# LEAST SQUARES AND SMOLYAK'S ALGORITHM

JAKOB EGGL, ELIAS MINDLBERGER AND MARIO ULLRICH

Abstract. We present novel, large-scale experiments for polynomial interpolation in high dimensional settings using some of the most popular algorithms available. We compare Smolyak's Algorithm (SA) on sparse grids and Least Squares (LSQ) using random data. We empirically confirm that interpolation using LSQ performs equally as good on smooth and better on non-smooth functions if SA is given $n$ and LSQ is given $2 \cdot n$ points.
Code available at: `https://github.com/th3lias/NumericalExperiments`

## 1. Introduction

Smolyak's Algorithm on Sparse Grids has been of high theoretical and practical interest for a long time.

## 2. Notation

We denote the indexset containing all indices $i \in \mathbb{Z}$ where $m \leq i \leq n$ for $m \leq n$ with $[m : n]$. For the set of polynomials $p \colon V \to W$ of maximal degree $N$ and sets $V, W \subseteq \mathbb{C}^K$, $K \in \mathbb{N}$ we use the notation $p \in \mathcal{P}^N(V, W)$. ***I mean this is the general notation but do we even have polynomials that map to $\mathbb{C}^K$.*** We use $f \propto g$ for functions $f, g$ to denote $f = cg$ for a constant $c \in \mathbb{R}$ and $f \lesssim g$ to denote $f \leq cg$. With $B(X)$ we denote the closed unit ball of a normed space $X$, i.e. $B(X) := \{x \in X : \|x\|_X \leq 1\}$. With $X^\star$ we denote the space of linear, bounded functionals on $X$, i.e. $X^\star := \{x^\star \colon X \to \mathbb{R} \mid x^\star \text{ is linear and bounded}\}$.

## 3. Interpolation

Given $i \in \mathbb{N}$ and distinct data points $\mathbf{X}_i := \{x_{i,0}, \dots, x_{i,n}\} \subseteq D \subseteq \mathbb{R}$ and function values $\mathbf{Y}_i := \{y_{i,0}, \dots, y_{i,n}\} = \{f(x_{i,0}), \dots, f(x_{i,n})\} \subseteq \mathbb{R}$ for $f : D_i \to \mathbb{R}$, it is a well known fact that there exists a polynomial $p$ of degree $n$ such that

$$p(x_{i,j}) = y_{i,j}, \quad j \in [0 : n].$$

*Date*: April 11, 2025.
*Key words and phrases.* Polynomial interpolation, Sparse-Grids, Least-Squares, Smolyak.

We then call $p$ an *interpolating* polynomial. It can be written down explicitly as

$$p = \sum_{j=0}^{n} y_{i,j} \ell_{i,j} \tag{3.1}$$

for basis functions $\ell_{i,j} \in \mathcal{P}^n(\mathbb{R}, \mathbb{R})$. Moreover, in a single dimension, the formula for this basis is, quite intuitively, $\ell_{i,j}(w) = \prod_{m \in [0:n] \setminus \{j\}} \frac{w - x_{i,m}}{x_{i,j} - x_{i,m}}$ [12, 16]. We call this type of interpolation *Lagrange interpolation*.

Suppose now we have $d$ point sets $\mathbf{X}_i$. By taking the full tensor product of point sets

$$\mathbf{X} := \bigotimes_{i=1}^{d} \mathbf{X}_i,$$

we may identify our grid by $\mathbf{X} = \{\mathbf{x_j}\}_{\mathbf{j}}$ and suppose again we have according function evaluations $\mathbf{Y} := \{y_{\mathbf{j}}\}_{\mathbf{j}} = \{f(\mathbf{x_j})\}_{\mathbf{j}}$ with $\mathbf{j} \in \{0, 1, \ldots, n\}^d$ for $f : D \to \mathbb{R}$, $D \subseteq \mathbb{R}^d$ and $(n+1)^d$ points. Then, we may construct an interpolating polynomial as in (3.1). Indeed, the general formula stays the same but we take tensor product of basis functions, i.e.

$$\ell_{\mathbf{j}}(\mathbf{w}) = \bigotimes_{i=1}^{d} \ell_{j_i}(w_i) = \prod_{i=1}^{d} \ell_{j_i}(w_i) = \prod_{i=1}^{d} \prod_{\substack{k=0 \\ k \neq j_i}}^{n} \frac{w_i - x_{i,k}}{x_{i,j_i} - x_{i,k}} \tag{3.2}$$

and the sum now ranges over all multiindices $\mathbf{j} = (j_1, \ldots, j_d) \in \mathbb{N}_0^d$ with $\mathbf{j} \in \{0, 1, \ldots, n\}^d$. Formula 3.2 is not a great choice for performing calculations on hardware. Leaving the storage of $(n+1)^d$ grid points aside, for computing one evaluation of the map $\mathbf{w} \mapsto p(\mathbf{w})$, one must compute the sum of $(n+1)^d$ terms, where in each summand, the evaluation of $d$ one dimensional basis functions is computed, each of which need a product over $n$ terms. Hence, this evaluation is as costly as $\mathcal{O}((n+1)^d dn)$. Although one can counter the costly computation of the interpolating polynomial at a given point by reformulating the above (numerically instable) Lagrange interpolator in it's barycentric form

$$p(\mathbf{w}) = \frac{\sum_{j_1=0}^{n} \ell_{j_1}^{(1)}(w_1) \cdots \sum_{j_d=0}^{n} \ell_{j_d}^{(d)}(w_d) y_{\mathbf{j}}}{\sum_{j_1=0}^{n} \ell_{j_1}^{(1)}(w_1) \cdots \sum_{j_d=0}^{n} \ell_{j_d}^{(d)}(w_d)} \tag{3.3}$$

where

$$\ell_{j_k}^{(k)}(w) := \frac{m_{j_k}}{w - \mathbf{x}_{\mathbf{j}_k}} \mid k \in [1:d]. \tag{3.4}$$

and $m_{j_k}$ are the *barycentric* weights which can be computed in $\mathcal{O}(n^2)$ time for general point distributions but admit $\mathcal{O}(1)$ computation for well–known and widely used point sets [10]. For example, due to [2],

$$m_0 = \frac{1}{2}, \quad m_l = (-1)^l,\ l \in [1 : n-1],\ m_n = \frac{1}{2}(-1)^n \tag{3.5}$$

for the *Chebyshev Gauss-Lobatto* (or just Chebyshev extrema), in which

$$x_l := x_{l,n} := -\cos\frac{l\pi}{n},\ l \in [0 : n].$$

Similarly, for the Chebyshev points of the second kind

$$x_l = x_{l,n} = \cos\frac{l\pi}{n},\ l \in [0 : n] \tag{3.6}$$

one gets the closed form

$$m_l = (-1)^l \cdot \begin{cases} 1/2 & l \in \{0, n\} \\ 1 & \text{else.} \end{cases} \tag{3.7}$$

In this case, one achieves an evaluation time of $\mathcal{O}((n+1)^d)$.

## 4. Smolyak's Algorithm & Sparse Grids

The question arises, how one may reduce the complexity of exact interpolation on grids. The central idea of sparse grids is to *not* take the full tensor product of one dimensional point sets. Firstly, take a variable number of points in each direction, i.e. let $\mathbf{X}_i := \left\{x_{1,i}, \ldots, x_{N(i),i}\right\}$ and take $\mathcal{I}_0 := 0$, $\Delta_i := \mathcal{I}_i - \mathcal{I}_{i-1}$. Then Smolyak's algorithm is given in a simple recursive manner

$$\mathcal{A}(q,d) := \sum_{\|\mathbf{j}\|_1 \le q} \Delta_{j_1} \otimes \cdots \otimes \Delta_{j_d} = \mathcal{A}_{q-1,d} + \sum_{\|\mathbf{j}\|_1 = q} \Delta_{j_1} \otimes \cdots \otimes \Delta_{j_d} \tag{4.1}$$

and $\mathcal{A}_{d-1,d} = 0$. Evidently, in no iteration of the algorithm do we have to discard old values.

## 5. Smolyak's Algorithm

Smolyak's Algorithm is determinstic and can be used in various settings. These include but are not limited to

(1) function interpolation,
(2) numerical integration,
(3) solving ODEs and PDEs,
(4) $\cdots$

**Interpolation in one dimension.** In the onedimensional setting it is well known that for a given function $f\colon \mathbb{R} \to \mathbb{R}$ and a point collection $\mathbf{Z} := \{z_n\}_{n=0}^N \subseteq \mathbb{R}$, containing of $N+1$ different points, there exists an interpolating polynomial $i \in \mathcal{P}^N(\mathbb{R}, \mathbb{R})$ such that for all $j \in [0:N]$ we have $i(z_j) = f(z_j)$. To find the polynomial $i$, one can, among other available methods, choose to perform Lagrange Interpolation which yields an explicit construction of the form

$$\mathcal{I}(\mathbf{F})(x) = \sum_{n=0}^{N} f(z_n)\ell_n(x) \quad \text{s.t.} \quad \ell_n(x) := \prod_{\substack{j=0 \\ j\neq n}}^{N} \frac{x - z_j}{z_n - z_j} \tag{5.1}$$

where $\mathbf{F} := \{[z_n, f(z_n)]\}_{n=0}^N$. We set $i := \mathcal{I}(\mathbf{F})$. Note that $\mathcal{I}$ is a mapping from data, which may be seen as arranged in a $2 \times N$ matrix, to an $N$-degree polynomial. Thus $\mathcal{I}\colon \mathbb{R}^{2\times N} \to \mathcal{P}^N(\mathbb{R}, \mathbb{R})$, since each basis function $\ell_n(x)$ is a polynomial of degree $N$. Therefore, if $f \in \mathcal{P}^N(\mathbb{R}, \mathbb{R})$, one obtains $i = f$. Moreover, $i$ is interpolating since $\ell_n(z_j) = \delta_n(j)$.

Even though (5.1) has nice theoretical properties, it is suboptimal for practical settings due to the following facts. (1) If a node $z_n$ is close to a node $z_j$ such that $j \neq n$, computing the product $\ell_n(x)$ becomes numerically unstable. (2) The addition of new data $(z, f(z))$ requires to recalculate all basis functions again in $\mathcal{O}(N^2)$ time. (3) To compute $i(x)$ in this form requires computational complexity of $\mathcal{O}(N^2)$. This does not mean that one has to abandon Lagrange Interpolation at once. We may write (5.1) in a much more stable way as

$$i(x) := \mathcal{I}(\mathbf{F})(x) = \frac{\sum_{n=0}^{N} \frac{w_n}{x - z_n} f(z_n)}{\sum_{n=0}^{N} \frac{w_n}{x - z_n}} \tag{5.2}$$

for $w_n \in \mathbb{R} : n \in [0:N]$. If the weights $w_n$ are known, computing $i(x)$ in this form admits a complexity of $\mathcal{O}(N)$. Luckily, the weights have a closed-form, analytic solution for many deterministic point sets used in practice and are hence considered to be computable in $\mathcal{O}(1)$

time. Thus, adding a new data pair $(z, f(z))$ requires total complexity of $\mathcal{O}(N)$ for the recomputation of said weights. The general identity

$$w_n = \frac{1}{\ell'(x_n)}$$

always holds. A survey on the *barycentric* form of Lagrange interpolation can be found in [2]. To specify the above, consider the set of *equidistant* nodes $\{z_n := 2/n\}_{n=0}^{N} \subset [-1, 1]$. Then

$$w_n = (-1)^n \binom{N}{n}. \tag{5.3}$$

For large $N$, (5.3) is problematic since weights $w_i, w_j : i \neq j$ now may vary by factors as large as $\mathcal{O}\left(2^N\right)$. This means that interpolation in equispaced points is susceptible to the *Runge phenomenon*. For this interpolation problem to be well posed one should use point sets with asymptotic density $\rho(x) \propto 1/\sqrt{1-x^2}$. This is called the *Chebyshev* density.

Many point sets admit this asymptotic density, for example the Chebyshev points of the first kind are given as the roots of the $n+1$-th Chebyshev polynomial and thus given by

$$\mathbf{Z}_C^{(1)}(N) := \left\{ \cos \frac{(2n+1)\,\pi}{2N+2} \mid n \in [0:N] \right\}. \tag{5.4}$$

Similarly, the Chebyshev points of the second kind are defined

$$\mathbf{Z}_C^{(2)}(N) := \left\{ \cos \frac{n\pi}{N} \mid n \in [0:N] \right\}. \tag{5.5}$$

In either of these cases, the weights $w_n$ admit nice closed forms as $w_n^{(k)} : k \in \{1, 2\}$ dependent on which point set is in use as

$$w_n^{(1)} = (-1)^n \sin \frac{(2n+1)\,\pi}{2n+2}, \quad w_n^{(2)} = (-1)^n \cdot \begin{cases} 1/2 & n \in \{0, N\} \\ 1 & \text{else.} \end{cases} \tag{5.6}$$

**Interpolation in $d$ dimensions.** In principle, one may just *extend* one dimensional interpolation rules to dimension $d$ by forming the tensor product of all interpolation rules. To specify, take the one dimensional algorithm

$$\mathcal{I}^{(N)}(\mathbf{F}(N))(x) = \sum_{n=1}^{m(N)} b_n(x) f(z_n) \tag{5.7}$$

such that $\mathcal{I}^{(N)}(\mathbf{F}(N))$ is interpolating on the data $\mathbf{F}(N) := \{[z_n, f\,(z_n)]\}_{n=1}^{m(N)}$ with basis functions $b_n \in C[0,1]$ by having $b_n(z_j) = \delta_n(j)$. Note that $m \colon \mathbb{N}_0 \to \mathbb{N}_0$ is a common abstraction for the number of points used to

interpolate a given function. In the case of special interpolation methods or their corresponding underlying point set, $m$ will be specified. To obtain a set of $d$-dimensional point sets from $d$ such one-dimensional rules like (5.4) or (5.5), one uses the usual cartesian product

(5.8)
$$\mathbf{Z}(N,d) := \mathbf{Z}(N_1) \times \cdots \times \mathbf{Z}(N_d) := \bigcup_{n_1=1}^{m(N_1)} \cdots \bigcup_{n_d=1}^{m(N_d)} \left\{ \left( z_{n_1}^{(1)}, \ldots, z_{n_d}^{(d)} \right) \right\}$$

where $z_j^{(i)}$ denotes the $j$-th point of the $i$-th rule and $N := (N_1, \ldots, N_d) \in \mathbb{N}_0^d$ and $z_n := (z_{n_1}, \ldots, z_{n_d})$ as the usual notation for multi-indices. Using the tensor product formula, one obtains the very general expression

$$\left( \mathcal{I}^{(N_1)} \otimes \cdots \otimes \mathcal{I}^{(N_d)} \right)(\mathbf{F}) = \sum_{n_1=1}^{m(N_1)} \cdots \sum_{n_d=1}^{m(N_d)} \left( b_{n_1} \otimes \cdots \otimes b_{n_d} \right) f(z_n) \tag{5.9}$$

where $\mathbf{F} := \bigcup_{j=1}^{d} \mathbf{F}(N_j)$. In this case, the tensor product interpolation is a mapping of the form

$$\bigotimes_{j=1}^{d} \mathcal{I}^{(N_j)} \colon \mathbb{R}^{d \times \left( \prod_{j=1}^{d} m(N_j) \right)} \to \mathcal{P}^{(\max\{m(N_j)\})} \left( \mathbb{R}^d, \mathbb{R} \right). \tag{5.10}$$

We denote $\mathcal{I}^{(N)} := \bigotimes_{j=1}^{d} \mathcal{I}^{(N_j)}$. If the basis functions $b_n$ are scalar-valued, their tensor product is just

$$\left( \bigotimes_{n=1}^{d} b_n \right)(x_1, \ldots, x_d) := \prod_{n \in [d]} b_n(x_n). \tag{5.11}$$

In Smolyak's Algorithm, the function $m \colon \mathbb{N}_{\to} \mathbb{N}_0$ is used as a growth rule specifying how many points are to be used by the interpolant. Specifically, we will use a doubling rule such that

$$m(n) := \begin{cases} 2^{n-1} + 1 & n \geq 2 \\ 1 & \text{else.} \end{cases}$$

Clearly, the usual representation of the Lagrange polynomial (5.1) fits this framework. For the barycentric form, see [2], one obtains the $d$-variate interpolation

$$\mathcal{I}(\mathbf{F})(x) := \frac{\sum_{n_1=1}^{m(N_1)} b_{n_1}^{(1)}(x_1) \cdots \sum_{n_d=1}^{m(N_d)} b_{n_d}^{(d)}(x_d) f(z_{n_1}, \ldots, z_{n_d})}{\sum_{n_1=1}^{m(N_1)} b_{n_1}^{(1)}(x_1) \cdots \sum_{n_d=1}^{m(N_d)} b_{n_d}^{(d)}(x_d)} \tag{5.12}$$

where

$$b_{n_k}^{(k)}(x) := \frac{w_{n_k}}{x - z_{n_k}} \mid k \in [d]. \tag{5.13}$$

If a rule yielding admissible point sets, like Chebyshev's rules (5.4) or (5.5), is available, the only thing left to do is to specify how many nodes $m(N_j)$ to pick in each dimension. If this number is large simultaneously for all dimensions, (5.9) becomes intractable very quickly. Smolyak's algorithm defines a resolution-like number $q \in \mathbb{N}$ to specify how close-meshed points should be picked. It proceeds by taking all multi-indices $N$ with 1-norm less equal to $q$ and specifies to interpolate with the algorithm

$$\mathcal{A}(q,d)\,(\mathbf{X}) := \sum_{\|N\|_1 \leq q} \mathcal{I}^{(N)}(\mathbf{X}(N)). \tag{5.14}$$

Note that, with this construction $q \geq d$ is necessary. The set $\mathbf{X}$ in this case is

$$\mathbf{X}(N) := \bigcup_{n \in [d]} \mathbf{F}(N_n) \tag{5.15}$$

All points used by this algorithm are then

$$\mathbf{X} := \mathbf{X}^{\leq}(q) := \bigcup_{\|N\|_1 \leq q} \mathbf{X}(N). \tag{5.16}$$

We call $\mathbf{X}$ a *sparse grid*. We use $\mathbf{X}^{=}(q)$ for denoting the corresponding set replaced with the rule that it takes all vectors $N_0 \in \mathbb{N}^d$ with $\|N\|_1 = q$. In case the point set is nice, for example for Chebyshev points, one obtains a *nesting* of sparse grids for increasing fineness scales: $\mathbf{X}^{\leq}(q) \subset \mathbf{X}^{\leq}(q+1)$. This construction has the advantage that in the case of applying this algorithm on a computer, storing whole grid $\mathbf{X}$ in the memory is not necessary. It is enough to use the refinements of each level-increase $q \mapsto q+1$ and interpolate at each such level separately, each time discarding the old values. We may thus define the difference operator

$$\Delta^{(q)} := \left( \bigotimes_{\|N\|_1 = q+1} \mathcal{I}^{(N)} \right) - \left( \bigotimes_{\|N\|_1 = q} \mathcal{I}^{(N)} \right) \tag{5.17}$$

Now, Smolyak's algorithm can be written as

$$\mathcal{A}(q,d) := \sum_{\|N\|_1 \leq q} \Delta^{(q)}(\mathbf{X}(N)) \tag{5.18}$$

or equivalently

$$\mathcal{A}(q,d) = \mathcal{A}(q-1,d) + \sum_{\|N\|_1 = q} \mathcal{I}^{(N)}(\mathbf{X}(N)) \tag{5.19}$$

with $\mathcal{I}^{(0)} = 0$.

## 6. Least Squares

Contrary to the construction of exactly interpolating approximants in the case of Smolyak's algorithm, Least Squares is a conceptually simpler algorithm. We are given the overdetermined system

$$Vz = y$$

where $V \in \mathbb{R}^{n\times m}$ with $n > m$. It is well–known that this system may be inconsistent and no exact solution exists. However, one may always pose the optimisation problem solving for $z \in \mathbb{C}^m$ with the smallest error

$$\inf_{z\in\mathbb{R}^m} \|Vz - y\| . \tag{6.1}$$

It is further known that, in case of a full–rank matrix $V$, the unique solution to (6.1) is given by

$$z^\star = \left(V^\top V\right)^{-1} V^\top y \in \mathbb{R}^m. \tag{6.2}$$

In our specific case of polynomial interpolation, $V$ is the Vandermonde matrix, consisting of basis polynomials $b_1, b_2, \ldots, b_m$ evaluated at the $n$ different sampled points $x_1, x_2, \ldots, x_n$ in $\Omega \subseteq \mathbb{R}^d$ and $y$ is the vector of function values sampled from the unknown function $f\colon \Omega \to \mathbb{R}$, i.e. $y = (f(x_j))_{j=1}^n$. As for such points, the Vandermonde matrix is never singular, the solution to the approximation problem

$$\inf_p \|f - p\|$$

can analytically be expressed as

$$p^\star\colon \Omega \to \mathbb{R}, t \mapsto \sum_{j=1}^m z_j^\star b_j(t)$$

where $z^\star = (z_j\star)_{j=1}^m$ is given by (6.2).

## 7. Theoretical Guarantees

The notation in the following is borrowed from [15]. In this section we introduce a formal setting to the former considerations. That is, we consider a Hilbert space $H$ of real-valued functions on a set $D$ such that point evaluation

$$\delta_x : f \mapsto \int_D f \, \mathrm{d}\delta_x = f(x)$$

is a bounded, linear functional on $H$. The general formulation of Least Squares allows for a broad class of recovery problems. In our specific case, the function recovery of real–valued functions on a $d$–dimensional

(compact) subset $D$ using basis functions of a $k$–dimensional subspace $V_k := \operatorname{span}\{b_1, \dots, b_k\}$, we consider the specific form of Least Squares, given by

$$A_{n,k}(f) := \operatorname*{argmin}_{g \in V_k} \sum_{i=1}^{n} \frac{|g(x_i) - f(x_i)|^2}{\varrho_k(x_i)}$$

where

$$\varrho_k(x) = \frac{1}{2}\left(\frac{1}{k}\sum_{j<k} b_{j+1}(x)^2 + \frac{1}{\sum_{j\geq k} a_j^2}\sum_{j\geq k} a_j^2 b_{j+1}(x)^2\right)$$

and $x_1, \dots, x_n \in D$. Whenever $f \in V_k$, then, of course, $f = A_{n,k}(f)$. With

$$e\left(A_{n,k}, H\right) := \sup_{f \in B(H)} \left\|f - A_{n,k}(f)\right\|_{L_2}, \tag{7.1}$$

we denote the worst case error of $A_{n,k}$, where we measure the error of the reconstruction in the space $L_2 := L_2(D, \Sigma, \mu)$ of square integrable functions on $D$ with respect to the measure $\mu$, such that $H$ is embedded into $L_2$. In light of this, the $n$–th minimal error is denoted by

$$e_n(H) := \inf_{\substack{x_1,\dots,x_n \in D \\ \varphi_1,\dots,\varphi_n \in L_2}} \sup_{f \in B(H)} \left\|f - \sum_{i=1}^{n} f(x_i)\varphi_i\right\|_{L_2}$$

and can be understood as the worst case error of the optimal algorithm using $n$ function values. We get the clear inequality $e_n(H) \leq e(A_{n,k}, H)$ for any point set $\{x_1, \dots, x_n\}$. With

$$a_n(H) := \inf_{\substack{h_1^\star,\dots,h_n^\star \in H^\star \\ \varphi_1,\dots,\varphi_n \in L_2}} \sup_{f \in B(H)} \left\|f - \sum_{i=1}^{n} h_i^\star(f)\varphi_i\right\|_{L_2}$$

we denote the $n$-th approximation number, which is the worst-case error of an optimal algorithm that uses the $n$ best arbitrary linear and bounded functionals as information about the unknown. This quantity is equal to the $n$-th singular value of the embedding $\mathrm{id}\colon H \to L_2$.

The following is known since [11].

**Theorem 7.1** (Krieg–Ullrich)**.** *There exist constants $C, c > 0$ and a sequence of natural numbers $(k_n)$ with each $k_n \geq cn/\log(n+1)$ and for any $n \in \mathbb{N}$ and measure space $(D, \Sigma, \mu)$, and any RKHS $H$ of real-valued functions on $D$ embedded into $L_2(D, \Sigma, \mu)$, we have*

$$e_n(H) \leq \sqrt{\frac{C}{k_n}\sum_{j \geq k_n} a_j(H)^2}.$$

In particular, for

$$a_n(H) \lesssim n^{-s} \log^{\alpha+s}(n) \tag{7.2}$$

with $s > 1/2, \alpha \in \mathbb{R}$, this implies

$$e_n(H) \lesssim n^{-s} \log^{\alpha+s}(n).$$

The following follows from [15].

**Theorem 7.2** (Ullrich)**.** *Given $n \geq 2$ and $c > 0$, let*

$$k_n := \left\lfloor \frac{n}{2^8(2+c)\log n} \right\rfloor,$$

*then, for any measure space $(D, \Sigma, \mu)$ and any RKHS $H$ of real-valued functions on $D$, embedded into $L_2(D, \Sigma, \mu)$, it holds that*

$$e_n\left(A_{n,2k_n}, H\right) \leq \sqrt{\frac{2}{k_n} \sum_{j>k_n} a_j(H)^2}$$

*with probability at least $1 - 8n^{-c}$.*

**Examples.** In particular, (7.2) is satisfied for the approximation numbers on the Sobolev space of dominating mixed smoothness,

$$\begin{aligned} H &:= H^s_{\mathrm{mix}}\left(\mathbb{T}^d\right) \\ &:= \left\{ f \in L_2\left(\mathbb{T}^d\right) : \|f\|_H^2 := \sum_{m \in \mathbb{N}_0^d} \prod_{j=1}^{d} (1 + |m_j|^{2s}) \left\langle f, b_m \right\rangle_{L_2}^2 < \infty \right\} \end{aligned}$$

where $b_m := \otimes_{j=1}^d b^{(1)}_{m_j}$ and $m = (m_1, \ldots, m_d)$ with

$$\begin{aligned} b^{(1)}_{2m} &:= \sqrt{2}\cos(2\pi m x) \\ b^{(1)}_{2m-1} &:= \sqrt{2}\sin(2\pi m x) \end{aligned}$$

and $b^{(1)}_0 := 1$. This satisfies the assumption for $s > 1/2$. In particular, we can say

$$e_n\left(H^s_{\mathrm{mix}}\left(\mathbb{T}^d\right)\right) \lesssim n^{-s} \log^{sd}(n) \tag{7.3}$$

whenever $s > 1/2$. This disproves a previously posted conjecture (Conjecture 5.26) in [5] and shows that Smolyak's algorithm is not optimal in this case. The surprising fact is that, despite an optimal, deterministic construction of the point sets used for reconstruction being unknown, random i.i.d. points suffice for a reconstruction error that is on the order of optimal points, with probability tending to 1. We verify this by our experimental findings, presented in Section 8 with a much better relative number of points used for LSQ function recovery

vs. SA recovery than guaranteed in this section, i.e. better constants than explicitly known before. It remains an open problem to rigorously improve upon the constants in (7.3).

## 8. Experimental Findings

For assessing the performance of the Least Squares algorithms in comparison to the Sparse Grid alternative, we use the following 12 families of test functions from [14], each defined of the $d$-dimensional

unit-cube $[0,1]^d$.

1. Continuous: $$f_1(x) = \exp\left(-\sum_{i=1}^{d} c_i \left|x_i - w_i\right|\right)$$
2. Corner Peak: $$f_2(x) = \left(1 + \sum_{i=1}^{d} c_i x_i\right)^{-(d+1)}$$
3. Discontinuous: $$f_3(x) = \begin{cases} 0, & \text{if } x_1 > w_1 \text{ or } x_2 > w_2, \\ \exp\left(\sum_{i=1}^{d} c_i x_i\right), & \text{otherwise} \end{cases}$$
4. Gaussian: $$f_4(x) = \exp\left(-\sum_{i=1}^{d} c_i^2 (x_i - w_i)^2\right)$$
5. Oscillatory: $$f_5(x) = \cos\left(2\pi w_1 + \sum_{i=1}^{d} c_i x_i\right)$$
6. Product Peak: $$f_6(x) = \prod_{i=1}^{d} \left(c_i^{-2} + (x_i - w_i)^2\right)^{-1}$$
7. G-Function: $$f_7(x) = \prod_{i=1}^{d} \frac{|4x_i - 2 - w_i| + c_i}{1 + c_i}$$
8. Morokoff & Calfisch 1: $$f_8(x) = (1 + 1/d)^d \prod_{i=1}^{d} \left(c_i x_i + w_i\right)^{1/d}$$
9. Morokoff & Calfisch 2: $$f_9(x) = \frac{1}{(d - 0.5)^d} \prod_{i=1}^{d} \left(d - c_i x_i + w_i\right)$$
10. Roos & Arnold: $$f_{10}(x) = \prod_{i=1}^{d} \left|4c_i x_i - 2 - w_i\right|$$
11. Bratley: $$f_{11}(x) = \sum_{d=1}^{d} (-1)^i \prod_{j=1}^{d} \left(c_i x_i - w_i\right)$$
12. Zhou: $$f_{12}(x) = \frac{10^d}{2}\left[\varphi\left(x - \frac{1}{3}\right) + \varphi\left(x - \frac{2}{3}\right)\right]$$ $$\text{with } \varphi(x) = \frac{10}{(2\pi)^{d/2}} \exp\left(-\frac{1}{2}\left\|c\,(x - w)\right\|_2^2\right)$$

Note that the first 6 function classes are also known as the *Genz Integrand Families* and were introduced by Genz in [6, 7]. In comparison to the original definition in [14], we also introduce the parameters $c$ and $w$ in each class by making something like an affine-linear transformation

of the input $x$. This allows for testing multiple realizations of these functions.

Generating a function from a specific family is done by sampling the random vectors $c, w \in \mathbb{R}^d$. In our experiments, we sample each entry of $c$ and $w$ from a uniform distribution $\mathcal{U}(0,1)$ and rescale $c$ afterwards such that $\|c\|_1 = d$.

*Remark* 8.1. In [1], experiments were performed for the Genz families, defined on $[-1,1]^d$. We use $[0,1]^d$ as this ensures that also the other families are well-defined for any sampled $c, w \in \mathbb{R}^d$.

In the following experiments, we compare the Smolyak algorithm with two realizations of the *weighted* Least Squares algorithm. In the first realization we use random points that are uniformly distributed in $[0,1]^d$, and we don't reweight those point. In the second realization we sample the points from $(1-x^2)^{1/2}$, which is often referred as the *Chebyshev Density* or *Chebyshev Weight* and we use the value of this exact density at each point as the basis for the weight calculation. All 3 algorithms have the same basis functions, which are established from the application of the Smolyak algorithm. All Algorithms are tested for all families of functions with multiple $> 10$ realizations and for all dimensions $d \in [2:10]$. In each dimension, the fineness scale $q \in \mathbb{N}$ was varied. Note that $q > d$. Depending on $d$ smaller or larger values for $q$ were possible because of computational bottlenecks based on the exponential increase in the used points, see also [3] for a overview on the number of points in a sparse grid. Both Least Squares algorithms enjoyed twice the amount of points, compared to the Smolyak algorithm, sampled in their respective distributions.

- Least Squares Implementation: [8] using the *lstsq*-method
- Smolyak Implementation: [13] or [9] with [4]

For assessing the quality of the interpolants, we generated $n$ random points $x_1, \ldots, x_n$ distributed uniformly in $[0,1]^d$, where $n$ is the number of points in the Sparse Grid. Afterwards we compute the error quantities

$$e_{\ell_2}(A_j, q, f) := \left( \frac{1}{n} \sum_{i=1}^{n} \left( f(x_i) - \left(A_j(q,d)(f)\right)(x_i) \right)^2 \right)^{1/2}$$
$$e_{\ell_\infty}(A_j, q, f) := \max_{i \in [1:n]} \left| f(x_i) - \left(A_j(q,d)(f)\right)(x_i) \right|$$

for all three algorithms $A_j$ with $j = \{\text{Smolyak}, \text{LS-Uniform}, \text{LS-Chebyshev}\}$ and for all sampled functions $f$ in each family of functions. Those quantities are now depicted in Figures 1 to 3 as a function of $q$. Additionally,

**??** summarizes in combination with the different distributions for each function class Figures 4 to 6 the performance of each algorithm based on multiple realizations.

The results are also depicted in the following tables and figures.

***Reference to the formulas above, i.e. which formula is Least Squares? Section 5?***

***Write something about the setting in which we tested the functions. Maybe this can and should be combined with all the previous stuff. I.e. talk about the error metrics. Tell them which formulas were used for each algorithm. Some implementation details. Then refer to the figures and also to important tables. Tell them, what scale means (Probably need to adapt the notation of the images, i.e. use $q$ and not scale). Maybe adapt notation to $q$ or whichever character we used before for the description of Smolyak.***

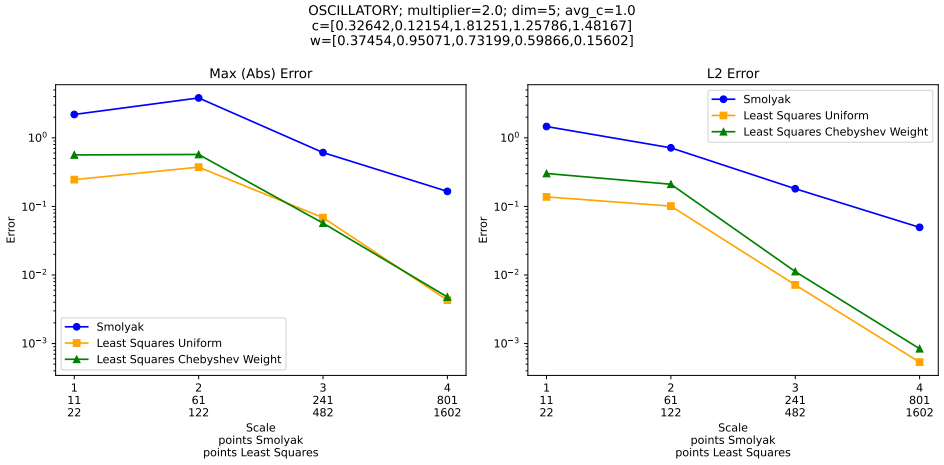


FIGURE 1. Caption

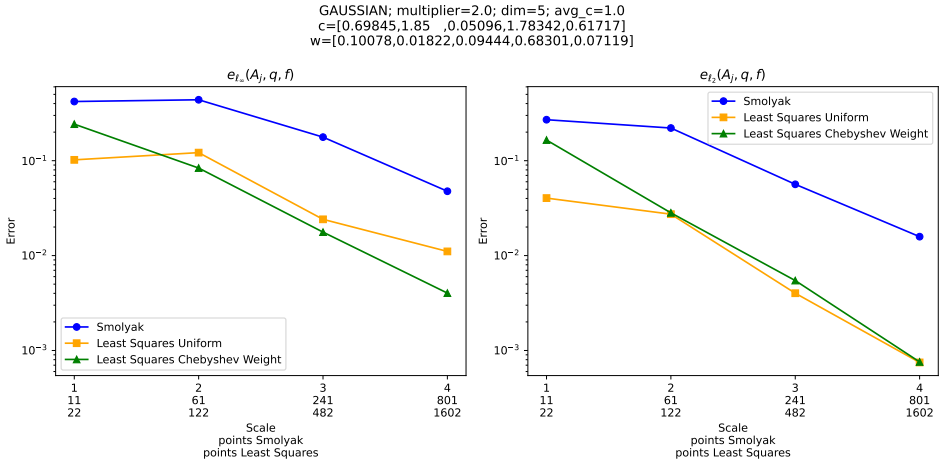


FIGURE 2. Caption

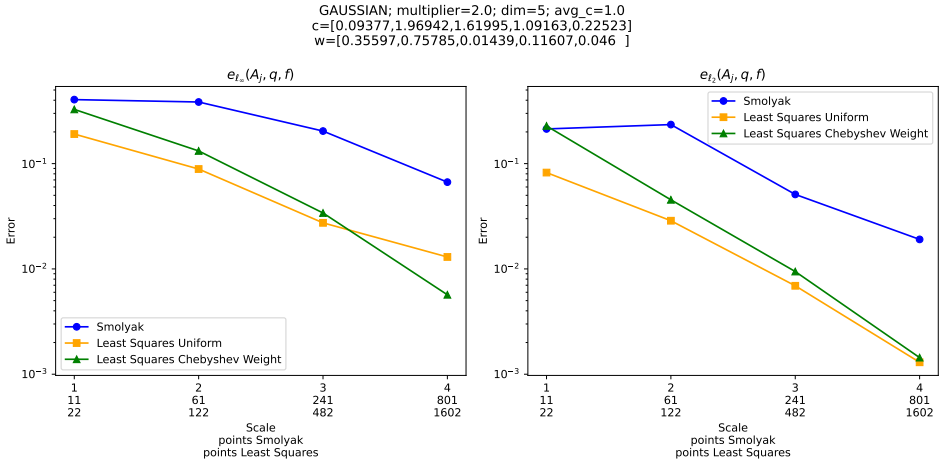


FIGURE 3. Caption

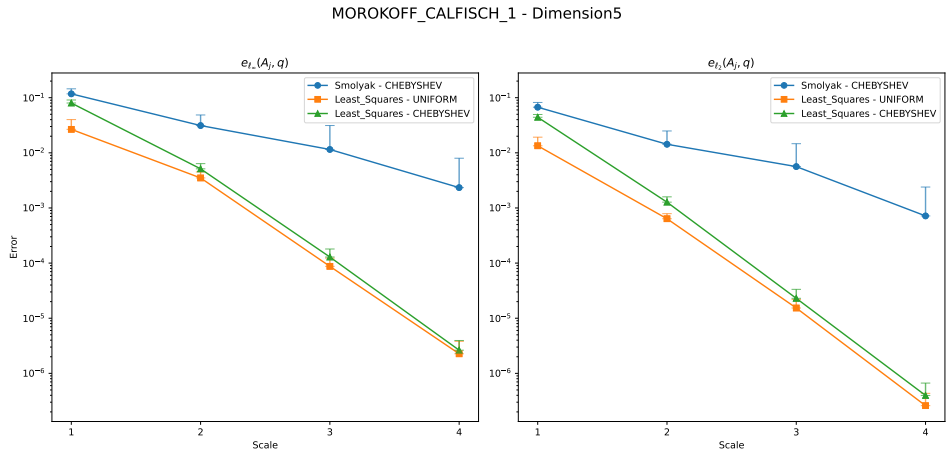


FIGURE 4. Caption

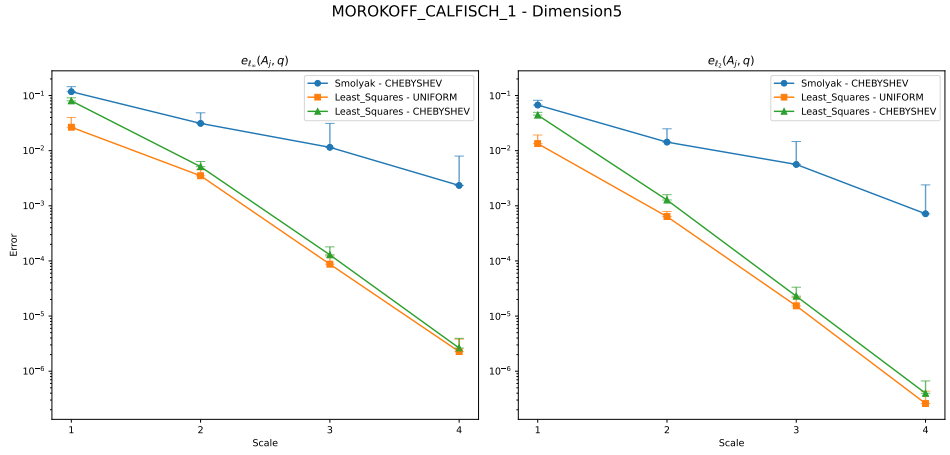


FIGURE 5. Caption

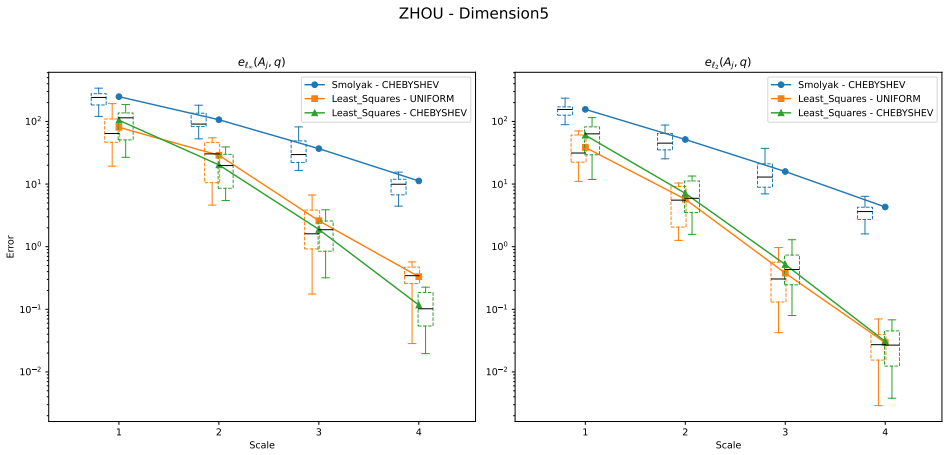


FIGURE 6. Caption

## 9. Conclusion

Ideas

- Same (or usually not a worse—mostly better) order of decay
- $2n$ points seem to suffice compared to the number of the paper
- In some cases, Least Squares outperforms Smolyak a lot
- Tasmainian: Sometimes bad performance: Bad implementation or maybe sometimes Smolyak really bad (Mario: Approxing the 0-Function). People might not be aware of the fact that the approximation quality might be extra-poor
-

## Acknowledgements

## References


[1] Volker Barthelmann, Erich Novak, and Klaus Ritter. "High dimensional polynomial interpolation on sparse grids". In: *Advances in Computational Mathematics* 12 (2000), pp. 273–288.

[2] Jean-Paul Berrut and Lloyd N Trefethen. "Barycentric lagrange interpolation". In: *SIAM review* 46.3 (2004), pp. 501–517.

[3] John Burkardt. "Counting Abscissas in Sparse Grids". In: (2014).

[4] Chase Coleman and Spencer Lyon. *Efficient Implementations of Smolyak's Algorithm for Function Approximation in Python and Julia*. `https://github.com/EconForge/Smolyak`. 2013.

[5] Dinh Dũng, Vladimir Temlyakov, and Tino Ullrich. *Hyperbolic Cross approximation*. Ed. by Sergey Tikhonov. Birkhäuser, 2018.

[6] Alan Genz. "A package for testing multiple integration subroutines". In: *Numerical integration: Recent developments, software and applications* (1987), pp. 337–340.

[7] Alan Genz. "Testing multidimensional integration routines". In: *Proc. of international conference on Tools, methods and languages for scientific and engineering computation*. 1984, pp. 81–94.

[8] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. URL: `https://doi.org/10.1038/s41586-020-2649-2`.

[9] Kenneth L Judd et al. "Smolyak method for solving dynamic economic models: Lagrange interpolation, anisotropic grid and adaptive domain". In: *Journal of Economic Dynamics and Control* 44 (2014), pp. 92–123.

[10] ANDREAS KLIMKE, KAI WILLNER, and BARBARA WOHLMUTH. "UNCERTAINTY MODELING USING FUZZY ARITHMETIC BASED ON SPARSE GRIDS: APPLICATIONS TO DYNAMIC SYSTEMS". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 12.06 (2004), pp. 745–759.

[11] David Krieg and Mario Ullrich. "Function Values Are Enough for $L_2$-Approximation". In: *Foundations of Computational Mathematics* 21.4 (Dec. 2020), pp. 1141–1151. URL: `http://dx.doi.org/10.1007/s10208-020-09481-w`.

[12] Joseph-Louis Lagrange. "Lectures on Elementary Mathematics. On the employment of curves in the solution of problems". Trans. by Thomas J. McCormack. In: (1901). Originally published in French in 1795.

[13] Miroslav Stoyanov et al. *Tasmanian*. Sept. 2013. URL: `https://github.com/ORNL/Tasmanian`.

[14] S. Surjanovic and D. Bingham. *Virtual Library of Simulation Experiments: Test Functions and Datasets*. Retrieved March 29, 2025, from https://www.sfu.ca/~ssurjano/integration.html.

[15] Mario Ullrich. "On the worst-case error of least squares algorithms for $L_2$-approximation with high probability". In: *Journal of Complexity* 60 (Oct. 2020), p. 101484. URL: http://dx.doi.org/10.1016/j.jco.2020.101484.

[16] Edward Waring. "Problems concerning Interpolations. By Edward Waring, M. D. F. R. S. and of the Institute of Bononia, Lucasian Professor of Mathematics in the University of Cambridge". In: *Philosophical Transactions of the Royal Society of London* 69 (1779), pp. 59–67. URL: http://www.jstor.org/stable/106408 (visited on 04/11/2025).

J.E., Johannes Kepler University Linz; jakob.eggl@jku.at;
E.M., Johannes Kepler University Linz; elias.mindlberger@jku.at;
M.U., Johannes Kepler University Linz; mario.ullrich@jku.at