

Approximating optimal point configurations for multivariate polynomial interpolation

Marc Van Barel

Matthias Humet

Laurent Sorber

Report TW 637, October 2013



KU Leuven

Department of Computer Science

Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

Approximating optimal point configurations for multivariate polynomial interpolation

Marc Van Barel

Matthias Humet

Laurent Sorber

Report TW 637, October 2013

Department of Computer Science, KU Leuven

Abstract

Efficient and effective algorithms are designed to compute the coordinates of nearly optimal points for multivariate polynomial interpolation on a general geometry. “Nearly optimal” refers to the property that the set of points has a Lebesgue constant near to the minimal Lebesgue constant with respect to multivariate polynomial interpolation on a finite region. The proposed algorithms range from cheap ones that produce point configurations with a reasonably low Lebesgue constant, to more expensive ones that can find point configurations for several two-dimensional shapes which have the lowest Lebesgue constant in comparison to currently known results.

Keywords : multivariate polynomial interpolation, Lebesgue constant, mini-max approximation, greedy algorithms.

MSC : Primary : 65D05, Secondary : 41A10, 41A50, 41A63.

Approximating optimal point configurations for multivariate polynomial interpolation

Marc Van Barel Matthias Humet Laurent Sorber *

October 15, 2013

Dedicated to Lothar Reichel on the occasion of his sixtieth birthday.

Abstract

Efficient and effective algorithms are designed to compute the coordinates of nearly optimal points for multivariate polynomial interpolation on a general geometry. “Nearly optimal” refers to the property that the set of points has a Lebesgue constant near to the minimal Lebesgue constant with respect to multivariate polynomial interpolation on a finite region. The proposed algorithms range from cheap ones that produce point configurations with a reasonably low Lebesgue constant, to more expensive ones that can find point configurations for several two-dimensional shapes which have the lowest Lebesgue constant in comparison to currently known results.

1 Introduction

In several theoretical as well as computational mathematical problems, one wants to work with complicated multivariate functions. However, in a lot of cases performing operations with these original functions is cumbersome and requires an unacceptably high computational effort. A solution to this problem is to replace the original complicated function by a function that can be handled much more easily, e.g., polynomial functions. Within this

*The research was partially supported by the Research Council K.U.Leuven, project OT/10/038 (Multi-parameter model order reduction and its applications), PF/10/002 Optimization in Engineering Centre (OPTEC), and by the Interuniversity Attraction Poles Programme, initiated by the Belgian State, Science Policy Office, Belgian Network DYSCO (Dynamical Systems, Control, and Optimization). Laurent Sorber is supported by a Flanders Institute of Science and Technology (IWT) doctoral scholarship. The scientific responsibility rests with its author(s).

space of simpler functions, we can look for the function that optimizes one of several possible criteria. One example is the minmax criterion, but the computational effort to find the function that minimizes the infinity norm error, is large. Instead an approximant can be found that is almost as good as the minmax approximant by interpolating the original function in certain well-chosen points. These points are chosen in an optimal or nearly optimal way with respect to minimizing the Lebesgue constant.

In this manuscript we develop several algorithms to compute point configurations for multivariate polynomial interpolation that have a low or even almost minimal Lebesgue constant for a given geometry. Interpolating in these points will yield good polynomial approximants for the geometry, compared to the minmax polynomial approximant.

For the problem of approximating univariate functions by polynomials in a typical compact set on the real line, i.e., an interval, both the theory and the corresponding software are well-developed. We refer to Chebfun, a MATLAB toolbox, whose theoretical foundation and several of its applications are described in the book by Trefethen [8]. If one transforms an arbitrary compact interval to the interval $[-1, 1]$, it turns out that different types of Chebyshev points not only form nearly optimal point configurations, but that the computation of the corresponding interpolant can be performed very efficiently (and accurately) by using the Fast Fourier Transform (FFT). The zero sets of other orthogonal polynomials, e.g., Legendre polynomials, have similar approximating properties but they can not be represented explicitly and the corresponding approximant cannot be computed equally efficient. For univariate rational interpolation, the so-called rational Chebyshev points are nearly optimal on the interval $[-1, 1]$ (see [10]).

The problem setting is more complicated in the multivariate case, because the geometry can take on more general forms (e.g., a polygon, a disk, ...), in contrast to the univariate case where the typical geometry is the interval. Moreover the degree structure of the polynomial functions is more general. For a theoretical overview, we refer the interested reader to [1].

One of the criteria to determine the location of good points for polynomial approximation in a geometry, is minimizing the Lebesgue constant, which is the maximum of the Lebesgue function.¹ Points in some geometry are considered to be nearly optimal if the Lebesgue constant with respect to that geometry is small, and they are optimal if the Lebesgue constant is as small as possible. The Padua points seem to be the first known example of nearly optimal points for total degree polynomial interpolation in two variables, with a Lebesgue constant increasing like log square of the

¹The corresponding definitions are given in Section 2.

degree. The corresponding geometry is a square or a rectangle (or another derived form). These Padua points have been discovered and extensively studied by the Padova-Verona research group on “Constructive Approximation and Applications” (CAA-group) and their collaborators (<http://www.math.unipd.it/~marcov/CAA.html>).

For other geometries there are no explicit representations known for (nearly) optimal points with respect to minimizing the Lebesgue constant. The CAA-group has developed MATLAB software to compute such nearly optimal points for several geometries, e.g., the disk and the simplex, not only for minimizing the Lebesgue constant but also for maximizing the corresponding Vandermonde determinant (Fekete-points) [3]. Initializing the software with reasonably nearly optimal points, it can also be used to derive point sets with a smaller Lebesgue constant than the initial set. A disadvantage of the software is that it is rather slow and that it is limited to a relatively small number of points because of the ill-conditioning of the corresponding Vandermonde matrices.

On March 4, 2013, an extension of Chebfun was made available to work with functions in two variables defined on a rectangle (<http://www2.maths.ox.ac.uk/chebfun/chebfun2/>). In our approach we work with functions on more general geometries with corresponding nearly optimal point configurations and numerically sound representations for the multivariate polynomials approximants.

In this manuscript, we represent the polynomial functions using orthogonal bases with respect to a discrete inner product where the mass points are lying within the considered geometry. This leads to small condition numbers for the generalized Vandermonde matrices involved in the computations that allow us to find nearly optimal point configurations that are much larger compared to the point configurations obtained by currently known techniques.

Instead of solving the minmax problem (10), the algorithms of this manuscript tackle different, but related, optimization problems that approximately solve the same problem. Although the optima of these related problems do not coincide with the optima of the original minmax problem, they can be solved much more efficiently, making it possible to minimize the Lebesgue constant much more effectively.

The manuscript is divided into the following sections. In Section 2 the definition of the Lebesgue function and the Lebesgue constant is given. In Section 3, it is explained how a good approximation of the Lebesgue constant can be computed in an efficient way. Section 4 describes the representation that will be used for the multivariate polynomials given a certain geometry. Section 5 gives several algorithms to compute nearly optimal point configurations, ranging from cheap ones that produce non-optimal point configurations

with a reasonably low Lebesgue constant, to more expensive ones that can find point configurations with an almost optimal Lebesgue constant. In Section 6 we show the results of applying these algorithms on several geometries for different degrees.

2 Lebesgue constant

Let Ω be a compact subset of \mathbb{R}^n . Consider the space \mathcal{P}_δ^n of polynomials in n variables having total degree $\leq \delta$.² This space has dimension N with

$$N = \binom{\delta + n}{n}. \quad (1)$$

Consider a set $X = \{\mathbf{x}_k\}_1^N$ of N points in Ω and a basis $\{\phi_k\}_1^N$ for \mathcal{P}_δ^n . Let $V_X = [\phi_j(\mathbf{x}_i)]_{i,j}$ denote the generalized Vandermonde matrix for this basis in the points X . Given a function $f \in C(\Omega)$, we can approximate this function by computing the multivariate polynomial interpolant $p \in \mathcal{P}_\delta^n$ in the set of points X . Note that this interpolant is well defined and unique iff the generalized Vandermonde matrix V_X is nonsingular. If that is the case, the set of points X is called unisolvent for the space \mathcal{P}_δ^n .

Definition 1 (Lebesgue function and Lebesgue constant) *Given a compact set $\Omega \subset \mathbb{R}^n$ and a set of points $X = \{\mathbf{x}_k\}_1^N \subset \Omega$ that is unisolvent for \mathcal{P}_δ^n . The Lebesgue function $\lambda_X(\mathbf{y})$ is defined as*

$$\lambda_X(\mathbf{y}) = \sum_{i=1}^N |l_i(\mathbf{y})|$$

with $l_i(\mathbf{y})$ the i th Lagrange polynomial, i.e.,

$$\begin{cases} l_i \in \mathcal{P}_\delta^n \\ l_i(\mathbf{x}_j) = \delta_{i,j}, \end{cases} \quad \text{for } i, j = 1, 2, \dots, N.$$

The Lebesgue constant Λ_X is defined as the maximum of the Lebesgue function $\lambda_X(\mathbf{y})$ for $\mathbf{y} \in \Omega$, i.e.,

$$\Lambda_X = \max_{\mathbf{y} \in \Omega} \lambda_X(\mathbf{y}).$$

² More general subsets of polynomials can be considered, i.e., having another degree structure in comparison to the total degree.

The Lebesgue constant is a measure to compare the polynomial interpolant with the best polynomial approximant in the uniform norm. More precisely, for any function $f \in C(\Omega)$, let p denote the polynomial interpolant and p^* the best polynomial approximant in uniform norm, then

$$\|f - p\|_\infty \leq (1 + \Lambda_X) \|f - p^*\|_\infty.$$

Hence, when the Lebesgue constant Λ_X is small, we can find an approximation of a function f that is almost as good as the best polynomial approximation p^* , by just computing the polynomial interpolant p , which is generally much easier to compute than p^* .

The magnitude of the Lebesgue constant Λ_X depends heavily on the configuration of the points X in the compact subset Ω . Before we look for different algorithms to find point configurations with a low Lebesgue constant, the next section investigates how we can efficiently approximate the Lebesgue constant Λ_X .

3 Approximating the Lebesgue constant Λ_X

Computing the Lebesgue constant for a region $\Omega \subset \mathbb{R}^n$ is not an easy problem. Following the same approach as in [3], we approximate the Lebesgue constant by taking the maximum over a finite set $Y \subset \Omega$ of K well-chosen points

$$\Lambda_X \approx \max_{\mathbf{y} \in Y} \sum_{i=1}^N |l_i(\mathbf{y})|. \quad (2)$$

Instead of working with admissible meshes [3, 2], we use DistMesh [7] to generate adequate discretization for many possible geometries. This package turns out to be very suitable for the point sets we need, following the intuitive notion that the set Y should be more dense near boundaries of the geometry Ω .³ The main advantage over using admissible meshes, is the fact that it is easy to work with very general geometries. In Section 6, we show the efficiency and effectiveness of this approach and give more details on the values of the parameters used in the numerical experiments. To give an idea of the meshes generated by DistMesh, Figure 1 shows a mesh for the L-shape consisting of 3475 points.

³We believe that this is true for convex geometries, but not for the “non-convex” part of a boundary, e.g., the non-convex part of the boundary of the L-shape. In Figure 8 we show a nearly optimal point configuration on the L-shape, that exhibits a low density of points near the non-convex part of the geometry.

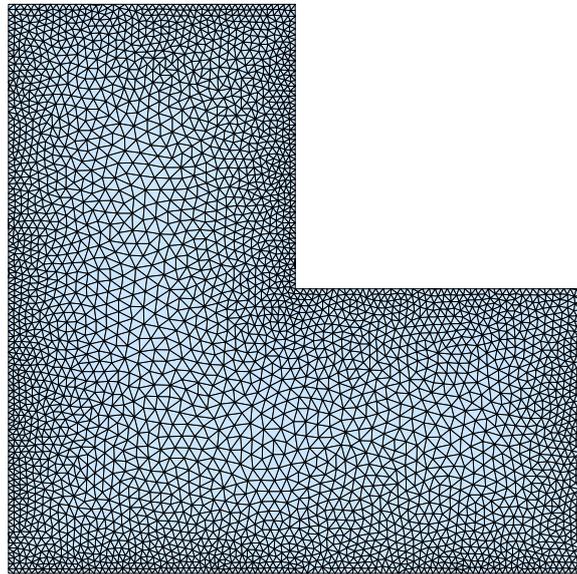


Figure 1: Example of a mesh generated by DistMesh for the L-shape consisting of 3475 points.

To compute the approximation (2), we choose a basis $\{\phi_k\}_1^N$ in \mathcal{P}_δ^n . More details on the choice of this basis will be given in Section 4. From the definition of Lagrange polynomials, we have the following expression for the basis polynomials:

$$[\phi_1(\mathbf{y}) \quad \cdots \quad \phi_N(\mathbf{y})] = [l_1(\mathbf{y}) \quad \cdots \quad l_N(\mathbf{y})] \begin{bmatrix} \phi_1(\mathbf{x}_1) & \cdots & \phi_N(\mathbf{x}_1) \\ \vdots & & \vdots \\ \phi_1(\mathbf{x}_N) & \cdots & \phi_N(\mathbf{x}_N) \end{bmatrix},$$

or evaluated in each of the K points $\mathbf{y}_j \in Y$:

$$\begin{bmatrix} \phi_1(\mathbf{y}_1) & \cdots & \phi_N(\mathbf{y}_1) \\ \vdots & & \vdots \\ \phi_1(\mathbf{y}_K) & \cdots & \phi_N(\mathbf{y}_K) \end{bmatrix} = \begin{bmatrix} l_1(\mathbf{y}_1) & \cdots & l_N(\mathbf{y}_1) \\ \vdots & & \vdots \\ l_1(\mathbf{y}_K) & \cdots & l_N(\mathbf{y}_K) \end{bmatrix} \begin{bmatrix} \phi_1(\mathbf{x}_1) & \cdots & \phi_N(\mathbf{x}_1) \\ \vdots & & \vdots \\ \phi_1(\mathbf{x}_N) & \cdots & \phi_N(\mathbf{x}_N) \end{bmatrix}.$$

We write this in a concise way as

$$V_Y = L V_X. \tag{3}$$

Note that K is chosen such that $K \gg N$.

The matrices V_X and V_Y are the basis polynomials evaluated in the points of the sets X and Y and V_X is the generalized Vandermonde matrix of the previous section. If the point set X is unisolvent, the matrix L of Lagrange polynomials can be computed by solving a system of linear equations with coefficient matrix V_X . Taking its matrix infinity norm results in approximation (2) of the Lebesgue constant, i.e.,

$$\Lambda_X \approx \|L\|_\infty = \|V_Y V_X^{-1}\|_\infty.$$

The accuracy of the computation of $\|L\|_\infty$ depends on the condition number of the generalized Vandermonde matrix V_X . For this number to be small, it is important to obtain a good basis $\{\phi_k\}_1^N$ for the geometry Ω considered, which we discuss in more detail in the next section.

4 Obtaining a good basis for a specific geometry

In this section we discuss some of the possible choices for the basis of \mathcal{P}_δ^n that are used to compute the Lebesgue constant Λ_X . First we mention the bases that have been used in [3] to obtain point configurations with a low Lebesgue constant for the square, the simplex and the disk. Then we discuss

orthonormal bases with respect to a discrete inner product, which can be computed by solving an inverse eigenvalue problem [9]. We briefly describe the problem setting and mention some of the approaches to solve the inverse eigenvalue problem. Finally we introduce a technique to extend a basis, which will be used in Section 5.1.

Since the choice of the basis determines the Vandermonde matrix V_X of the system (3), it has a large impact on the conditioning of the problem of computing Λ_X . The idea we pursue in this paper is to use a basis for which the condition number of V_X is small enough. The precise meaning of “small enough” depends on how accurate the computed value of Λ_X needs to be. For example, for the algorithms of Section 5, in practice it suffices to know only a couple of correct significant decimal digits of the matrix L in (3), so that $\text{cond}(V_X)$ may be as large as 10^{12} .

Briani et al. [3] use three different orthonormal bases for the respective geometries considered. Let $\Omega \in \mathbb{R}^n$ be a compact set, then we say that two polynomials $p, q \in \mathcal{P}_\delta^n$ are orthogonal with respect to Ω and the weight function $w(\mathbf{x})$ if

$$\langle p, q \rangle_\Omega := \int_\Omega p(\mathbf{x})q(\mathbf{x})w(\mathbf{x})d\mathbf{x} = 0. \quad (4)$$

The three bases consist of product Chebyshev polynomials for the square, Dubiner polynomials for the simplex and Koornwinder type II polynomials for the disk. These polynomials are orthonormal with respect to the respective geometries and the respective weight functions $w(\mathbf{x}) = \prod_{i=1}^n (1 - x_i)^{-\frac{1}{2}}$, $w(\mathbf{x}) = 1$ and $w(\mathbf{x}) = 1$.

Our approach is to consider a discrete inner product

$$\langle p, q \rangle_X = \sum_{i=1}^N w_i^2 p(\mathbf{x}_i)q(\mathbf{x}_i), \quad (5)$$

with points $X := \{\mathbf{x}_i\}_1^N \subset \mathbb{R}^n$ and weights $w_i \in \mathbb{R}^+$. An advantage of using an orthonormal basis $\{\phi_k\}_1^N$ with respect to this inner product is that, for $w_i = 1$, the matrix V_X is orthogonal. Hence, numerical difficulties to compute Λ_X for a set of points X can be avoided by taking an orthonormal basis with respect to (5) defined on the same point set X .

The problem of computing orthogonal multivariate polynomials with respect to (5) has been studied in [9]. In this work the orthogonal polynomials are represented by the recurrence coefficients $h_{i,j}^{(k)}$ of the recurrence relation

$$x_k \phi_j = \sum_{i=1}^{\pi_j^{(k)}} h_{i,j}^{(k)} \phi_i, \quad (6)$$

which gives an expression for $\phi_{\pi_j^{(k)}}$ if the previous polynomials $\phi_1, \dots, \phi_{\pi_j^{(k)}-1}$ are known. The index $\pi_j^{(k)}$ depends on j and k and will be discussed later. The polynomials have to be ordered along a term order, meaning that $\phi_k(\mathbf{x}) = a_k \mathbf{x}^{\alpha_k} + \dots + a_1 \mathbf{x}^{\alpha_1}$ and the monomials $\mathbf{x}^{\alpha_k} := x_1^{\alpha_{k,1}} \dots x_n^{\alpha_{k,n}}$ satisfy a term order: $1 \prec \mathbf{x}^\beta$ for all $\beta \neq \mathbf{0}$ and if $\mathbf{x}^{\alpha_i} \prec \mathbf{x}^{\alpha_j}$, then $\mathbf{x}^\beta \mathbf{x}^{\alpha_i} \prec \mathbf{x}^\beta \mathbf{x}^{\alpha_j}$ for all $\beta \neq \mathbf{0}$. Here, we will restrict ourselves to graded term orders, imposing the additional condition that, if $\sum_k \alpha_{i,k} =: |\alpha_i| < |\alpha_j|$, then $\mathbf{x}^{\alpha_i} \prec \mathbf{x}^{\alpha_j}$. An example of a graded term order is the *graded lexicographical order*, which for $n = 3$ looks like

$$1 \prec z \prec y \prec x \prec z^2 \prec yz \prec y^2 \prec xz \prec xy \prec x^2 \prec \dots$$

A matrix expression for (6) is

$$x_k [\phi_1 \ \phi_2 \ \dots \ \phi_{\hat{N}}] = [\phi_1 \ \phi_2 \ \dots \ \phi_N] \hat{H}_k \quad (7)$$

with $\hat{H}_k(i, j) = h_{i,j}^{(k)}$ and $\hat{H}_k \in \mathbb{R}^{N \times \hat{N}}$. Here N and \hat{N} are the dimensions of the spaces \mathcal{P}_δ^n and $\mathcal{P}_{\delta-1}^n$, respectively. The element $h_{\pi_j^{(k)}, j}^{(k)}$ associated with the leading basis polynomial in (6) is called a pivot element of \hat{H}_k and it is the last nonzero element in the j -th column. The positions $(\pi_j^{(k)}, j)$ of the pivot elements follow from the monomial order and can be determined at a negligible cost. E.g., for the graded lexicographical ordering and $n = 3$, the matrix \hat{H}_x has pivots at positions

$$(4, 1), (8, 2), (9, 3), (10, 4), (15, 5), (16, 5), \dots$$

If $\mathbf{w} = [w_1 \ \dots \ w_N]^T$ is a vector with the weights and $X_k = \text{diag}(x_{1,k}, \dots, x_{N,k})$ is the diagonal matrix with the k -th coordinates of the points $\mathbf{x}_i \in X$, then the recurrence matrices \hat{H}_k can be found from the inverse eigenvalue problem

$$Q^T Q = I, \quad Q^T \mathbf{w} = \|\mathbf{w}\|_2 e_1, \quad \text{and} \quad H_k = Q^T X_k Q, \quad k = 1, \dots, n, \quad (8)$$

where the matrices \hat{H}_k are embedded in the $H_k \in \mathbb{R}^{N \times N}$ as follows

$$H_k = [\hat{H}_k \quad \times]. \quad (9)$$

The basic idea is to apply orthogonal transformations to \mathbf{w} and X_k to make zeros in \mathbf{w} while at the same time assuring that the matrices H_k have the correct pivot element structure, which is determined by the monomial order. If the pivot elements in the matrices H_k are positive, then the process has a unique outcome.

We have implemented two methods to solve (8), where the user can supply any graded term order. The first method adds one points at a time. In each step, it uses Givens transformations to make one weight in \mathbf{w} zero and to bring the matrices H_k to the desired structure. The algorithm is explained in [9] for the bivariate case. The second method uses Householder transformations. A first Householder is applied to \mathbf{w} to make all the zeros at once. Subsequent Householders then bring H_k to the desired structure.

Although the method with Householder transformations has a higher flop-count than the method with Givens transformations, it becomes faster for large problems, because the operations are less granular. By using more matrix vector products instead of fine grain operations on vectors, most of the work is done using BLAS-2 routines (see [4, Chapter 1]). We will therefore prefer the second method for large problems, but the first method remains useful, because it allows to add points to an existing inner product.

As noted in [9], there is some freedom in the algorithms concerning which pivot is used to construct the Givens or Householder transformation. Several criteria to choose the pivot have been implemented, so the reader can experiment with them. We have adopted the approach to construct the orthogonal transformation from the vector with the highest 2-norm, since this seemed the most accurate in numerical tests. Numerical tests also pointed out to use a similar approach to evaluate the orthonormal polynomials using the recurrence relations (6): if $l = \pi_{j_1}^{(k_1)} = \dots = \pi_{j_m}^{(k_m)}$, so there are m pivot elements in the l -th row of respective matrices H_{k_i} , then ϕ_l is computed from (6) for that k_i associated with the biggest pivot $h_{l,j_i}^{(k_i)}$.⁴

The last part of this section is devoted to explain a simple technique that extends a basis. In Section 5.1, we motivate this technique and give some numerical results that show its use. Suppose we have a basis $\{\phi_k\}_1^N$ for \mathcal{P}_δ^n associated with a graded term order, which is a good representation on a certain domain $\Omega \in \mathbb{R}^n$. We extend this basis with polynomials $\phi_{N+1}, \phi_{N+2}, \dots, \phi_{N+m}$ by taking products of the original basis

$$\phi_i = \phi_{k_i} \cdot \phi_{l_i}, \quad i = N + 1, \dots, N + m,$$

where the indices k_i and l_i satisfy

- (i) $\alpha_i = \alpha_{k_i} + \alpha_{l_i}$,
- (ii) $|\alpha_{l_i}| = |\alpha_{N+1}| - 1$,
- (iii) k_i is as low as possible.

⁴Note that choosing the biggest pivot is similar to the optimal pivoting strategy for Gaussian elimination.

Condition (i) follows directly from the definition of the monomial order and condition (ii) implies that we take the total degree of one of the factors to be one less than the total degree of the first polynomial that extends the basis. From (i) and (ii), the total degree of ϕ_{k_i} is fixed, and condition (iii) then determines the values of k_i and l_i .

Such an extension of a good basis on a domain will usually be less good than the original basis, and it is clear that it will deteriorate as m grows larger. However, the main advantage is that it can be evaluated very cheaply in points where the original basis has been evaluated. In Section 5.1 it is explained how this technique can be used to decrease computation time, while at the same time maintaining a high enough level of robustness.

5 Computing nearly optimal interpolation points

As explained in Section 2, we get a good polynomial approximation of the minmax polynomial approximant by interpolation in points X with a small Lebesgue constant Λ_X . To obtain such a set X , we want to solve the following minmax optimization problem

$$\min_{X \subset \Omega} \Lambda_X = \min_{X \subset \Omega} \max_{\mathbf{y} \in \Omega} \lambda_X(\mathbf{y}). \quad (10)$$

If we approximate the Lebesgue constant as in Section 3 by $\Lambda_X \approx \|L\|_\infty$, we get the optimization problem

$$\begin{aligned} \min_{X \subset \Omega} \|L\|_\infty \\ \text{subject to } V_Y = L V_X, \end{aligned} \quad (11)$$

where $X = \{\mathbf{x}_i\}_1^N$ and $Y = \{\mathbf{y}_i\}_1^K$.

This is a minmax optimization problem with constraints because the points \mathbf{x}_i have to lie in the region Ω . Minmax optimization problems are notoriously difficult to solve. In addition the objective function Λ_X is not everywhere differentiable, and the number of variables grows fast when increasing the degree δ and/or the number of dimensions n . E.g., for $n = 2$ and $\delta = 20$, the dimension N of the vector space \mathcal{P}_δ^n is 231. Hence, the number of real variables is the number of components of the N points \mathbf{x}_k , i.e., 462.

In [3], Briani et al. describe a collection of MATLAB scripts to solve the optimization problem (11) using the MATLAB Optimization Toolbox. They consider $n = 2$ and Ω equal to the square, the disk and the simplex, and their results include nearly optimal point configurations for these geometries up to a total degree of $\delta = 20$. There is no certainty that the real optimum

is reached, but the Lebesgue constants found are the smallest at the point of their writing.

In the next subsections, we present alternative methods to find a point set X with a low Lebesgue constant. These methods work for very general geometries, can be used for larger point sets and are faster compared to current techniques. The first algorithm uses a relaxed optimization criterion and creates a point configuration with a relatively low Lebesgue constant in an efficient, non-iterative way. The second algorithm iterates over the point set one point at a time, using the same criterion. The third and fourth algorithm are more advanced optimization algorithms that solve a similar but easier problem than (11) leading to point configurations with a nearly optimal Lebesgue constant.

5.1 Greedy algorithm by adding points

Evaluating the objective function $\|L\|_\infty$ of the optimization algorithm (11) requires the evaluation of the basis in the points X and the solution of a system of linear equations. Since the objective function is not differentiable on Ω and the number of variables can become very high (e.g., 462 for $n = 2$ and $\delta = 20$), the convergence to a local minimum using standard MATLAB Optimization tools can take a lot of iterations, and consequently a lot of objective function evaluations.

In this section, we develop a “greedy” algorithm to generate a point configuration for any geometry Ω with a reasonably low Lebesgue constant Λ_X , with only a small computational effort. The algorithm is based on two ideas:

1. In each step, one point of the region Ω is added, while the other points remain where they are.
2. This point is added there where the Lebesgue function reaches its maximum.

We will refer to the algorithm as the Greedy Add algorithm.

Criterion 2 is reasonable in the sense that it guarantees that the updated Lebesgue constant has the value 1 in the new point. This point can be approximated by taking it from the set $Y \subset \Omega$, where the Lebesgue function reaches a maximum. Note that the new point could also be chosen to minimize the Lebesgue constant as a function of only one point, but this would be much more costly. Instead, we use a greedy approach where the next point is picked based on the mentioned relaxed criterion. Numerical experiments will show that, although the point configurations obtained are clearly not

optimal, they exhibit a structure in the domain Ω similar to (nearly) optimal configurations, and their Lebesgue constant is reasonably low.

Obviously, the first point can be chosen freely, and since the Lebesgue function for one point is a constant, the same holds for the second point. Although this choice can influence the quality of the obtained point configuration, i.e., the value of the Lebesgue constant, in practice we have noticed that choosing both points randomly in Ω works sufficiently well.

Theoretically, the method can fail. It is possible that at some step, after adding the next point, the configuration is not unisolvent anymore, meaning that the Vandermonde matrix is singular. As a result, the Lebesgue constant reaches infinity, leaving the next point undefined. When close to singular, numerical problems are to be expected in the evaluation of the Lebesgue function, giving an inaccurate computation of the next point. In practice however, we have only encountered such situations if the first two points were not random. If they are chosen randomly, we believe that the probability for such an event to occur is zero, although we do not have any proof.

Suppose that we want to generate a configuration of N points, where N is the dimension (1) of the space \mathcal{P}_δ^n .⁵ For now assume that there is a suitable basis for \mathcal{P}_δ^n on the geometry Ω , e.g., product Chebyshev polynomials on the square $[-1, 1]^2$. If that is the case, the Greedy Add algorithm can be formulated as Algorithm 1. Since the grid Y consists of K points, the matrix V_Y is of dimension $K \times N$. Furthermore, $V_Y^{(k)}$ is the $K \times k$ matrix with the first k columns of V_Y and $V_X^{(k)}$ is the $k \times k$ (generalized) Vandermonde matrix for the first k basis polynomials and the points in X . In step k , the matrix L is $K \times (k - 1)$ and each column contains one of the Lagrange polynomials for the points in X evaluated in Y . The index i selects the point in Y where the Lebesgue function is maximal.

Two remarks have to be made. First, the computation of L can be accelerated using the Sherman-Morrison-Woodbury formula ([4, p. 50]). Indeed, in step k the matrices $V_X^{(k-1)}$ and $V_Y^{(k-1)}$ are the same as in the previous step, except for their last columns and the last row of $V_X^{(k-1)}$. The matrix of the system is therefore a rank-2 update of the system in the previous step. Making use of this fact improves the efficiency of one step from $O(Kk^2)$ flop to $O(Kk)$. There should be a $O(k^3)$ term as well, but we get rid of it by updating the QR factorization of $V_X^{(k-1)}$ ([4, Section 12.5]).

Second, given a geometry Ω , it is not always apparent which basis to use, if the Vandermonde matrices in the algorithm have to remain well conditioned.

⁵Note that all the algorithms work for any value of N , but for notational convenience we work with spaces of total degree.

Algorithm 1 Greedy Add algorithm

Input: N, Y , basis**Output:** X
$$X \leftarrow \{2 \text{ random points } \mathbf{x}_1 \text{ and } \mathbf{x}_2\}$$
$$V_Y \leftarrow \text{evaluate basis functions in grid } Y \in \Omega$$
for $k = 3, \dots, N$ **do**
$$V_X^{(k-1)} \leftarrow \text{evaluate basis functions in } X$$
$$L \leftarrow V_Y^{(k-1)} = LV_X^{(k-1)}$$
$$i \leftarrow \text{index of row of } L \text{ with largest one norm}$$
$$\mathbf{x}_k \leftarrow Y(i)$$
$$X \leftarrow X \cup \{\mathbf{x}_k\}$$
end for

As an example, we carry out Algorithm 1 on the L-shape using product Chebyshev polynomials for a degree $\delta = 30$ or $N = 496$, and we plot the condition number of $V_X^{(k)}$ in Figure 2. The condition number keeps growing steadily until at some point it becomes so large that the Lebesgue function evaluations possibly have no correct significant digits left.

A solution to this problem is using polynomials orthogonal with respect to a discrete inner product (5) with the current points in step k . In this way, the Vandermonde matrix is always perfectly conditioned. This solution involves solving the inverse eigenvalue problem (8) of size k in every step, after finding the next point, and evaluating the new set of orthogonal polynomials in the points Y . The inverse eigenvalue problem can be updated one point at the time using the Givens implementation (see Section 4), at a cost of $O(k^2)$ flops per step. Hence, the expensive part of the process is evaluating the new basis functions in the points Y at a cost of $O(Kk^2)$ flops per step.

To avoid the costly procedure of updating the basis in each step, we try to extend the current basis with products of the original basis functions, as explained in Section 4. We keep track of the reciprocal condition number of $V_X^{(k)}$, which is cheap to compute⁶, and only if $V_X^{(k)}$ becomes too badly conditioned we compute a new orthogonal basis. In Figure 3 we plot again the condition number of $V_X^{(k)}$ for the L-shape, now using the adaptations just described. The condition number grows steadily, but once it becomes too large, the basis is updated. For $N = 496$, only 2 costly basis updates have been carried out, which is a significant improvement.

Each time the basis is updated, we recompute the matrix L by solving

⁶MATLAB's RCOND gives an approximation of the reciprocal condition number $\text{cond}(V_X^{(k)})^{-1}$.

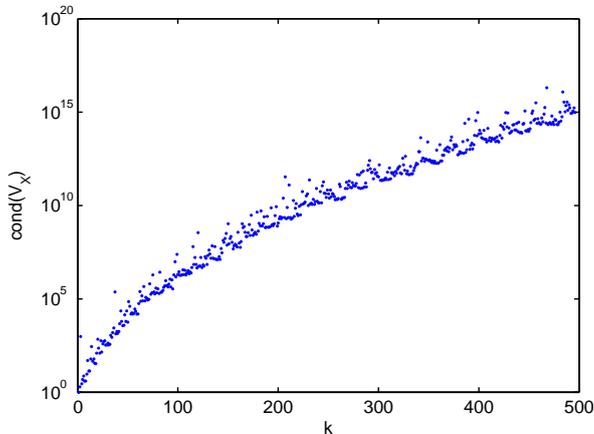


Figure 2: The condition number of the Vandermonde matrix $V_X^{(k)}$ using Algorithm 1 for the L-shape $\Omega = [-1, 1] \times [-1, 0] \cap [-1, 0] \times [0, 1]$, with product Chebyshev polynomials as basis.

a regular linear system. Note that this is not strictly necessary, since the Lagrange polynomials are independent of the basis that is used, so it is possible to continue updating L via low rank updates. However, it might be useful to avoid inaccuracies in the matrix L obtained by the subsequent low rank updates. A stability analysis of these updates is not covered in this paper.

Since the implementation of the adapted Greedy Add Algorithm is a bit too technical to be included in this paper, we refer to the documentation in the code. In Figure 4 the value of the Lebesgue constant is plotted for each iteration of the adapted algorithm, for several pairs of random starting points and for several sizes of the grid Y . Observe that the Lebesgue constant fluctuates a lot, and that the final value Λ_N can be a lot larger than the previous value. This shows that the obtained point configurations are by no means optimal, but they can serve as a starting point for the algorithms in the following sections. In addition, observe that the choice of the starting points influences the obtained Lebesgue constants, as does the size of the grid Y .

The resulting point configuration is shown in Figure 5 for one particular choice of the starting points and the size of the grid, for both the square and the L-shape. In Section 6 we obtain point configurations with nearly optimal Lebesgue constants, which are shown in Figure 8. We observe that the structure in these optimal point configurations is already present in the point configurations obtained by the Greedy Add Algorithm.

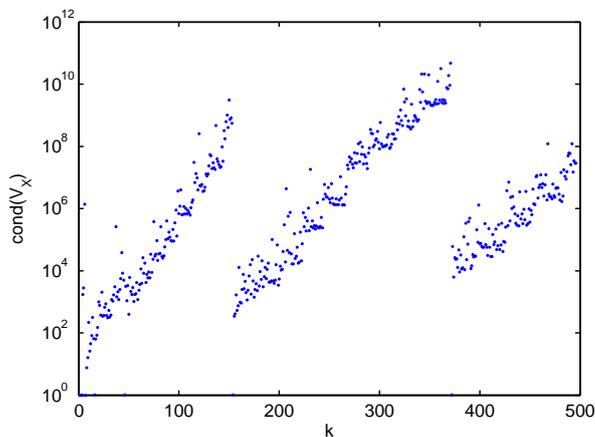


Figure 3: The condition number of the Vandermonde matrix $V_X^{(k)}$ using the adapted version of Algorithm 1 for the L-shape, with orthogonal polynomial updates and basis extension.

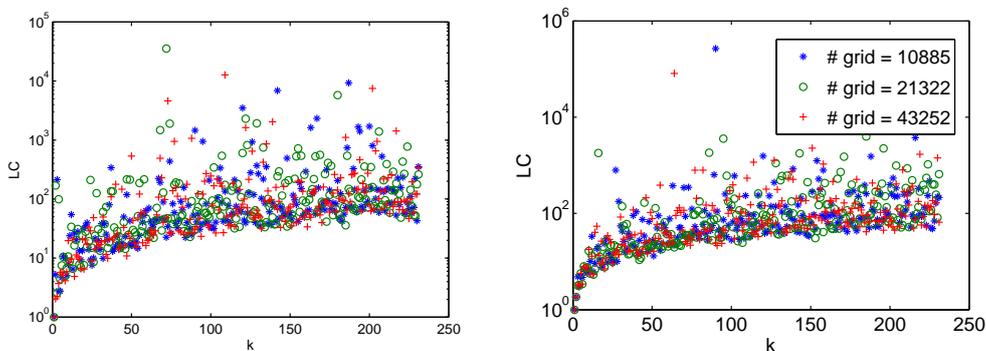


Figure 4: The Lebesgue constant Λ_k after adding the first k points with the adapted Greedy Add Algorithm as a function of k , for several random choices of the first two points (left) and for several sizes of the grid Y (right). The geometry Ω is the square and $\delta = 20$, so $N = 231$. The gridsize for the left plot is 21322.

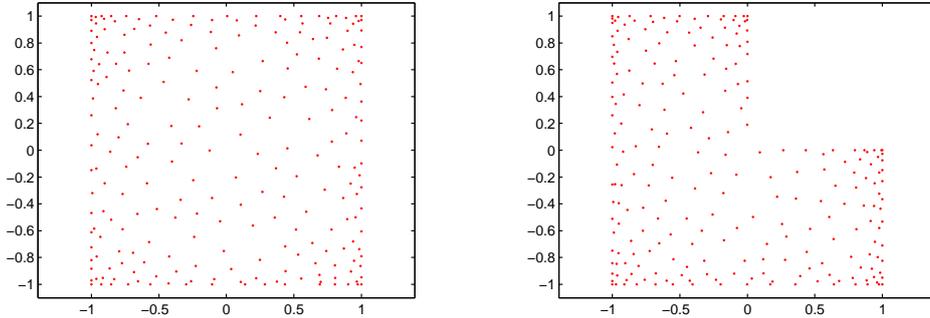


Figure 5: Point configurations X of $N = 231$ points ($\delta = 20$) obtained by the Greedy Add Algorithm for the square on the left, and the L-shape on the right.

5.2 Greedy algorithm by updating points

In this section we develop the Greedy Update Algorithm, implementing a straightforward approach to improve the point configuration $X = \{\mathbf{x}_k\}_1^N$ obtained by the Greedy Add Algorithm of the previous section. The idea is iterate over all the points, remove each point from X and immediately add a new point according to the same greedy criterion. By iterating several times over all the points, the Lebesgue constant typically stabilizes at a reasonably low value.

The algorithm is described schematically in Algorithm 2. The input variables are a point configuration X , e.g., obtained by the Greedy Add Algorithm, a grid $Y \in \Omega$ and the variables needed to evaluate the basis that is used. One possibility is an basis orthogonal with respect to X . We have observed that if the input point configuration X has a low enough Lebesgue constant, then this basis will remain good enough for all the iterations. We have added the functionality that the basis is updated if the Vandermonde matrix $V_X^{(N-1)}$ becomes too badly conditioned.

Similar to the Greedy Add Algorithm, the computation of L in each step can be accelerated by using low rank updates. Indeed, the matrix $V_X^{(N-1)}$ in step $k + 1$ is identical to $V_X^{(N-1)}$ in step k , except for its k -th row. They are the same basis polynomials (columns) evaluated in the same points (rows) except for one. Hence, the matrix of the system is a rank-1 update of the system in the previous step and we can again reduce the amount of work in one step from $O(KN^2)$ to $O(KN)$ flop. The QR factorization of V_X^{N-1} is updated as well.

Figure 6 is an extension of Figure 4, where the value of the Lebesgue

Algorithm 2 Greedy Update Algorithm

Input: X, Y , basis**Output:** X

$V_Y \leftarrow$ evaluate basis functions in grid $Y \in \Omega$
while stopping criterion is not satisfied **do**
 for $k = 1, 2, \dots, N$ **do**
 $X \leftarrow X \setminus \{\mathbf{x}_k\}$
 $V_X^{(N-1)} \leftarrow$ evaluate basis functions in X
 $L \leftarrow V_Y^{(N-1)} = LV_X^{(N-1)}$
 $i \leftarrow$ index of row of L with largest one norm
 $\mathbf{x}_k \leftarrow Y(i)$
 $X \leftarrow X \cup \{\mathbf{x}_k\}$
 end for
end while

constant is plotted for each iteration of the adapted Greedy Add Algorithm and the Greedy Update Algorithm, for several pairs of random starting points and for several sizes of the grid Y . The Greedy Update Algorithm runs for 10 iterations over all the points. We observe that usually the Lebesgue constant stabilizes after a couple of runs and that the value of the final Lebesgue constant depends on the particular choice of the starting points and on the size of the grid.

5.3 Algorithm based on approximating the infinity norm

The infinity norm in (11) is notoriously difficult to optimize using numerical optimization techniques because it combines two of the most exacting objective function properties: taking the maximum over a set and summing (nonsmooth) absolute values. For many initial point sets X , the Lebesgue constant will be quite large and it may suffice to solve a neighbouring problem approximating (11) in order to obtain a substantial reduction of the Lebesgue constant.

5.3.1 Unweighted least squares problem

One approach could be to replace the infinity norm by the (squared) Frobenius norm since

$$\frac{1}{\sqrt{KN}} \|L\|_F \leq \|L\|_\infty \leq \sqrt{N} \|L\|_F. \quad (12)$$

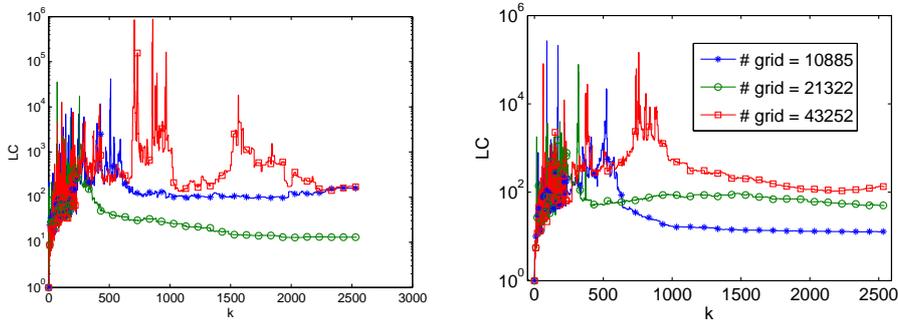


Figure 6: The Lebesgue constant Λ_k at each iteration of the adapted Greedy Add Algorithm and the Greedy Update Algorithm, for several random choices of the first two points (left) and for several sizes of the grid Y (right). The geometry Ω is the square and $\delta = 20$, so $N = 231$. The gridsize for the left plot is 21322.

For example, for $n = 2$ and $\delta = 20$ we have $N = 231$ and hence $\|L\|_F$ bounds $\|L\|_\infty$ from above by about a factor of 15. In practice, the two norms are often even closer than the bound (12) suggests. The objective is now to solve the optimization problem

$$\begin{aligned} \min_{X \subset \Omega} \frac{1}{2} \|L\|_F^2 \\ \text{subject to } V_Y = L V_X. \end{aligned} \quad (13)$$

By eliminating the (linear) constraint in (13), we obtain a nonlinear least squares (NLS) problem in $X \subset \Omega$. There are several algorithms for solving NLS problems, many of which can be adapted for solutions restricted to a domain Ω . In our experiments, we use a projected Gauss-Newton dogleg trust-region method, which is a straightforward generalization of the bound-constrained projected Newton algorithm of [5] to a larger class of geometries. To define a geometry Ω , the user is asked to implement a function which projects points outside of the geometry onto its boundary.

Given a current iterate, the Gauss-Newton dogleg trust-region method computes two additive steps. The first is the Cauchy step \mathbf{p}_{CP} , which is approximated as a scaled steepest descent direction $-\mathbf{g} := \frac{df(\mathbf{z})}{d\mathbf{z}}$, where $\mathbf{z} := \text{vec}(X)$ ⁷ and the objective function $f(\mathbf{z})$ is defined as $\frac{1}{2} \|L\|_F^2$. Here, the points X are stored as $[x_i^{(j)}]_{i,j}$, where $x_i^{(j)}$ is the j th component of the i th point. The

⁷If X is stored in MATLAB as a $N \times n$ matrix, then $\text{vec}(X) := X(:)$ is the $Nn \times 1$ vectorization of X .

second is the Gauss–Newton step

$$\mathbf{p}_{\text{GN}} := -\text{red}(J^T J)^\dagger \mathbf{g}, \quad (14)$$

where the Jacobian J is defined as $\frac{d\text{vec}(L)}{dz^T}$, and $\text{red}(\cdot)$ “reduces” the Hessian approximation $J^T J$ by setting those rows and columns corresponding to the active set equal to those of the identity matrix of the same size as $J^T J$. The active set is defined as the set of indices i for which the variables z_i are on the boundary of the geometry. For more details on the reduction of the Hessian, see [5]. Since J is tall and skinny, its Gramian $J^T J$ is a relatively small square matrix of order Nn . Furthermore, it is a positive (semi-)definite approximation of the objective function’s Hessian and hence may be expected to deliver a high-quality descent direction \mathbf{p}_{GN} for a relatively low computational cost. Importantly, we will see that computing the two descent directions can be done with an amount of computational effort that is *independent of the number of mesh points* K .

To compute the aforementioned descent directions, let

$$W^{(i)} := \left[\begin{array}{ccc} \frac{\partial(V_X)_{1:}^T}{\partial x_1^{(i)}} & \dots & \frac{\partial(V_X)_{N:}^T}{\partial x_N^{(i)}} \end{array} \right]^T$$

be a compact representation of the derivative of V_X with respect to the i th component of the points X . Furthermore, let

$$W := [W^{(1)T} \ \dots \ W^{(N)T}]^T,$$

then after some straightforward computation we find that

$$-\mathbf{g} = -J^T \text{vec}(L) = [(1_{n \times 1} \otimes (V_X^{-T} (V_Y^T V_Y) V_X^{-1})) * W] 1_{N \times 1} \quad (15)$$

and

$$J^T J = (1_{n \times n} \otimes (V_X^{-T} (V_Y^T V_Y) V_X^{-1})) * (W V_X^{-1} V_X^{-T} W^T), \quad (16)$$

where $1_{m \times n}$ is an $m \times n$ matrix of ones, \otimes and $*$ are the Kronecker and Hadamard (or elementwise) product, respectively. Notice that the only computation involving vectors of length K is the term $V_Y^T V_Y$, which need only be computed once and can be done on beforehand. Consequently, the cost per Gauss–Newton iteration is dominated by the cost of solving (14), which requires $O(N^3 n^3)$ flop.

Once the Cauchy and the Gauss–Newton steps are computed, the projected Gauss–Newton dogleg trust-region algorithm proceeds to project them

in such a way that the sum of the current iterate \mathbf{z}_k and these steps does not exceed the boundary of the geometry. In other words, using the user-defined projection function $\text{proj}(\cdot)$, the steps are corrected as

$$\mathbf{p} \leftarrow \text{proj}(\mathbf{z}_k + \mathbf{p}) - \mathbf{z}_k.$$

The dogleg trust-region algorithm then searches for a step which improves the objective function in (a subspace of) the plane spanned by the projected Cauchy and Gauss–Newton steps. For more details on dogleg trust-region, see, e.g., [6].

5.3.2 Weighted least squares problem

Because the Frobenius norm is only a crude approximation for the infinity norm, we introduce a diagonal weighting matrix $D_w = \text{diag}(d_w(i))$ in the least squares optimization problem (13):

$$\begin{aligned} \min_{X \in \Omega} \frac{1}{2} \|D_w L\|_F^2 \\ \text{subject to } V_Y = L V_X. \end{aligned} \tag{17}$$

This problem is solved in an approximate way by performing a small number of Gauss–Newton dogleg trust-region iteration steps⁸. Based on this new approximate solution, the weights $d_w(i)$ are adapted. More weight is put on the points $\mathbf{y}_i \in Y \subset \Omega$ where the Lebesgue function is large. Solving the least squares problem with the adapted weights (17), generically pushes the Lebesgue function down in those subregions where more weight was placed.

To obtain an efficient and effective algorithm, it is crucial to design a good heuristic for this adaptation of the weights. By trial and error, the following heuristic came out as a good choice and was implemented. The number of points \mathbf{y}_i of the set Y is chosen approximately equal to one hundred times the number of points \mathbf{x}_i of X . In total there are one hundred outer iterations each with another adapted weight matrix D_w . Initially the weights are all equal to one. After each outer iteration k the Lebesgue function is computed in all points \mathbf{y}_i and the first $n_y(k)$ largest values are considered. The weight of each of the corresponding points is increased by a fixed amount δ_w , taken equal to 0.4 in our implementation. Note that the number $n_y(k)$ of points \mathbf{y}_i whose weight is increased, depends on the index of the outer iteration. The formula for this number is

$$n_y(k) = \max\{10, N - \lfloor \frac{N}{60} k \rfloor\}$$

⁸In our implementation, the number of iterations is taken equal to two.

with $\lfloor r \rfloor$ the largest integer number less than or equal to the real number r . Hence, in each subsequent iteration, less points are receiving a higher weight until this number is equal to 10 after which it remains constant.

6 Numerical experiments

The algorithms were implemented in MATLAB R2012a and can be obtained from the corresponding author. The experiments were executed on a Linux machine with 2 Intel Xeon Processors E5645 and 48 GByte of RAM.

6.1 Experiment 1: Nearly-optimal point configurations for the square, simplex, disk and L-shape

For each of the geometries, the square, simplex, disk and L-shape, a nearly optimal point configuration X is computed for each of the total degrees $\delta = 3, 4, \dots, 30$. To derive these points, the different optimization algorithms of Section 5 are used subsequently.

First, the Greedy Add Algorithm of Section 5.1 is used to obtain an initial configuration X_1 with a reasonably small Lebesgue constant. The point set Y_1 from which these initial points are taken, is generated by DistMesh with the parameter determined such that approximately $100N$ points are contained in set the Y_1 where N is the number of points of X_1 . This initial configuration is then improved by performing 2 iterations of the Greedy Update Algorithm of Section 5.2, using the same points Y_1 as in the first phase. This improved point configuration X_2 is the initialization of the final phase where the weighted least squares optimization algorithm from Section 5.3.2 is used. For the disk, the same point set Y_1 is used in this final phase. For the polygon-geometries, we generate a triangular mesh based on the points of X_2 together with the edge points of the polygon (square, simplex, L-shape). Each triangle is then divided in a number of subtriangles such that the side lengths are 10 times smaller. This results in a point set Y_2 that contains approximately $100N$ points. Performing 100 outer iterations of the weighted least squares algorithm results in the nearly optimal point configuration $X = X_3$.

In Figure 7 the estimated Lebesgue constant of the resulting set $X = X_3$ is shown for the square, simplex, L-shape and disk, respectively. The estimation of the Lebesgue constant is done by sampling the Lebesgue function on a point set generated as Y_2 based on the point set X_3 for degree 30 and with a multiplication factor 1000 instead of 100. In the subfigures also the results obtained by the CAA-group [3] are given when available.

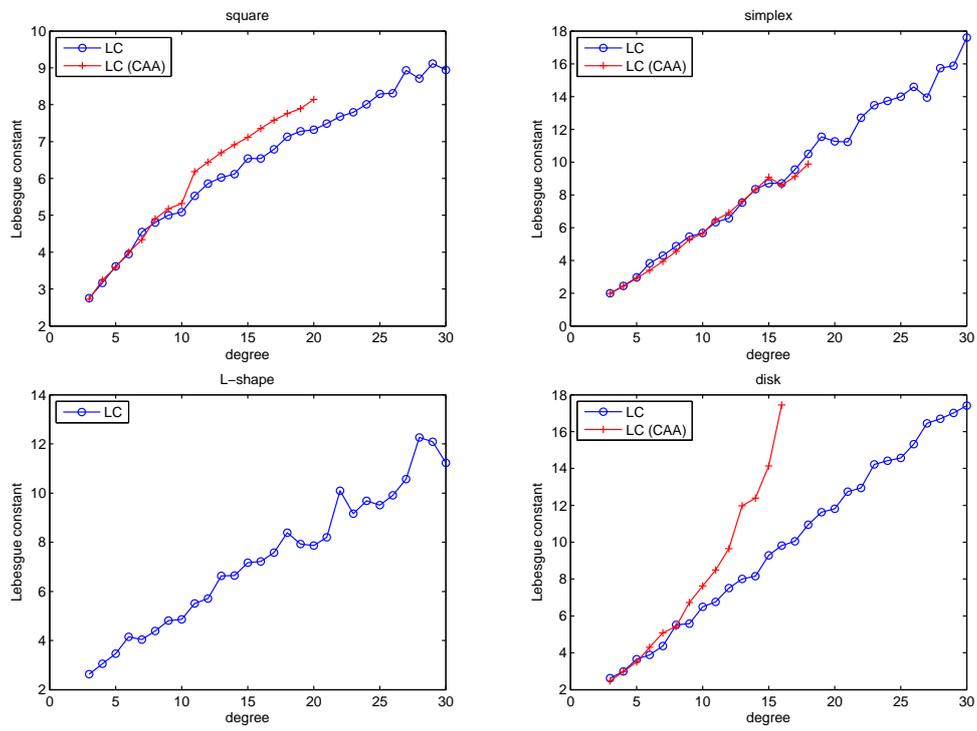


Figure 7: Lebesgue constant in function of the degree for the square, simplex, L-shape and disk

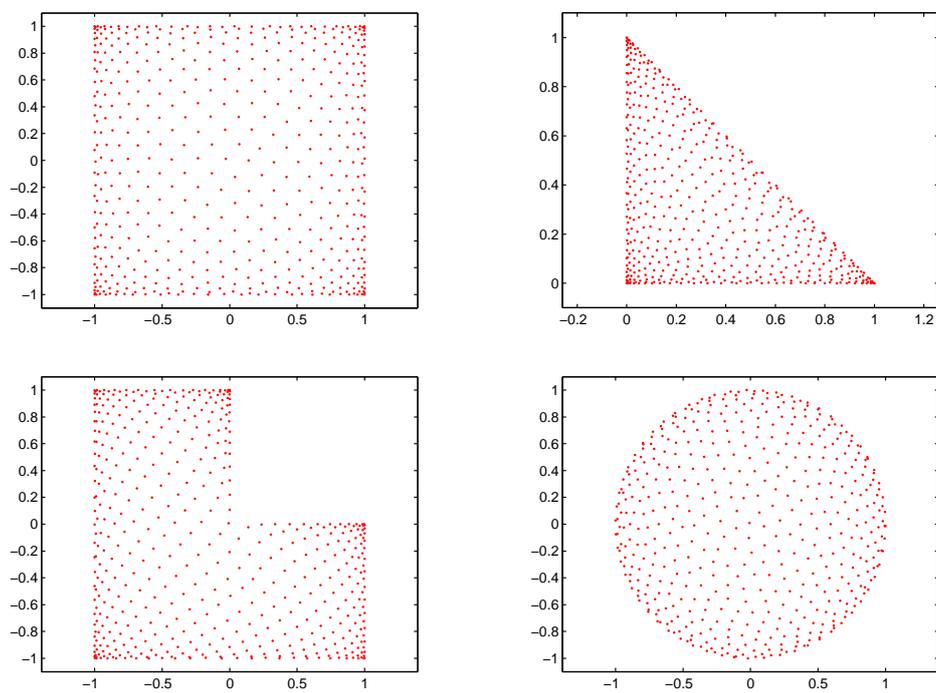


Figure 8: Nearly optimal point configurations of degree 30 for the square, simplex, L-shape and disk

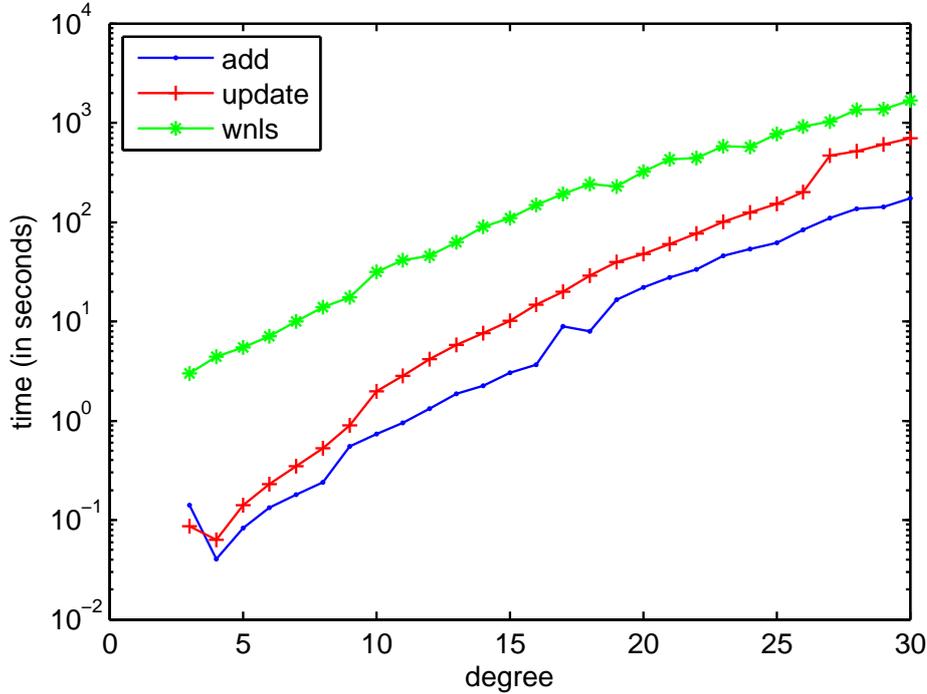


Figure 9: Time (in seconds) for the three phases (greedy add, greedy update, weighted nonlinear least squares) of the algorithm for the square

In Figure 8 the corresponding nearly optimal point configurations for degree 30 are given. In Figure 9 the time for each of the three phases of the algorithm is presented. The lower curve is the time (in seconds) in function of the degree for the Greedy Add Algorithm. The middle curve shows the time for the Greedy Update Algorithm. The upper curve presents the time for the weighted least squares phase.

Compared to the algorithms of [3], to obtain a comparable Lebesgue constant the combined algorithm of this paper needs less computing time.

6.2 Experiment 2: Nearly optimal point set for degree 60 on the square

This experiment shows that much larger nearly optimal point sets can be generated compared to existing techniques. For degree $\delta = 60$, the number of points is $N = 1891$ which is more than 8 times larger than for degree $\delta = 20$. For this experiment, we run only the two first phases of our com-

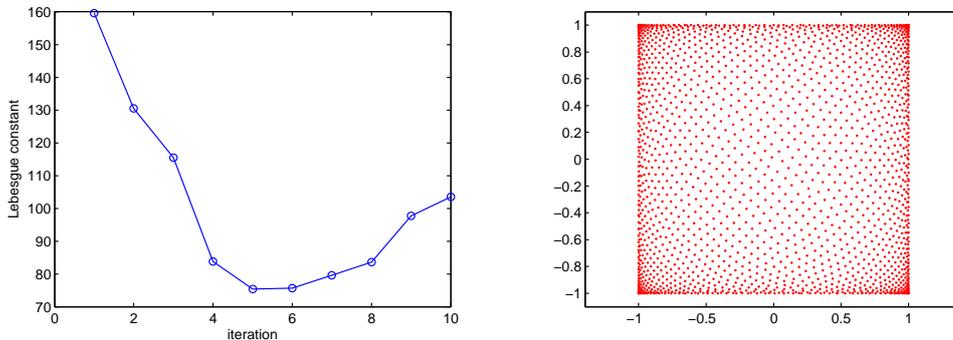


Figure 10: left: Lebesgue constant after each of the 10 iterations of the greedy update step for degree 60 on the square; right: resulting nearly optimal point configuration

bined optimization scheme, i.e., greedy adding and greedy updating, with 10 instead of 2 iterations for the greedy update step. The greedy add step takes 2.16 hours, while the greedy update step takes 23.38 hours. In Figure 10, the estimated Lebesgue constant is shown for each of the 10 iterations of the greedy update step as well as the resulting nearly optimal point configuration having an estimated Lebesgue constant of 75 which was reached in iteration 5.

7 Conclusion

In this paper several optimization algorithms were designed to compute nearly optimal point configurations for different geometries. These algorithms can be combined to derive an efficient and effective algorithm where one algorithm uses the output of the previous one as an initialization. By choosing a representation of the multivariate polynomials in terms of an orthogonal basis with respect to a discrete inner product for a geometry, numerical problems are avoided for larger point sets. Also the efficiency is at least one order of magnitude better compared to existing techniques.

In future research several topics can be studied:

- The different algorithms of Section 5 can be combined in many ways with different heuristics for the number of iterations in the greedy algorithm for updating and the inner and outer iteration of the weighted least squares algorithm. Also different point sets Y can be used in each of the algorithms.

- It is not clear to us if the weighted least squares algorithm that has been developed to approximately solve the minmax optimization problem is known in the literature. At this point it uses a crude heuristic and more investigation is necessary to make this approach more robust. The generalization of this approach to other minmax optimization problems can be studied.

References

- [1] T. Bloom, L. Bos, J.-P. Calvi, and N. Levenberg. Polynomial interpolation and approximation in C^d . *Annales Polonici Mathematici*, 106:53–81, 2012.
- [2] L. Bos and M. Vianello. Low cardinality admissible meshes on quadrangles, triangles and disks. *Mathematical Inequalities and Applications*, 15(1):229–235, 2012.
- [3] M. Briani, A. Sommariva, and M. Vianello. Computing Fekete and Lebesgue points: Simplex, square, disk. *Journal of Computational and Applied Mathematics*, 236:2477–2486, 2012. Not yet available on JCAM website...
- [4] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, Maryland, USA, third edition, 1996.
- [5] C. T. Kelley. *Iterative methods for optimization*. Siam, 1999.
- [6] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer, second edition, 2006.
- [7] P.-O. Persson and G. Strang. A simple mesh generator in MATLAB. *SIAM Review*, 46(2):329–345, 2004.
- [8] L. N. Trefethen. *Approximation Theory and Approximation Practice*. SIAM, 2012.
- [9] M. Van Barel and A. A. Chesnokov. A method to compute recurrence relation coefficients for bivariate orthogonal polynomials by unitary matrix transformations. *Numerical Algorithms*, 55:383–402, 2010.
- [10] J. Van Deun, K. Deckers, A. Bultheel, and J. A. C. Weideman. Algorithm 882: Near-best fixed pole rational interpolation with applications in spectral methods. *ACM Transactions on Mathematical Software*, 35(2):1–20, July 2008. Article 14.