conditions. For any $r = 1, 2, \ldots$, we define $C^r(\Omega)$ as the space of all functions in $C(\Omega)$ with continuous r-th derivatives and the norm

$$|f|_{C^r(\Omega)} := \max_{0<|\alpha|\le r} \|D^\alpha f\|_{L_\infty(\Omega)}. \tag{2.14}$$

We denote by $\mathcal{C}(r,M,d)$ the set of all functions $f \in C^r(\Omega)$ for which

$$|f|_{C^r} \le M. \tag{2.15}$$

We recall the following known results concerning functions $f \in C^r$. Let $D(d,r)$ denote the dimension of the span of x^α, with $|\alpha| < r$ in $\mathbb{R}^d$. We fix once and for all a set $S_\Omega \subset \Omega$ of $D(d,r)$

points in general position and do not indicate the dependence on this choice in going further. Any dyadic cube Q is the image of Ω under a dilation and translation. We apply this transformation to S_Ω and obtain the set S_Q. There is a unique polynomial $P_Q := P_Q(f)$ of degree $< r$ which interpolates f at the points in S_Q. Then P_Q approximates f to the following accuracy on Q

$$\|f - P_Q\|_{L_\infty(Q)} \le C(d,r)|f|_{C^r(Q)}|Q|^{r/d}, \tag{2.16}$$

where $C(d,r)$ is a constant depending only on d, r. We define

$$\hat{m}_Q := \hat{m}_Q(f) := \inf_{z\in Q} P_Q(z). \tag{2.17}$$

Let ξ be the point in Q where P_Q takes its minimum value on Q, then denoting here and later by $m(f,Q)$ the minimum of f on Q, we have

$$m(f,Q) = f(\xi) = P_Q(\xi) + f(\xi) - P_Q(\xi) \le \hat{m}_Q + C(d,r)|f|_{C^r(Q)}|Q|^{r/d}. \tag{2.18}$$

Similarly, if ξ' is the point where f takes its minimum on Q, then

$$\hat{m}_Q \le P_Q(\xi') = f(\xi') + P_Q(\xi') - f(\xi') \le m(f,Q) + C(d,r)|f|_{C^r(Q)}|Q|^{r/d}. \tag{2.19}$$

Therefore,

$$|m(f,Q) - \hat{m}_Q| \le C(d,r)|f|_{C^r(Q)}|Q|^{r/d} =: |f|_{C^r(Q)}\phi_r(|Q|)/2 \tag{2.20}$$

where we use the abbreviated notation $\phi_r(t) := 2C(d,r)t^{r/d}$, $t \ge 0$, with $C(d,r)$ the constant in (2.16). Notice that $\hat{m}_Q$ may be smaller than the true minimum $m(f,Q)$.

With these preliminaries in hand, we can now describe the core algorithm for C^r functions.

CORE(r,M) *Given a positive integer r and a positive real number M, The algorithm takes as input any continuous function f and does the following steps for $k = 0, 1, \ldots$.*

STEP 0*: We define $\Lambda_0 := \Gamma_0 := \{\Omega\}$ and query $f(z)$, for each $z \in S_\Omega$. We then compute $\hat{m}_\Omega$*

STEP $k+1$**:** *Assume $\Gamma_k = \Gamma_k(M) \subset \mathcal{D}_k$ and Λ_k have been defined and the values $f(z)$, $z \in S_Q$, have been queried and the value $\hat{m}_Q$ has been computed for each $Q \in \Lambda_k$. We then compute the value*

$$\hat{m}_k := \hat{m}_k(r,M) := \min\{\hat{m}_Q : \ Q \in \Gamma_k\}, \tag{2.21}$$

which is our current estimate for the minimum of f. We define $\tilde{\Gamma}_k$ as the set of all cubes $Q \in \Gamma_k$ for which

$$\hat{m}_Q - M\phi_r(|Q|) \le \hat{m}_k. \tag{2.22}$$

Notice that $\tilde{\Gamma}_k$ is always nonempty. The set $\Gamma_{k+1} = \Gamma_{k+1}(M)$ is then defined as the collection of all dyadic cubes $Q \in \mathcal{D}_{k+1}$ which are the children of a cube $Q \in \tilde{\Gamma}_k$. Finally, $\Lambda_{k+1} := \Lambda_k \cup \Gamma_{k+1}$.

Remark 2.5 *Notice that if $f \in \mathcal{C}(r,M,d)$ with $M > |f|_{C^r}$, and Q is in Γ_k but not in $\tilde{\Gamma}_k$, then the minimum of f over Q satisfies*

$$m(f,Q) \ge \hat{m}_Q - M\phi_r(|Q|)/2 > \hat{m}_k + M\phi_r(|Q|)/2 \ge m(f), \tag{2.23}$$

where the first and last inequalities use (2.20) *and the second uses that* (2.22) *does not hold. It follows that the minimum of f on Ω cannot occur on Q. Hence, the minimum of f on Ω must be taken on one of the cubes $Q \in \Gamma_{k+1}$.*

Theorem 2.6 *Suppose that* **CORE(r,M)** *is run with a constant $M > 0$ and that $f \in \mathcal{C}(r, M^*, d)$ with $M^* \leq M$, then the following hold.*
(ii) *The estimator $\hat{m}_k$ satisfies*

$$|m(f) - \hat{m}_k| \leq C(d,r)M^* 2^{-kr}, \tag{2.24}$$

where $C(d,r)$ is the constant in (2.16).
(ii) *If in addition $f \in \mathcal{L}(s,L)$ for some $s, L > 0$, then at step k, the algorithm has the following properties: The number n_k of point evaluations at the end of Step k, is at most*

$$n_k \leq D(d,r) + C(d,s,r)LM^s \begin{cases} 2^{k(d-s)}, & 0 < s < d, \\ k, & s = d \end{cases} \tag{2.25}$$

Here $C(d,s,r)$ can be replaced by $C(d,s_0,r)$ if $0 < s \leq s_0 < d$.

Proof: Let us first prove (i), that is, we prove (2.24). As we have already observed the minimum of f on Ω is always taken at a point $\xi^* \in Q^*$ with cube $Q^* \in \Gamma_k$. Since the measure of Q^* is 2^{-kd}, we have from (2.20),

$$\hat{m}_k \leq \hat{m}_{Q^*} \leq m_{Q^*} + C(d,r)M^* 2^{-kr} = m(f) + C(d,r)M^* 2^{-kr}. \tag{2.26}$$

On the other hand, if $\hat{m}_k = \hat{m}_Q$ with $Q \in \Gamma_k$, then

$$m(f) \leq m_Q \leq \hat{m}_Q + C(d,r)M^* 2^{-kr} = \hat{m}_k + C(d,r)M^* 2^{-kr}. \tag{2.27}$$

Hence, we have proved (i).

Now, we prove (ii). The total number of point evaluations used in the algorithm is the same as $D(d,r)\#(\Lambda_k)$. Since $\#(\Gamma_j) = 2^d \#(\tilde{\Gamma}_{j-1})$, for $j \geq 1$, we have that the number of point evaluation after step k does not exceed

$$D(d,r)\{1 + 2^d \sum_{j=1}^{k-1} \#(\tilde{\Gamma}_j)\}. \tag{2.28}$$

If $Q \in \tilde{\Gamma}_j$ and ξ is the point in Q where f assumes its minimum on Q, we have

$$\begin{aligned} f(\xi) &= m(f,Q) \\ &\leq \hat{m}_Q + C(d,r)M^*|Q|^{r/d} \\ &\leq \hat{m}_j + 3C(d,r)M|Q|^{r/d} \\ &\leq m(f) + M^*C(d,r)|Q|^{r/d} + 3C(d,r)M|Q|^{r/d} \\ &\leq m(f) + 4MC(d,r)2^{-jr}, \end{aligned} \tag{2.29}$$

where the first inequality uses (2.20), the second inequality uses (2.22), and the third inequality uses (2.24). Since $r \geq 1$, $f \in \mathrm{Lip}_{M^*}\, 1$ and so for any $x \in Q$, from (2.22), we find

$$\begin{aligned} f(x) &\leq f(\xi) + \sqrt{d}M^* 2^{-j} \\ &\leq m(f) + 4MC(d,r)2^{-jr} + \sqrt{d}M^* 2^{-j} \\ &\leq m(f) + \{4C(d,r) + \sqrt{d}\}M2^{-j}. \end{aligned} \tag{2.30}$$

Hence,

$$\#(\tilde{\Gamma}_j)2^{-jd} \le |\{x:\ f(x) \le m(f) + \{4C(d,r) + \sqrt{d}\}M2^{-j}\}| \le L[\{4C(d,r) + \sqrt{d}\}M2^{-j}]^s. \quad (2.31)$$

This gives that

$$n_k \le D(d,r)\{1 + 2^d \sum_{j=0}^{k-1} \#(\tilde{\Gamma}_j)\} \le D(d,r)\{1 + 2^d L M^s [4C(d,r) + \sqrt{d}]^s \sum_{j=0}^{k-1} 2^{j(d-s)}\}.$$

This gives the bounds stated in (ii) and completes the proof of the theorem. □

As in the previous case of Lip functions, we get that when the **CORE(r,M)** algorithm is run on a function $f \in C^r$ with $|f|_{C^r} = M^* \le M$ and $f \in \mathcal{L}(s,L)$, $0 < s < d$, then

$$E_n(f) \le C(d,r,s)M^* M^{\frac{rs}{d-s}} \Big(\frac{L}{n}\Big)^{\frac{r}{d-s}}, \quad n \ge D(d,r), \quad (2.32)$$

where n is the number of point evaluations of f that are used.

3 Universal algorithms

In the case of Lip 1 functions, the analysis of the preceding section gives the optimal rate (2.13) if $M = M^*$. Since, in general, M^* is unknown, the choice of M is problematic. One of the main points of [5] is that it is possible to design an algorithm even when a bound for M^* is not available. For example, in the case $d = 1$ and $s = 1/2$, a universal algorithm is given in [5] for which one has

$$E_n(f) \le C_0[M^*]^8 \Big(\frac{L}{n}\Big)^2. \quad (3.1)$$

While this estimate is optimal in the rate n^{-2}, the factor $[M^*]^8$ does not match the factor $[M^*]^2$ which one obtains when M^* is known and $d = 1$ and $s = 1/2$.

In this section, we shall give algorithms which do not require the knowledge of the value of M^* and yet give the following bound for Lipschitz functions in $\mathcal{L}(s,L)$ and any d,

$$E_n(f) \le C_0 M^* [M^*]^{\frac{s}{d-s}} [\log M^*]^{\frac{2}{d-s}} [L/n]^{\frac{1}{(d-s)}}, \quad n \ge 1, \quad (3.2)$$

with C_0 an absolute constant. Thus, except for the appearance of the logarithm factor, this estimate is the same as that for the CORE algorithm when M^* is known and as we have mentioned earlier is provably optimal both in the dependence on M^* and n. Let us also mention that it is possible to give a universal algorithm which satisfies

$$E_{n \log n}(f) \le C_0 M^* [M^*]^{\frac{s}{d-s}} [L/n]^{\frac{1}{(d-s)}}, \quad n \ge 1. \quad (3.3)$$

Rather than treating the case of Lip 1 smoothness separately, we shall treat the general case of C^r, $r \ge 1$, smoothness when developing our universal algorithm. We recall the core algorithm **CORE**(r,M). We will run the **CORE**(r,M) algorithm for values of $M = 2^j$, $j = 0, 1, \ldots$, but we want to run them in parallel and so we will organize the computation in a new way. Given a cube $Q \in \mathcal{D}_k$ which is being considered for subdivision in **CORE(r,** 2^j**)**, we check whether

$$\hat{m}_Q - 2^{j+1} C(d,r)|Q|^{r/d} \le \bar{m}_k, \quad (3.4)$$

where $\bar{m}_k$ is our current estimate of $m(f)$ and $C(d,r)$ is the constant in (2.16). We will use one and the same $\bar{m}_k$ when running **CORE(r,2^j)**, $j = 0, 1, \ldots$ Thus, Q will be marked for subdivision if 2^j is large enough and will not be subdivided when j is small. We can easily find the first value $j = j(Q)$ when Q is subdivided; it is the smallest integer j for which $2^{j+1}C(r,d)|Q|^{r/d} \geq \hat{m}_Q - \bar{m}_k$. Once this $j(Q)$ is found we know that Q will be subdivided by the **CORE(r,2^j)** algorithm if and only if $j \geq j(Q)$. In the universal algorithm given below, we will put this cube Q in the bucket with label j if and only it is marked to be subdivided, i.e., if and only if it satisfies (3.4) and in addition the number of cubes already in the bucket is not too large as will be precisely defined in the algorithm.

The following is our universal algorithm for C^r functions.

UNIV(r,R) *Given positive integers r, R, the algorithm takes as input any continuous function f and does the following steps for $k = 0, 1, \ldots$.*

STEP 0**:** *We put Ω in bucket j, for each $j = 0, 1, \ldots$ until $(j+1)^2 > 2^{Rd}$. We compute $f(\xi)$, for each $\xi \in S_\Omega$ and then define $\bar{m}_0 := \hat{m}_\Omega$. We denote by $\mathcal{B}_j(0)$ the set of all cubes currently in the j-th bucket, hence, in the current stage of the algorithm $\mathcal{B}_j(0) = \{\Omega\}$ for all j. We also define $\bar{\Gamma}_1$ as the collection of all children of Ω. Hence $\bar{\Gamma}_1 = \mathcal{D}_1$ at this stage.*

STEP k**:** *Given that the collection $\bar{\Gamma}_k \subset \mathcal{D}_k$ has been defined, we do the following. For each dyadic cube $Q \in \bar{\Gamma}_k$ we compute $\hat{m}_Q$, and $\bar{m}_k := \min\{\hat{m}_Q : Q \in \bar{\Gamma}_k\}$. We then find $j(Q)$ using this $\bar{m}_k$, and place Q in all buckets with label, $j \geq j(Q)$ provided that adding all such Q from this dyadic level to this bucket will still keep the number of cubes in this bucket smaller than $2^{Rd}/(j+1)^2$. Otherwise, we do not add any of the cubes from this dyadic level to the bucket with label j. This gives the collection of cubes $\mathcal{B}_j(k)$. We then define $\bar{\Gamma}_{k+1}$ as the collection of all cubes in $\mathcal{D}_{k+1}$ that are children of some $Q \in \mathcal{B}_j(k) \cap \mathcal{D}_k$ for some $j \geq 0$. Notice then that for each j, there is a final level $\ell(j)$ and the j-th bucket contains all the cubes from level $\ell(j)$ which qualify and this bucket contains no cubes from any higher dyadic level. The algorithm stops at step $K = K(r,R)$ when no cubes are added at this step. To define $\tilde{m}_R$, our estimate for $m(f)$, we let $\tilde{\Lambda}_R$ be the collection of all cubes which appear in one of the buckets and none of its children appear in any of the buckets. Each $Q \in \tilde{\Lambda}_R$ has a smallest value $\bar{j}(Q)$ for which it appears in the bucket labelled $\bar{j}(Q)$. We then define*

$$\tilde{m}_R := \min_{Q \in \tilde{\Lambda}_R} [\hat{m}_Q + 2^{\bar{j}(Q)} C(d,r)|Q|^{r/d}], \tag{3.5}$$

as our estimate for $m(f)$.

Remark 3.1 *Note that for all $j \geq 0$, we have $\ell(j) \geq \ell(j+1)$. Indeed, any cube that qualifies for the j-th bucket qualifies for the $(j+1)$-st bucket and therefore, the $(j+1)$-st bucket will fill to its quota at least as fast as the j-th bucket.*

The following theorem gives the performance of **UNIV(r,R)**

Theorem 3.2 *Suppose* **UNIV(r,R)** *is applied to a function $f \in \mathcal{C}(r, M^*, d)$, with $M^* \geq 1$. If $f \in \mathcal{L}(s, L)$ with $L \geq 1$ and $0 < s < d$, then,*
(i) *The number of point evaluations used in the algorithm is at most $C_0 D(r,d) 2^{Rd}$, with C_0 an absolute constant.*
(ii) *The estimator $\tilde{m}_R$ satisfies*

$$|m(f) - \tilde{m}_R| \leq C(d,r) M^* [[M^*]^s (\log M^*)^2]^{\frac{r}{d-s}} \left(L 2^{-Rd}\right)^{\frac{r}{d-s}}, \tag{3.6}$$

where $C(d,r)$ is a constant depending only on d and r.

Proof: (i) The number of point evaluations used in the algorithm is $D(r,d)N$ where N is the total number of cubes in all of the buckets. Since each bucket has at most $2^{Rd}/(j+1)^2$ such cubes, (i) follows.

(ii) We select $j^* \geq 1$ such that $2^{j^*-1} < M^* \leq 2^{j^*}$ and look at the cubes that are in the bucket with label j^*. The cubes in this bucket are exactly the same as what we would get when running the **CORE(r, 2^{j^*})** algorithm with the estimators $\bar{m}_k$ up to dyadic level $\ell(j^*)$. We first want to prove that

$$|\tilde{m}_R - m(f)| \leq C_0(d,r)M^* 2^{-\ell(j^*)r}. \tag{3.7}$$

Let us first prove that there is a cube Q in the j^* bucket at level $\ell(j^*)$ where $m(f,Q) = m(f)$. This is proved by showing by induction that at each level $k \leq \ell(j^*)$, there is a cube Q_0 in the bucket j^* at this level which satisfies $m(f,Q_0) = m(f)$. It is clear that Ω has this property. Now assume we have proven this property for level k and Q_0 is the cube at level k with this property. We need only show that Q_0 qualifies for subdivision. To see this let Q^* be the cube at dyadic level k which attains $\bar{m}_k$. Then, using (2.20), we find that

$$\begin{aligned} \hat{m}_{Q_0} &\leq m(f,Q_0) + C(d,r)M^*2^{-kr} \leq m(f,Q^*) + C(d,r)M^*2^{-kr} \leq \hat{m}_{Q^*} + 2C(d,r)M^*2^{-kr} \\ &= \bar{m}_k + 2C(d,r)M^*2^{-kr} \leq \bar{m}_k + 2C(d,r)2^{j^*}2^{-kr}. \end{aligned} \tag{3.8}$$

This shows that Q is indeed subdivided and so one of its children contains a point of global minima for f and the induction hypothesis is advanced.

Now, let Q denote the cube in bucket j^* at level $\ell(j^*)$ where f takes a true minimum. Then,

$$\begin{aligned} \tilde{m}_R &\leq \hat{m}_Q + C(d,r)2^{j^*}2^{-\ell(j^*)r} \leq m(f,Q) + C(d,r)(2^{j^*} + M^*)2^{-\ell(j^*)r} \\ &\leq m(f,Q) + 3C(d,r)M^*2^{-\ell(j^*)r} = m(f) + 3C(d,r)M^*2^{-\ell(j^*)r}, \end{aligned} \tag{3.9}$$

which is half of what we want to prove in (3.7).

To prove the other half, let Q be any cube in $\tilde{\Lambda}_R$. We consider two cases. The first is where $Q \in \tilde{\Lambda}_R$ is in a bucket j with $j < j^*$. By Remark 3.1, $\ell(j) \geq \ell(j^*)$, and so using (2.18)

$$\begin{aligned} \hat{m}_Q + 2^j C(d,r)|Q|^{r/d} &\geq m(f,Q) - M^*C(d,r)|Q|^{r/d} \geq m(f) - M^*C(d,r)|Q|^{r/d} \\ &\geq m(f) - M^*C(d,r)2^{-\ell(j^*)r}. \end{aligned} \tag{3.10}$$

The second case is when Q is in a bucket $j \geq j^*$. In this case,

$$\hat{m}_Q + 2^j C(d,r)|Q|^{r/d} \geq m(f,Q) + 2^j C(d,r)|Q|^{r/d} - M^*C(d,r)|Q|^{r/d} \geq m(f). \tag{3.11}$$

Now, if we take an infimum over all $Q \in \tilde{\Lambda}_R$, we obtain

$$\tilde{m}_R \geq m(f) - C_0(d,r)M^*2^{-\ell(j^*)r}, \tag{3.12}$$

which completes the proof of (3.7).

Now, let N be the number of cubes in the bucket with label j^*. We know that the cubes that accumulate in bucket j^* when running **UNIV(r,R)** are the same as those that occur when running **CORE(r,2^{j^*})** with the caveat that the $\bar{m}_k$ are used in place of the usual $\hat{m}_k$. Since $\bar{m}_k \leq \hat{m}_k$ for

each level k, we have that the number N_0 of cubes that accumulate in running **CORE(r,2^{j^*})** up to level $\ell(j^*)$ is larger than N.

From (2.25), we get, for $M = 2^{j^*}$

$$N_0 \leq C(d,r,s)LM^s 2^{\ell(j^*)(d-s)}. \tag{3.13}$$

As in the proof of Corollary 2.3 we rewrite (3.13) as

$$2^{\ell(j^*)} \geq \left(\frac{N_0}{C(d,s,r)L2^{j^*s}}\right)^{\frac{1}{d-s}} \tag{3.14}$$

and substitute it into (3.7) to get

$$E_{N_0}(f) = |\tilde{m}_R - m(f)| \leq C(d,r,s)M^* M^{\frac{rs}{d-s}}(L/N_0)^{\frac{r}{d-s}} \tag{3.15}$$

From the definition of $\ell(j^*)$ we know

$$2^{-d}2^{Rd}/[j^*+1]^2 \leq N \leq 2^{Rd}/[j^*+1]^2. \tag{3.16}$$

Hence,

$$N_0 \geq 2^{-d}2^{Rd}/[j^*+1]^2. \tag{3.17}$$

If we place this into (3.15), we derive (3.6) because $M \leq 2M^*$. □

4 Finding the minimum in high dimensions

The above results suffer from the curse of dimensionality in that the approximation rate deteriorates badly as d gets large. The usual way of avoiding this curse is to assume additional properties of f. In this section, we show that it is possible to construct an algorithm for finding the minimum of f that avoids the curse of dimensionality whenever f is in the model class $\mathcal{F}(r,\ell,M)$ introduced and analyzed in [3]. This class consists of all functions $f \in C(\Omega)$ such that

$$f(x_1,\ldots,x_d) = g(x_{i_1},\ldots,x_{i_\ell}) \tag{4.1}$$

where $g \in C^r$ and $\|g\|_{C^r} \leq M$. In other words, this class of functions is r times smooth and depends only on ℓ variables which are unknown to us. If the indices $I = \{i_1,\ldots,i_\ell\}$ were known to us then we could simply apply the **CORE** algorithm of §2.2 or the universal algorithm of §3 to the function g.

To get around the fact that we do not know I, we proceed as in [3] and use a family $\mathcal{A} = \mathcal{A}(\ell,d)$ of partitions $\mathbf{A}$ of $\{1,2,\ldots,d\}$. Each $\mathbf{A}$ consists of ℓ disjoint sets $\mathbf{A} = \{A_1,\ldots,A_\ell\}$ which satisfy

Partition Assumption: *The collection $\mathcal{A}$ is rich enough so that given any ℓ distinct integers $i_1,\ldots,i_\ell \in \{1,\ldots,d\}$, there is a partition $\mathbf{A}$ in $\mathcal{A}$ such that each set in $\mathbf{A}$ contains precisely one of the integers $i_1,\ldots,i_\ell$.*

This **Partition Assumption** is known as perfect hashing in theoretical computer science. We are interested in such families with small cardinality. For general d and ℓ, such collections are constructed using probability. It is known that such collections exist with

$$\#(\mathcal{A}(\ell,d)) \leq 2\ell e^\ell \ln d. \tag{4.2}$$

Remark 4.1 *Let us note that although probability theory is used to prove the existence of a class of partitions $\mathcal{A}$ with the favorable probability bound* (4.2)*, once such $\mathcal{A}$ is in hand, the results that follow in this section hold without any probability restriction.*

Given a set $B \subset \{1, \ldots, d\}$, we denote by χ_B the point in Ω which takes the value one at each coordinate $i \in B$ and otherwise takes the value zero. Corresponding to any $\mathbf{A} \in \mathcal{A}$ and any d variable function f, we define the function

$$f_{\mathbf{A}}(t_1, \ldots, t_\ell) = f(\sum_{i=1}^{\ell} t_i \chi_{A_i}), \quad 0 \le t_i \le 1, \ i = 1, \ldots, \ell. \tag{4.3}$$

Notice that asking for the value of $f_{\mathbf{A}}$ at any point $\mathbf{t} = (t_1, \ldots, t_\ell)$ is the same as asking for the value of f at the point $\sum_{i=1}^{\ell} t_i \chi_{A_i}$ which in turn is the same as $g(t_{j_1}, \ldots, t_{j_\ell})$ where j_k is the index i of the set A_i which contains k. Whenever $f \in \mathcal{C}(r, M^*, d)$, each of the $f_{\mathbf{A}}$ is in $\mathcal{C}(r, CM^*, \ell)$ with a suitable constant C. Indeed, taking derivative of $f_{\mathbf{A}}$, we find,

$$|f_{\mathbf{A}}|_{C^r} \le C(d, r)|f|_{C^r}, \quad \mathbf{A} \in \mathcal{A}, \tag{4.4}$$

where the constant $C(d, r)$ depends only on d and r.

In going further, we assume that $f \in \mathcal{F}(r, \ell, M)$. From the partition assumption, we know that for one of the $\mathbf{A}^* \in \mathcal{A}$, we have that each of the partition sets $A_1^*, \cdots, A_\ell^*$ contains exactly one change coordinate of f. Thus, up to a relabelling of indices, $g = f_{\mathbf{A}^*}$. Thus the minimum of f is the same as the minimum of g over $[0, 1]^\ell$ and this is the same as the minimum of $f_{\mathbf{A}^*}$ on $[0, 1]^\ell$. Of course, we do not know $\mathbf{A}^*$. So our strategy is to apply the minimization algorithms of the previous section to each of the functions $f_{\mathbf{A}}$ and then take the smallest of the minima produced. While we can apply this strategy to the **CORE** algorithm, we formulate and analyze only the universal algorithm of the previous section.

Let **UNIV(r,R)** be the universal algorithms for C^r functions described in the previous section.

HDUNIV(r,R): *This algorithm takes as input a function f of d variables and does the following:*
STEP 1**:** *For each $\mathbf{A} \in \mathcal{A}$, we apply* **UNIV(r,R)** *to $f_{\mathbf{A}}$ thereby obtaining the empirical estimate* $\tilde{m}_R(f_{\mathbf{A}})$ to $m(f_{\mathbf{A}})$
STEP 2**:** *We define* $\tilde{m}_R(f) := \min_{\mathbf{A} \in \mathcal{A}} \tilde{m}_R(f_{\mathbf{A}})$.

The following describes the performance of the algorithm **HDUNIV(r,R)**.

Theorem 4.2 *Suppose $f(x_1, \ldots, x_d) = g(x_{i_1}, \ldots, x_{i_\ell})$ with $g \in \mathcal{C}(r, M^*, \ell)$ with $M^* \ge 1$. If $g \in \mathcal{L}(s, L)$ with $0 < s < \ell$ and $L \ge 1$, then applying* **HDUNIV(r,R)** *to f gives*
(i) *The total number of point evaluation does not exceed $C(\ell, r)[\log d]2^{R\ell}$.*
(ii) *We have the accuracy estimate*

$$|m(f) - \tilde{m}_R| \le C(\ell, r)M^*[[M^*]^s(\log M^*)^2]^{\frac{r}{\ell - s}} \left(L2^{-R\ell}\right)^{\frac{r}{\ell - s}}. \tag{4.5}$$

Proof: (i) From Theorem 3.2, the number of evaluations used in computing $\tilde{m}_R(f_{\mathbf{A}})$ does not exceed $C(\ell, r)2^{R\ell}$. Since there are at most $\ell 2^\ell \ln d$ functions $f_{\mathbf{A}}$, we have proved (i).

(ii) In view of part (ii) of Theorem 3.2 and the estimate for derivatives given in (4.4), we know that for each $\mathbf{A} \in \mathcal{A}$, we have

$$|m(f_{\mathbf{A}}) - \tilde{m}_R(f_{\mathbf{A}})| \le C(\ell, r)M^*[[M^*]^s(\log M^*)^2]^{\frac{r}{\ell - s}} \left(L2^{-R\ell}\right)^{\frac{r}{\ell - s}}. \tag{4.6}$$

Since $m(f) = \min_{\mathbf{A} \in \mathcal{A}} m(f_{\mathbf{A}})$ and $\tilde{m}_R(f_{\mathbf{A}}) = \min_{\mathbf{A} \in \mathcal{A}} \tilde{m}_R(f_{\mathbf{A}})$, the bound (4.5) follows from (4.6) □

5 Optimality

We give examples that show that our estimates are in a certain sense optimal. We restrict our attention to the case of $\mathrm{Lip}_{M^*}1$ spaces. Our examples are a modification of examples given in [5]. Suppose we have an arbitrary algorithm which (adaptively or not) queries the function f in points $x_1, x_2, \ldots, x_n$ and based on values $f(x_1), f(x_2), \ldots, f(x_n)$ gives an approximation $\hat{m}_n(f)$ to a true minimum $m(f)$. We are interested in giving a lower bound for

$$E_n = \sup_{f \in \mathrm{Lip}_{M^*}1 \cap \mathcal{L}(s,L)} |m(f) - \hat{m}_n(f)|. \tag{5.1}$$

In going further, we fix the numbers $M^* > 1$, $L > 2$, and $0 < s < d$ and for an integer N, we define

$$\alpha := \left(\frac{M^*}{2N}\right)^{\frac{s}{d-s}} L^{\frac{1}{d-s}}; \quad \beta := \left(\frac{M^*}{2N}\right)^{\frac{d}{d-s}} L^{\frac{1}{d-s}} = \frac{M^*}{2N}\alpha, \tag{5.2}$$

where in going further we always require that N is large enought that $\alpha, \beta \le 1$. We then define the function

$$F(x) := \begin{cases} \beta, & \text{if } x \in [0,\alpha]^d \\ \beta + \mathrm{dist}(x, [0,\alpha]^d), & \text{if } x \notin [0,\alpha]^d. \end{cases} \tag{5.3}$$

Lemma 5.1 *Let $Q \subset [0,\alpha]^d$ be any cube of sidelength $l \ge \alpha/N$ and let*

$$g(x) := g_Q(x) := \begin{cases} F(x), & \text{if } x \notin Q \\ F(x) - M^* \mathrm{dist}(x, \partial Q), & \text{if } x \in Q. \end{cases} \tag{5.4}$$

Then, for N sufficiently large (depending on s, d, M^*, L), $g \in \mathrm{Lip}_{M^*}1 \cap \mathcal{L}(s,L)$.

Proof: That $g \in \mathrm{Lip}_{M^*}1$ is immediate from its definition. We now check that $g \in \mathcal{L}(s,L)$. Notice that $m(g) = 0$ when $l = \alpha/N$ and is negative when $l \ge \alpha/N$. To check that $g \in \mathcal{L}(s,L)$ we consider three cases.

CASE $\delta < M^* l/2$: In this case, $\{g(x) \le m(g) + \delta\}$ is a cube with center c_Q and sidelength $2\delta/M$. So we must show that $(2\delta/M^*)^d \le L\delta^s$, i.e., $\delta \le [M^*]^{\frac{d}{d-s}} L^{\frac{1}{d-s}} 2^{-d/(d-s)}$. Since $l \le \alpha$, we only need to show that $M^*\alpha/2 < M^{\frac{d}{d-s}} L^{\frac{1}{d-s}} 2^{-d/(d-s)}$. From the definition of α, we see that this holds for any $N \ge 1$.

CASE $\delta = M^* l/2$: In this case, we see that $\{g(x) \le m(g) + \delta\} = [0,\alpha]^d$. So, it suffices to notice that $L(M^* l/2)^s \ge L(M^*\alpha/(2N))^s = \alpha^d$.

CASE $\delta > M^* l/2$: we see that $\{x \in \Omega : g(x) \le m(g) + \delta\} = \{x \in \Omega : F(x) \le m(g) + \delta\}$. Since $m(g) \le 0$ we can write $m(g) + \delta = \beta + \eta$ with $0 \le \eta \le \delta$. This gives

$$\{x : F(x) \le m(g) + \delta\} \subset [0, \alpha + \eta]^d \cap [0,1]^d, \tag{5.5}$$

and so it is enough to prove that $(\alpha + \eta)^d \le L(\beta + \eta)^s$ or equivalently

$$\alpha + \eta \le L^{1/d}(\beta + \eta)^{s/d}, \quad 0 \le \eta \le 1 - \alpha. \tag{5.6}$$

As functions of η, the function on the right of (3.15) is concave and the one on the left is linear. It suffices to check that the inequality holds at the two endpoints $\eta = 0, 1 - \alpha$. When $\eta = 0$, the two functions appearing in(5.6) are equal because of the definition of α and $\beta = \frac{M^*}{2N}\alpha$. When $\eta = 1 - \alpha$, (5.6) holds because $L \ge 2$ and both α and β tend to zero as N gets large. □

Theorem 5.2 *Let $M^* > 1$ and $L > 2$ and any $0 < s < d$. Consider any algorithm for finding the minimum of a function f and its performance E_n on the class $\mathrm{Lip}_{M^*}1 \cap \mathcal{L}(s, L)$ given by* (5.1)*. Then, for any n sufficiently large we have*

$$E_n \geq C(d, s)[M^*]^{\frac{d}{d-s}} (L/n)^{\frac{1}{d-s}}. \tag{5.7}$$

Proof: We fix a value of N that is an even integer and such that Lemma 5.1 is valid. We consider any integer n with $n < (N/2)^d$. We run the proposed algorithm on F and obtain the points $x_1, \ldots, x_n$ and the values $F(x_j)$. We divide $[0, \alpha]^d$ into $[N/2]^d$ disjoint cubes with equal sidelength $2\alpha/N$. Clearly there exists at least one such cube Q which does not contain any of the points $x_1, \ldots, x_n$. In addition we choose a subcube $Q_1 \subset Q$ with sidelength α/N. For each of these cubes we construct the functions g_Q and g_{Q_1} of Lemma 5.1 which are both in our class $\mathrm{Lip}_{M^*}1 \cap \mathcal{L}(s, L)$.

Applying the algorithm to these two functions is the same as applying it to F and hence the algorithm gives the same estimate $\hat{m}_n(g_Q) = \hat{m}_n(g_{Q_1})$. We now check that the true minima of these two functions are quite different. Indeed, we have

$$m(g_{Q_1}) = F(c_{Q_1}) - M^* \frac{\alpha}{2N} = \beta - M^* \frac{\alpha}{2N} = 0$$

and

$$m(g_Q) = F(c_Q) - M^* \alpha/N = \beta - M^* \frac{\alpha}{N} = -M^* \frac{\alpha}{2N}.$$

It follows that

$$E_n \geq M^* \frac{\alpha}{4N}, \quad n < (N/2)^d. \tag{5.8}$$

Given any n sufficiently large (depending on s, L, M^*, d), we can take N as the first even integer larger than $2n^{1/d}$ and use this value of N in (5.8) to obtain (5.7). □

References

[1] A. Cohen, I. Daubechies, R. DeVore, G. Kerkyacharian, and D. Picard, *Capturing Ridge Functions in High Dimensions from Point Queries*, Constructive Approximation, **35** (2012), 225–243.

[2] R. DeVore, *Nonlinear approximation*, Acta Numerica, **7**(1998), 51–150.

[3] R. DeVore, G. Petrova, and P. Wojtaszczyk, *Approximation of functions of few variables in high dimension*, Const. Approx., **33** (2011) 125–143.

[4] M. Fornasier, K. Schnass, and J. Vybiral, *Learning functions of few arbitrary linear parameters in high dimensions* J. Found. Comput. Math., to appear

[5] M. Horn , *Optimal algorithms for global optimization in case of unknown Lipschitz constant*, J. of Complexity **22** (2006), 50-70.

[6] E. Novak and H. Wozniakowski, *Tractability of Multivariate Problems, Volume I: Linear Information*, EMS Tracts in Mathematics, Vol. 6 Eur. Math. Soc. Publ. House, Zrich 2008

[7] A. Perevozchikov, *The complexity of the computation of the global extremum in a class of multiextremum problems*, USSR Comp. Math. and Math. Physics **30** (1990), 28-33.

Albert Cohen
UPMC Univ Paris 06, UMR 7598, Laboratoire Jacques-Louis Lions, F-75005, Paris, France
cohen@ann.jussieu.fr

Ronald DeVore
Department of Mathematics, Texas A&M University, College Station, TX 77840, USA
rdevore@math.tamu.edu

Guergana Petrova
Department of Mathematics, Texas A&M University, College Station, TX 77840, USA
gpetrova@math.tamu.edu

Przemysław Wojtaszczyk
Interdisciplinary Center for Mathematical and Computational Modelling, University of Warsaw, 00-838 Warsaw, ul. Prosta 69, Poland.
wojtaszczyk@mimuw.edu.pl