Computing

A Numerical Study of the Xu Polynomial
Interpolation Formula in Two Variables

L. Bos, Calgary, **M. Caliari, S. De Marchi,** Verona and **M. Vianello**, Padova

Abstract

In his paper "Lagrange interpolation on Chebyshev points of two variables" (J. Approx. Theor. 87, 220–238, 1996), Y. Xu proposed a set of Chebyshev like points for polynomial interpolation in the square $[-1, 1]^2$, and derived a compact form of the corresponding Lagrange interpolation formula. We investigate computational aspects of the Xu polynomial interpolation formula like numerical stability and efficiency, the behavior of the Lebesgue constant, and its application to the reconstruction of various test functions.

AMS Subject Classifications: 65D05.

Keywords: Bivariate polynomial interpolation, Xu points, Lagrange interpolation formula, Lebesgue constant.

1. Introduction

The problem of choosing *good* nodes on a given compact set is a central one in polynomial interpolation. As is well-known, the problem is essentially solved in one dimension (all good nodal sequences for the sup-norm are asymptotically equidistributed with respect to the arc-cosine metric), while in several variables it is still substantially open. Even the basic question of unisolvence of a given set of interpolation points is by no means simple, and received special attention in the literature of the last twenty years, cf., e.g. [1], [16], and the survey paper [6]. The same could be said concerning theoretical analysis of the Lebesgue constant, also in the case of the immediate bivariate generalization of one dimensional compact intervals, i.e., squares and rectangles. Another weak point in non tensor-product interpolation is given by a substantial lack of computational efficiency of the interpolation formulas. Improvements have been recently made on general purpose Lagrange and Newton like formulas, cf., e.g. [10], [11], but the problem is still not completely solved.

Recently [3], we studied numerically various nodal sets for polynomial interpolation of degree n (n even) in $[-1, 1]^2$, which are equidistributed with respect to the *Dubiner metric* (a multivariate analog to the arc-cosine metric, cf. [5]). The most promising set, termed *Padua points*, experimentally turned out to be unisolvent and gave a Lebesgue constant growing like $\log^2(n)$. However, the interpolation was

constructed by solving the corresponding Vandermonde system, which means a $\mathcal{O}(N^3)$ complexity (N being the dimension of the underlying full polynomial space, i.e., $N = (n + 1)(n + 2)/2$), plus a $\mathcal{O}(N)$ complexity for each pointwise evaluation of the interpolant.

A very appealing alternative is given by the compact interpolation formula at Chebyshev like points of two variables, proposed some years ago by Y. Xu [15]. Even though the nodal set requires to work in a subspace $\mathcal{V}_n^2$ of the full polynomial space $\mathbb{P}_n^2$, with $\mathbb{P}_{n-1}^2 \subset \mathcal{V}_n^2 \subset \mathbb{P}_n^2$, there is theoretical unisolvence, and the Lebesgue constant grows (experimentally) again like $\log^2(n)$. Moreover, the computational complexity of the *stabilized formula* can be bounded between $c_1 N \sim c_1 n^2/2$ and $c_2 N^{3/2} \sim c_2 n^3/2$ flops (N being here the dimension of $\mathcal{V}_n^2$, $N = \dim \mathcal{V}_n^2 = n(n+2)/2$), for each pointwise construction and evaluation of the interpolant, and in practice remains close to the lower bound (*linear* in the dimension, i.e., quadratic in the degree).

In the next section, we discuss how to implement in an efficient and stable way the Xu interpolation formula. In Sect. 3, we show numerically that the Lebesgue constant of the Xu interpolation operator grows like $(2/\pi \log(n + 1))^2$, which corresponds to the growth of the Lebesgue constant of tensor-product Chebyshev interpolation and of interpolation at Padua points [3]. Finally, in Sect. 4, we apply the Xu interpolation formula to the reconstruction of some test functions with different degrees of regularity, comparing the interpolation errors with those of polynomial interpolation at tensor-product Chebyshev, (extended) Morrow-Patterson and Padua points (cf. [7], [14], [3]).

2. Implementation of the Xu Interpolation Formula

We start by recalling briefly the construction of the Xu interpolation formula of degree n on the square $[-1, 1]^2$. In what follows we restrict to *even* degrees n, in order to compare the present with our previous results (for more details, see [15], [3]). Considering the Chebyshev-Lobatto points on the interval $[-1, 1]$

$$ z_k = z_{k,n} = \cos \frac{k\pi}{n}, \quad k = 0, \dots, n, \quad n = 2m, \tag{1} $$

the Xu interpolation points on the square are defined as the two-dimensional Chebyshev array $X_N = \{\mathbf{x}_{r,s}\}$ of dimension $N = n(n + 2)/2$

$$ \begin{cases} \mathbf{x}_{2i,2j+1} = (z_{2i}, z_{2j+1}), \, 0 \le i \le m, \quad 0 \le j \le m - 1, \\ \mathbf{x}_{2i+1,2j} = (z_{2i+1}, z_{2j}), \, 0 \le i \le m - 1, \, 0 \le j \le m. \end{cases} \tag{2} $$

The Xu interpolant in Lagrange form of a given function f on the square $[-1, 1]^2$ is

$$ L_n^{\text{Xu}} f(\mathbf{x}) = \sum_{\mathbf{x}_{r,s} \in X_N} f(\mathbf{x}_{r,s}) \ell_n(\mathbf{x}, \mathbf{x}_{r,s}), \quad \ell_n(\mathbf{x}, \mathbf{x}_{r,s}) := \frac{K_n^*(\mathbf{x}, \mathbf{x}_{r,s})}{K_n^*(\mathbf{x}_{r,s}, \mathbf{x}_{r,s})}, \tag{3} $$

where the polynomials $K_n^*(\cdot, \mathbf{x}_{r,s})$ are given by

$$
\begin{aligned}
K_n^*(\mathbf{x}, \mathbf{x}_{r,s}) := \frac{1}{2} \left(K_n(\mathbf{x}, \mathbf{x}_{r,s}) + K_{n+1}(\mathbf{x}, \mathbf{x}_{r,s}) \right) + \\
- \frac{1}{2} (-1)^r \left(T_n(x_1) - T_n(x_2) \right) ;
\end{aligned}
\tag{4}
$$

here x_1, x_2 are the coordinates of the generic point $\mathbf{x} = (x_1, x_2)$ and T_n is the Chebyshev polynomial of the first kind of degree n, $T_n(x) = \cos(n \arccos x)$. In particular when $\mathbf{x} = \mathbf{x}_{r,s}$ (cf. [15, formula (2.18)])

$$
K_n^*(\mathbf{x}_{r,s}, \mathbf{x}_{r,s}) = \frac{1}{2} \left(K_n(\mathbf{x}_{r,s}, \mathbf{x}_{r,s}) + K_{n+1}(\mathbf{x}_{r,s}, \mathbf{x}_{r,s}) \right) - 1 .
\tag{5}
$$

It is worth noticing that in formulas (4) and (5) we corrected two misprints appearing in [15, formula (2.15)] and [15, formula (2.18)], respectively: the correct indexes in the sum on the right hand sides are n and $n+1$ instead of n and $n-1$, and moreover the Chebyshev polynomials of the first kind in (4) are not scaled.

The polynomials $K_n(\mathbf{x}, \mathbf{y})$ can be represented in the form

$$
\begin{aligned}
K_n(\mathbf{x}, \mathbf{y}) = D_n(\theta_1 + \phi_1, \theta_2 + \phi_2) + D_n(\theta_1 + \phi_1, \theta_2 - \phi_2) \\
+ D_n(\theta_1 - \phi_1, \theta_2 + \phi_2) + D_n(\theta_1 - \phi_1, \theta_2 - \phi_2) , \\
\mathbf{x} = (\cos \theta_1, \cos \theta_2), \quad \mathbf{y} = (\cos \phi_1, \cos \phi_2) ,
\end{aligned}
\tag{6}
$$

where the function D_n is defined by

$$
D_n(\alpha, \beta) = \frac{1}{2} \frac{\cos((n - 1/2)\alpha) \cos(\alpha/2) - \cos((n - 1/2)\beta) \cos(\beta/2)}{\cos \alpha - \cos \beta} .
\tag{7}
$$

As shown in [15], the values $K_n^*(\mathbf{x}_{r,s}, \mathbf{x}_{r,s})$ are explicitly known in terms of the degree n, that is

$$
K_n^*(\mathbf{x}_{r,s}, \mathbf{x}_{r,s}) = \begin{cases} n^2 & \begin{cases} r = 0 \text{ or } r = n, \ s \text{ odd} \\ s = 0 \text{ or } s = n, \ r \text{ odd} \end{cases} \\ n^2/2 & \text{in all other cases} . \end{cases}
\tag{8}
$$

It is worth saying that the previous formula for $K_n^*(\mathbf{x}_{r,s}, \mathbf{x}_{r,s})$ corrects a misprint in [15, formula (2.16)], concerning the cases $r = n$ and $s = n$.

Observe that this *constructive* approach yields immediately *unisolvence* of the interpolation problem, since for any given basis of the underlying polynomial space $\mathcal{V}_n^2$ the corresponding Vandermonde system has a solution for every N-dimensional vector $\{f(\mathbf{x}_{r,s})\}$, and thus the Vandermonde matrix is invertible.

Rearranging (7) in the case that $\cos(\alpha) = \cos(\beta)$, we virtually have at hand an interpolation formula with pointwise evaluation cost $\mathcal{O}(N)$. However, formula (7) is *numerically ill-conditioned* when $\cos(\alpha) \approx \cos(\beta)$, being like a first divided difference,

and thus has to be stabilized (see Table 1). A stable formula to compute D_n can be obtained by simple trigonometric manipulations. In fact, we can write

$$
\begin{aligned}
D_n(\alpha, \beta) &= \frac{1}{4}\left(\frac{\cos n\alpha - \cos n\beta}{\cos\alpha - \cos\beta} + \frac{\cos(n-1)\alpha - \cos(n-1)\beta}{\cos\alpha - \cos\beta}\right) \\
&= \frac{1}{4}\left(\frac{\sin n\phi \sin n\psi}{\sin\phi \sin\psi} + \frac{\sin(n-1)\phi \sin(n-1)\psi}{\sin\phi \sin\psi}\right) \\
&= \frac{1}{4}\left(U_{n-1}(\cos\phi)U_{n-1}(\cos\psi) + U_{n-2}(\cos\phi)U_{n-2}(\cos\psi)\right), \quad (9)
\end{aligned}
$$

where

$$
\phi = \frac{\alpha - \beta}{2}, \quad \psi = \frac{\alpha + \beta}{2}
$$

and U_n denotes the usual Chebyshev polynomial of the second kind. Formula (9) is a *stabilized* version of (7) for the computation of $D_n(\alpha, \beta)$, where the ratio of differences does not appear explicitly. The Chebyshev polynomial of the second kind U_n has the trigonometric representation $U_n(\cos\theta) = \sin(n+1)\theta/\sin\theta$, which is however numerically ill-conditioned at integer multiples of π, $\theta = k\pi$, $k \neq 0$. On the other hand, as it is well-known, the polynomials U_n can be computed by the three-term recurrence relation

$$
\begin{cases}
U_0(\cos\theta) &= 1, \quad U_1(\cos\theta) = 2\cos\theta, \\
U_n(\cos\theta) &= 2\cos\theta\, U_{n-1}(\cos\theta) - U_{n-2}(\cos\theta), \quad n \geq 2.
\end{cases} \quad (10)
$$

Such a recurrence is *stable* for any θ, by paying the price of a $\mathcal{O}(n)$ instead of $\mathcal{O}(1)$ cost for each evaluation of $D_n(\alpha, \beta)$.

Table 1. Computation of $D_n(0, \beta)$, as $\beta \to 0$, for $n = 4, 8$ with formulae (7) and (9) (NAN = not a number = 0/0)

β	D_4 unstable	D_4 stable	D_8 unstable	D_8 stable
$0.1E + 01$	$0.198154E + 01$	$0.198154E + 01$	$0.756801E + 00$	$0.756801E + 00$
$0.1E + 00$	$0.618528E + 01$	$0.618528E + 01$	$0.269450E + 02$	$0.269450E + 02$
$0.1E - 01$	$0.624935E + 01$	$0.624935E + 01$	$0.282367E + 02$	$0.282367E + 02$
$0.1E - 02$	$0.624999E + 01$	$0.624999E + 01$	$0.282499E + 02$	$0.282499E + 02$
$0.1E - 03$	$0.625000E + 01$	$0.625000E + 01$	$0.282500E + 02$	$0.282500E + 02$
$0.1E - 04$	$0.625000E + 01$	$0.625000E + 01$	$0.282500E + 02$	$0.282500E + 02$
$0.1E - 05$	$0.624999E + 01$	$0.625000E + 01$	$0.282499E + 02$	$0.282500E + 02$
$0.1E - 06$	$0.625341E + 01$	$0.625000E + 01$	$0.282469E + 02$	$0.282500E + 02$
$0.1E - 07$	$0.678905E + 01$	$0.625000E + 01$	$0.278910E + 02$	$0.282500E + 02$
$0.1E - 08$	$0.111111E + 00$	$0.625000E + 01$	$0.111111E + 00$	$0.282500E + 02$
$0.1E - 09$	NAN	$0.625000E + 01$	NAN	$0.282500E + 02$
$0.1E - 10$	NAN	$0.625000E + 01$	NAN	$0.282500E + 02$
$0.1E - 11$	NAN	$0.625000E + 01$	NAN	$0.282500E + 02$
$0.1E - 12$	NAN	$0.625000E + 01$	NAN	$0.282500E + 02$
$0.1E - 13$	NAN	$0.625000E + 01$	NAN	$0.282500E + 02$
$0.1E - 14$	NAN	$0.625000E + 01$	NAN	$0.282500E + 02$
$0.1E - 15$	NAN	$0.625000E + 01$	NAN	$0.282500E + 02$
$0.1E - 16$	NAN	$0.625000E + 01$	NAN	$0.282500E + 02$

At this point, we can compute the complexity of the pointwise evaluation of the Xu interpolation formula (3) via (9)–(10). First, the evaluation of $K_n^*(\mathbf{x}, \mathbf{x}_{r,s})$, which involves four evaluations of D_n and of D_{n+1} via K_n and K_{n+1}, respectively (cf. (4) and (6)), costs about $2n \times 4 = 8n$ flops, since D_n and D_{n+1} with the same arguments can be computed together using the recurrence (10) once. Since this has to be done for every interpolation point, the dominant term in the final complexity for the evaluation of $L_n^{\mathrm{Xu}} f(\mathbf{x})$ is $8nN \sim 4n^3$ flops.

In order to reduce the evaluation cost of the interpolant, one might think to apply a hybrid strategy in the computation of $D_n(\alpha, \beta)$, that is formula (7) when $|\cos\alpha - \cos\beta|$ is not too small, say above a given threshold δ, and the stabilized formula (9) along with the recurrence (10) otherwise. However, we have numerical evidence that choosing $\delta < 2$, which implies that also formula (7) is used, then the interpolation method fails. This can be seen in Table 2, where we show the interpolation errors and the use percentage of stabilized formula (9)–(10) for $n = 20$ on a uniform 100×100 grid for the smooth function $f(x_1, x_2) = \cos(x_1 + x_2)$, corresponding to different choices of δ. The percentages in the last column have been rounded, except for the first and the last which are exact. Notice that when *only* the stabilized formula is applied (i.e., $\delta = 2$) the error is close to machine precision, while in all other cases we have a dramatic loss of accuracy.

An effective way to reduce the computational cost of the stabilized formula (9), still preserving high accuracy, is computing the Chebyshev polynomials of the second kind U_n by the three-term recurrence relation (10) only when the representation $U_n(\cos\theta) = \sin(n+1)\theta / \sin\theta$ (whose cost is $\mathcal{O}(1)$ in n and θ) is ill-conditioned, say for $|\theta - k\pi| \leq \varepsilon$.

In Table 3, we compare the interpolation errors for the same example above with different choices of ε. We can see that with $\varepsilon = 0.01$, which involves the recurrence (10) for less than 1%, we have in practice no loss of accuracy, while for $\varepsilon = 10^{-5}$, where the recurrence is not used at all, we have an error of about 10^{-11}. In this example, for $\varepsilon \leq 0.01$, the algorithm uses almost only the trigonometric representation of $U_n(\cos\theta)$ and thus its computational cost is, in practice, linear in the dimension N, i.e., quadratic in the degree n.

It is worth observing that this behavior is not peculiar to the example just described. Indeed, in practice we have computed the average of the use percentage of the

Table 2. Interpolation errors of $f(x_1, x_2) = \cos(x_1 + x_2)$ on a uniform 100×100 grid at degree $n = 20$ (last column: use percentage of stabilized formula (9)–(10))

δ	$\|L_{20}^{\mathrm{Xu}} f - f\|_\infty$	% stab.
$1.0\mathrm{E} - 10$	$1.9\mathrm{E} + 00$	0
$1.0\mathrm{E} - 05$	$1.9\mathrm{E} + 00$	0.003
$1.0\mathrm{E} + 00$	$1.9\mathrm{E} - 02$	63.50
$1.5\mathrm{E} + 00$	$1.2\mathrm{E} - 02$	83.51
$1.8\mathrm{E} + 00$	$1.1\mathrm{E} - 02$	93.81
$2.0\mathrm{E} + 00$	$6.0\mathrm{E} - 15$	100

Table 3. Interpolation errors of $f(x_1, x_2) = \cos(x_1 + x_2)$ on a uniform 100×100 grid at degree $n = 20$ (last column: use percentage of recurrence relation (10))

ε	$\|L_{20}^{\mathrm{Xu}} f - f\|_\infty$	% recurr.
$1.0\mathrm{E}-05$	$7.0\mathrm{E}-12$	0
$1.0\mathrm{E}-04$	$8.6\mathrm{E}-13$	0.004
$1.0\mathrm{E}-03$	$1.6\mathrm{E}-13$	0.05
$1.0\mathrm{E}-02$	$1.6\mathrm{E}-14$	0.64
$1.0\mathrm{E}-01$	$7.1\mathrm{E}-15$	6.37
$1.0\mathrm{E}+00$	$6.4\mathrm{E}-15$	63.65
$1.0\mathrm{E}+01$	$6.0\mathrm{E}-15$	100

recurrence in evaluating all the Lagrange basis polynomials on a certain uniform grid. If we take random, uniformly distributed evaluation points, such a percentage becomes a random variable (function of the uniform random variable), whose expectation, say η, depends on the threshold ε but not on the degree n. This is clearly seen in Tables 4 and 5, where it is shown that the averages up to one million random points converge to values close to those obtained above on a uniform 100×100 grid, and that these values do not depend on degree n.

Now, the evaluation of $K_n^*(\mathbf{x}, \mathbf{x}_{r,s})$ using only the trigonometric representation of $U_n(\cos\theta)$ costs about $8 \times 4 = 32$ evaluations of the sine function, recalling that D_n and D_{n+1} appear with the same arguments in (4), (6). Denoting by $c_{\sin}$ the *average* evaluation *cost* of the *sine* function (which actually depends on its internal implementation), the average complexity for the evaluation of the Xu interpolant formula $L_n^{\mathrm{Xu}} f(\mathbf{x})$ is of the order of

$$C(n, \varepsilon) := 8n\tau N + 32 c_{\sin}(1 - \tau)N \sim 4n^3\tau + 16 c_{\sin}(1 - \tau)n^2 \text{ flops}, \qquad (11)$$

Table 4. Averages of the use percentage of recurrence relation (10), up to one million uniform random points, in evaluating all the Lagrange basis polynomials at degree $n = 20$

# of random points	% recurr. (averages)	
	$\varepsilon = 0.01$	$\varepsilon = 0.1$
$1.0\mathrm{E}+01$	0.50	7.00
$1.0\mathrm{E}+02$	0.75	6.25
$1.0\mathrm{E}+03$	0.69	6.27
$1.0\mathrm{E}+04$	0.63	6.34
$1.0\mathrm{E}+05$	0.64	6.36
$1.0\mathrm{E}+06$	0.64	6.37

Table 5. Average use percentage η of recurrence relation (10), in evaluating all the Lagrange basis polynomials at different degrees

degree n	percentage η	
	$\varepsilon = 0.01$	$\varepsilon = 0.1$
20	0.64	6.37
40	0.64	6.37
80	0.64	6.37

Table 6. CPU times (seconds) for the evaluation of all the Lagrange basis polynomials on a uniform 100×100 grid

% recurr.	$n = 20$	$n = 30$	$n = 40$	$n = 50$	$n = 60$
0.64 ($\varepsilon = 0.01$)	5.43	11.81	20.64	31.99	45.82
100 ($\varepsilon = 10$)	3.79	9.72	20.38	36.53	59.33

where $\tau = \eta/100$. Considering the experimental value $c_{\sin} = 10$ (obtained with GNU Fortran, see also [13]), we can conclude that, for $\varepsilon \leq 0.01$ (i.e., $\tau \leq 0.0064$), the size of the ratio $C(n, \varepsilon)/N$ remains constant up to degrees of the order of hundreds, that is in practical applications the computational *cost* can be considered *linear* in the number N of points.

A more sophisticated implementation may take into account that, for low degrees, the recurrence relation costs less than the trigonometric representation in evaluating $U_n(\cos\theta)$. Comparing the dominant costs, the algorithm should use only the former when $4n^3 < 16c_{\sin}n^2$, i.e., $n < 4c_{\sin}$.

Our Fortran implementation of the Xu interpolation formula resorts to all the tricks just described, in particular the last with the experimental value $c_{\sin} = 10$, i.e., a threshold degree $n = 40$ (as confirmed by the numerical test reported in Table 6, performed on an AMD Athlon 2800+ processor). The corresponding Fortran77 code is available at the web site *http://www.math.unipd.it/~marcov/software.html*.

3. Numerical Study of the Lebesgue Constant

In this section we discuss another key feature of the Xu interpolation formula, that is the behavior of its Lebesgue constant. First it comes easy to bound the Lebesgue constant linearly in the dimension of the polynomial space $\mathcal{V}_n^2$, which already shows that the Xu points are good candidates for interpolation purposes. Indeed, from the well-known bound for Chebyshev polynomials of the second kind $|U_n(\cos\theta)| \leq n+1$ (cf. [4, p. 306]), we immediately have

$$|D_n(\alpha, \beta)| \leq \frac{1}{4}(n^2 + (n-1)^2) \qquad (12)$$

which implies by (6)

$$|K_n(\mathbf{x}, \mathbf{y})| \leq n^2 + (n-1)^2. \qquad (13)$$

Hence, by means of the explicit representation (8) we get

$$|\ell_n(\mathbf{x}, \mathbf{x}_{r,s})| = \left| \frac{K_n^*(\mathbf{x}, \mathbf{x}_{r,s})}{K_n^*(\mathbf{x}_{r,s}, \mathbf{x}_{r,s})} \right| \leq \frac{(n+1)^2 + 2n^2 + (n-1)^2}{n^2} = 4 + \frac{2}{n^2}. \qquad (14)$$

Defining, in the usual way, the *Lebesgue function* for the Xu interpolation points

$$\lambda_n^{\mathrm{Xu}}(\mathbf{x}) := \sum_{\mathbf{x}_{r,s} \in X_N} |\ell_n(\mathbf{x}, \mathbf{x}_{r,s})|, \qquad (15)$$
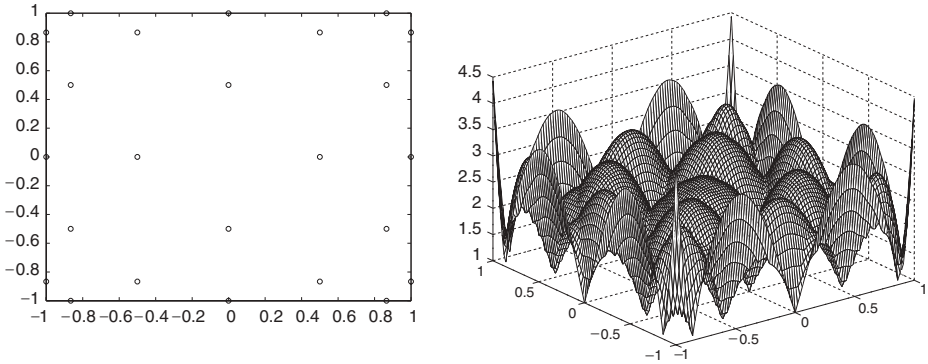
Fig. 1. The $N = 24$ Xu points for $n = 6$ and the corresponding Lebesgue function $\lambda_6^{\text{Xu}}(\mathbf{x})$

we finally obtain the following bound of the Lebesgue constant

$$\Lambda_n^{\text{Xu}} := \max_{\mathbf{x} \in [-1,1]^2} \lambda_n^{\text{Xu}}(\mathbf{x}) \leq \left(4 + \frac{2}{n^2}\right) N \sim 4N \sim 2n^2. \tag{16}$$

However, (16) is an overestimate of the actual Lebesgue constant. In fact, the Lebesgue function turns out to be symmetric and seems to attain its maximum at the four vertices of the square (see Fig. 1 for degree $n = 6$). This fact has been confirmed by a wide set of numerical experiments, where we have maximized the Lebesgue function $\lambda_n^{\text{Xu}}(\mathbf{x})$ up to degree $n = 100$ on a uniform 1000×1000 grid.

In Fig. 2, we compare the Lebesgue constant of Xu points up to degree $n = 100$ computed as $\lambda_n^{\text{Xu}}(1, 1)$, with the least-square fitting function $(0.95 + 2/\pi \, \log(n + 1))^2$ and the theoretical bound for tensor-product interpolation of degree n (cf. [2]), i.e., $(1 + 2/\pi \, \log(n + 1))^2$. These computations give a sound basis for the following
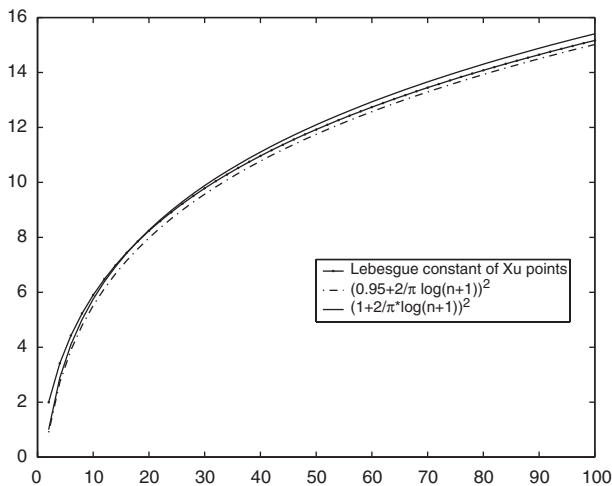

Fig. 2. The Lebesgue constant of Xu points up to degree 100

Conjecture: *The Lebesgue function* λ_n^{Xu} *of Xu interpolation points can be bounded as*

$$\max_{\mathbf{x}\in[-1,1]^2} \lambda_n^{Xu}(\mathbf{x}) = \Lambda_n^{Xu} \lesssim (2/\pi \log(n+1))^2, \quad n \to \infty. \tag{17}$$

Moreover, the maximum is attained at the four vertices of the square.

We stress finally that the Xu points are exactly equally spaced with respect to the Dubiner metric [5], which, on the square $\Omega = [-1, 1]^2$, turns out to be $\mu_\Omega(\mathbf{x}, \mathbf{y}) = \max\{|\arccos x_1 - \arccos y_1|, |\arccos x_2 - \arccos y_2|\}$. This fact confirms once more the conjecture stated in [3]: "nearly optimal interpolation points on a compact Ω are asymptotically equidistributed with respect to the Dubiner metric on Ω".

4. Tests on Functions Interpolation

In this section we compare interpolation at Xu points (XU) with tensor-product Chebyshev (TPC) interpolation, and interpolation at Morrow-Patterson (MP), extended Morrow-Patterson (EMP) and Padua (PD) (cf. [3]) points on three test functions with different degree of regularity, namely the Franke function

$$f_1(x_1, x_2) = \frac{3}{4} e^{-\frac{1}{4}((9x_1-2)^2+(9x_2-2)^2)} + \frac{3}{4} e^{-\frac{1}{49}(9x_1+1)^2 - \frac{1}{10}(9x_2+1)}$$
$$+ \frac{1}{2} e^{-\frac{1}{4}((9x_1-7)^2+(9x_2-3)^2)} - \frac{1}{5} e^{-((9x_1-4)^2+(9x_2-7)^2)},$$

$f_2(x_1, x_2) = (x_1^2 + x_2^2)^{5/2}$ and $f_3(x_1, x_2) = (x_1^2 + x_2^2)^{1/2}$. The above mentioned set of points for $n = 16$, that is MP, EMP, PD points, are displayed in Fig. 3 on the left and for comparison on the right we display the corresponding Xu points.

For MP, EMP and PD points, the interpolant was constructed by solving the corresponding Vandermonde system, whereas the Xu interpolant has been implemented as described in Sect. 2 with $\varepsilon = 0.01$ (cf. Tables 3–5). To obtain an estimate of how the
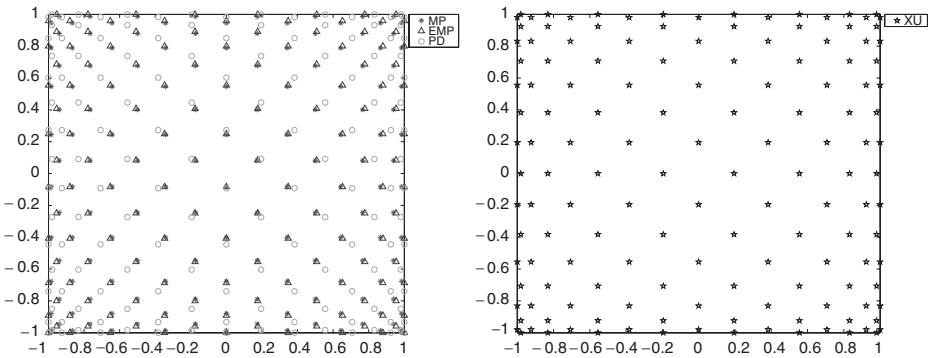

Fig. 3. *Left*: the $N = 153$ Morrow-Patterson (MP), extended Morrow-Patterson (EMP) and Padua (PD) points for $n = 16$. *Right*: the $N = 144$ Xu points for $n = 16$

interpolation errors grow, they have been computed, in infinity norm, on a uniform control grid of 100×100 points. The results are collected in Tables 8–12. For TPC points we have chosen as in [3] the sequence of degrees $n = 24, 34, 44, 54$ and, for the other sets of points, the interpolation degrees have been chosen in such a way that the dimension N of polynomial spaces, and thus the number of function evaluations, is as close as possible to the dimension of the tensor-product polynomial spaces. The resulting sequence of degree is $n = 34, 48, 62, 76$.

In Table 7, we display the Lebesgue constants (rounded to the nearest integer) of MP, EMP, PD and Xu points corresponding to the chosen degrees. As already observed, PD and Xu points have the smallest Lebesgue constants, which are very close to $(1 + 2/\pi \log(n + 1))^2$; see Fig. 2 and [3]. We recall that, denoting by $L_n f$ the interpolation polynomial of degree n on MP, or EMP or PD points, the interpolation error can be estimated by

$$\|f - L_n f\|_{\infty, \Omega} \le (1 + \Lambda_n) E_n(f), \quad \Omega = [-1, 1]^2, \tag{18}$$

where $E_n(f)$ denotes, as usual, the best uniform approximation error to f on Ω by polynomials in $\mathbb{P}_n^2$. As for the Xu points, the corresponding polynomial space $\mathcal{V}_n^2$ is a subspace of $\mathbb{P}_n^2$ with $\mathbb{P}_{n-1}^2 \subset \mathcal{V}_n^2 \subset \mathbb{P}_n^2$, from which we have $E_n(f) \le \inf_{p \in \mathcal{V}_n^2} \|f - p\|_{\infty, \Omega} \le E_{n-1}(f)$ and thus the estimate

$$\|f - L_n^{\mathrm{Xu}} f\|_{\infty, \Omega} \le (1 + \Lambda_n^{\mathrm{Xu}}) \inf_{p \in \mathcal{V}_n^2} \|f - p\|_{\infty, \Omega} \le (1 + \Lambda_n^{\mathrm{Xu}}) E_{n-1}(f). \tag{19}$$

The rate of decay of $E_n(f)$ as $n \to \infty$ depends on the degree of smoothness of f, in view of multivariate generalizations of Jackson's theorem (cf. [8]).

Some comments on the results in Tables 8–12 are now in order. First, we observe that in these examples, Xu interpolation errors are almost always very close to those

Table 7. Lebesgue constants (rounded to the nearest integer)

interp. pts.	Λ_{34}	Λ_{48}	Λ_{62}	Λ_{76}
MP	649	1264	2082	3102
EMP	237	456	746	1106
PD	11	13	14	15
XU	10	12	13	14

Table 8. Interpolation errors on $[0, 1]^2$ for the Franke function

TPC $n, N = (n+1)^2$	1.3E-03	2.6E-06	1.1E-09	2.0E-13
	24, 625	34, 1225	44, 2025	54, 3025
MP $n, N = (n+1)(n+2)/2$	1.3E-03	2.6E-06	1.1E-09	2.0E-13
	34, 630	48, 1225	62, 2016	76, 3003
EMP $n, N = (n+1)(n+2)/2$	6.3E-04	1.3E-06	5.0E-10	5.4E-14
	34, 630	48, 1225	62, 2016	76, 3003
PD $n, N = (n+1)(n+2)/2$	4.3E-05	3.3E-08	5.4E-12	1.9E-14
	34, 630	48, 1225	62, 2016	76, 3003
XU $n, N = n(n+2)/2$	3.2E-05	4.7E-08	7.8E-12	1.9E-13
	34, 612	48, 1200	62, 1984	76, 2964

Table 9. Interpolation errors on $[-1, 1]^2$ for the function $f_2(x_1, x_2) = (x_1^2 + x_2^2)^{5/2}$

TPC	6.0E-05	8.2E-06	1.8E-06	5.4E-07
$n, \ N = (n+1)^2$	24, 625	34, 1225	44, 2025	54, 3025
MP	1.8E-04	5.1E-05	1.9E-05	8.8E-06
$n, \ N = (n+1)(n+2)/2$	34, 630	48, 1225	62, 2016	76, 3003
EMP	6.5E-05	1.8E-05	6.7E-06	3.0E-06
$n, \ N = (n+1)(n+2)/2$	34, 630	48, 1225	62, 2016	76, 3003
PD	3.6E-06	6.5E-07	1.8E-07	6.5E-08
$n, \ N = (n+1)(n+2)/2$	34, 630	48, 1225	62, 2016	76, 3003
XU	7.1E-06	1.2E-06	3.4E-07	1.2E-07
$n, \ N = n(n+2)/2$	34, 612	48, 1200	62, 1984	76, 2964

Table 10. Interpolation errors on $[0, 2]^2$ for the function in Table 9

TPC	8.5E-09	1.7E-10	1.4E-11	1.1E-11
$n, \ N = (n+1)^2$	24, 625	34, 1225	44, 2025	54, 3025
MP	1.0E-08	3.8E-10	3.7E-11	2.3E-11
$n, \ N = (n+1)(n+2)/2$	34, 630	48, 1225	62, 2016	76, 3003
EMP	7.2E-09	2.6E-10	2.4E-11	8.6E-12
$n, \ N = (n+1)(n+2)/2$	34, 630	48, 1225	62, 2016	76, 3003
PD	2.8E-09	9.3E-11	9.4E-12	6.4E-12
$n, \ N = (n+1)(n+2)/2$	34, 630	48, 1225	62, 2016	76, 3003
XU	3.2E-09	1.1E-10	1.6E-11	4.5E-12
$n, \ N = n(n+2)/2$	34, 612	48, 1200	62, 1984	76, 2964

Table 11. Interpolation errors on $[-1, 1]^2$ for the function $f_3(x_1, x_2) = (x_1^2 + x_2^2)^{1/2}$

TPC	2.1E-01	1.1E-01	6.8E-02	4.6e-02
$n, \ N = (n+1)^2$	24, 625	34, 1225	44, 2025	54, 3025
MP	4.4E-01	4.4E-01	4.4E-01	4.4E-01
$n, \ N = (n+1)(n+2)/2$	34, 630	48, 1225	62, 2016	76, 3003
EMP	1.4E-01	1.4E-01	1.4E-01	1.4E-01
$n, \ N = (n+1)(n+2)/2$	34, 630	48, 1225	62, 2016	76, 3003
PD	3.7E-02	2.7E-02	2.1E-02	1.7E-02
$n, \ N = (n+1)(n+2)/2$	34, 630	48, 1225	62, 2016	76, 3003
XU	5.1E0-2	3.6E-02	2.8-02	2.3E-02
$n, \ N = n(n+2)/2$	34, 612	48, 1200	62, 1984	76, 2964

of interpolations at PD points, and smaller than the errors given by the other sets of points. Notice that on the functions f_2 and f_3, which have a singularity at the origin, interpolation performs always better when the singularity is located at the corner of the square, where all the five sets of points cluster by construction. Xu points exhibit the worst behavior only in the test of Table 12, especially for high degrees: this may be ascribed to the presence of a strong singularity (discontinuity of the gradient) at the origin, around which more points of the other sets cluster.

Table 12. Interpolation errors on $[0, 2]^2$ for the function in Table 11

TPC	2.8E-03	5.8E-04	1.1E-04	8.9E-05
$n,\ N = (n+1)^2$	24, 625	34, 1225	44, 2025	54, 3025
MP	8.8E-04	2.8E-04	2.6E-04	1.7E-05
$n,\ N = (n+1)(n+2)/2$	34, 630	48, 1225	62, 2016	76, 3003
EMP	8.3E-04	2.6E-04	2.1E-04	2.1E-05
$n,\ N = (n+1)(n+2)/2$	34, 630	48, 1225	62, 2016	76, 3003
PD	7.3E-04	3.7E-04	7.0E-06	4.6E-06
$n,\ N = (n+1)(n+2)/2$	34, 630	48, 1225	62, 2016	76, 3003
XU	9.4E-04	4.7E-04	2.8E-04	1.9E-04
$n,\ N = n(n+2)/2$	34, 612	48, 1200	62, 1984	76, 2964

4.1. Efficiency and Robustness

To support the efficiency and robustness of our implementation of the Xu interpolation formula, we did some comparisons with the `MPI software` by T. Sauer (cf. [10], [11]). The MPI software is one of the most efficient and robust implementations of multivariate interpolation by polynomials via finite differences and is based on the notion of *blockwise interpolation*. Calling XU our implementation of the Xu interpolation formula, we compared the CPU times necessary to build the interpolant and the interpolation errors for both XU and MPI on all our tests functions. The tests were performed, as explained above, on a AMD Athlon 2800+ processor machine, with both codes compiled using the optimization option `-O3`. The results are collected in Tables 13–16. Furthermore, all tests with MPI, were done modifying only the template file `demo.cc` to work with two-dimensional point sets. The target

Table 13. CPU times (in seconds) and interpolation errors on $[0, 1]^2$ of XU and MPI for the Franke function

n	20	30	40	50	60
XU	2.1	5.2	10.3	17.8	28.4
	7.3E-03	3.6E-04	3.1E-06	1.8E-08	2.5E-11
MPI	0.6	Unsolv.	Unsolv.	Unsolv.	Unsolv.
	3.8E-02	∗∗∗	∗∗∗	∗∗∗	∗∗∗

Table 14. CPU times (in seconds) and interpolation errors of MPI for the Franke function on different domains by a change of variables and reordering the Xu points as Leja sequences

n	20	30	40	50	60
MPI	0.6	4.3	21.0	75.6	Unsolv.
$[-1, 1]^2$	6.3E-03	3.5E-04	2.0E-01	3.8E-02	∗∗∗
MPI	0.5	3.7	17.4	62.3	183.4
$[-2, 2]^2$	6.4E-03	1.0E-02	2.7E+02	1.3E+14	1.9E+35
MPI-Leja	0.6	4.3	21.0	75.6	Unsolv.
$[-1, 1]^2$	6.4E-03	3.5E-04	1.1E-04	2.0E-03	∗∗∗

Table 15. CPU times (in seconds) and interpolation errors on $[-1, 1]^2$ of XU and MPI for the function
$$f_2(x_1, x_2) = (x_1^2 + x_2^2)^{1/2}$$

n	20	30	40	50	60
XU	2.1	5.2	10.3	17.8	28.4
	8.7E-02	5.8E-02	4.3E-02	3.5E-02	2.9E-02
MPI	0.6	4.3	20.8	74.8	Unsolv.
	8.7E-02	5.8E-02	2.8E+00	4.7E+00	$***$

Table 16. CPU times (in seconds) and interpolation errors on $[-1, 1]^2$ of XU and MPI for the function
$$f_3(x_1, x_2) = (x_1^2 + x_2^2)^{5/2}$$

n	20	30	40	50	60
XU	2.1	5.2	10.3	17.8	28.4
	1.1E-04	1.3E-05	3.1E-06	1.0E-06	4.0E-07
MPI	0.6	4.3	21.0	75.3	Unsolv.
	1.1E-04	1.3E-05	1.8E-03	4.8E-03	$***$

points on which we evaluate the interpolation errors were chosen as before, that is a grid of 100×100 points on the reference square.

Remarks: We performed many experiments with different functions and square domains, but all showed the same behavior: the MPI software works quite well, both in terms of CPU times and interpolation errors, for small degrees n but for higher degrees it becomes unstable ($n = 40, 50$) or unusable ($n = 60$). This has been verified also applying stabilization techiniques for the divided differences (on which the MPI software is ultimately based) like scaling the domain or reordering the interpolation points as *Leja sequences* (cf. [9], [12]). See the details for the Franke function in Tables 13 and 14. On the contrary, XU can suitably manage interpolation up to very high degrees.

5. Conclusions and Future Works

In this paper, we investigated the numerical aspects of Xu interpolation formula and, in particular, we obtained an efficient and stable formula for its computation and tight numerical bounds of the growth of the associated Lebesgue constant. Our numerical study shows that Xu interpolation gives almost the best one can do with polynomials on the square. In fact, besides the very good behavior of its Lebesgue constant (which grows like *logarithm square* of the degree), it can be implemented by an algorithm which has in practice a *linear* cost in the number of interpolation points (i.e., in the dimension of the underlying polynomial space).

However, important theoretical aspects have still to be faced. Supported by our results, it would be interesting first to provide analytically a precise growth rate of the Lebesgue constant and, as suggested by Fig. 1, prove theoretically that the

maxima of the Lebesgue function are taken at the vertices of the square $[-1, 1]^2$; see the conjecture at the end of Sect. 3. As we already observed, in this way we could simply restrict on finding an explicit formula for $\lambda_n^{\mathrm{Xu}}(1, 1)$.

References

[1] Bos, L.: On certain configurations of points in $\mathbb{R}^n$ which are unisolvent for polynomial interpolation. J. Approx. Theor. *64(3)*, 271–280 (1991).
[2] Brutman, L.: Lebesgue functions for polynomial interpolation – a survey. Ann. Numer. Math. *4*, 111–127 (1997).
[3] Caliari, M., De Marchi, S., Vianello, M.: Bivariate polynomial interpolation on the square at new nodal sets. Appl. Math. Comput. *165*, 261–274 (2005).
[4] Davis, P. J.: Interpolation and Approximation. New York: Dover Publications 1975.
[5] Dubiner, M.: The theory of multi-dimensional polynomial approximation. J. Anal. Math. *67*, 39–116 (1995).
[6] Gasca, M., Sauer, T.: Polynomial interpolation in several variables. Adv. Comput. Math. *12*, 377–410 (2000).
[7] Morrow, C. R., Patterson, T. N. L.: Construction of algebraic cubature rules using polynomial ideal theory. SIAM J. Numer. Anal. *15(5)*, 953–976 (1978).
[8] Pleśniak, W.: Remarks on Jackson's theorem in $\mathbb{R}^N$. East J. Approx. *2(3)*, 301–308 (1996).
[9] Reichel, L.: Newton interpolation at Leja points. BIT *30*, 332–346 (1990).
[10] Sauer, T.: Computational aspects of multivariate polynomial interpolation. Adv. Comput. Math. *3*, 219–238 (1995).
[11] Sauer, T., Xu, Y.: On multivariate Lagrange interpolation. Math. Comp. *64*, 1147–1170 (1995).
[12] Tal-Ezer, H.: High degree polynomial interpolation in Newton form. SIAM J. Sci. Stat. Comput. *12(3)*, 648–667 (1991).
[13] Tang, P. T. P.: Some software implementations of the functions sine and cosine. ANL Report 90/3, Argonne National Laboratory, April 1990.
[14] Xu, Y.: Gaussian cubature and bivariate polynomial interpolation. Math. Comp. *59*, 547–555 (1992).
[15] Xu, Y.: Lagrange interpolation on Chebyshev points of two variables. J. Approx. Theor. *87*, 220–238 (1996).
[16] Xu, Y.: Polynomial interpolation on the unit sphere and on the unit ball. Adv. Comput. Math. *20*, 247–260 (2004).

L. Bos
2500 University Drive NW
Calgary
Alberta T2N1N4
Canada
e-mail: lpbos@math.ucalgary.ca

M. Caliari
Dept. of Computer Science
University of Verona
37129 Verona
Italy

S. De Marchi
Dept. of Computer Science
University of Verona
37129 Verona
Italy

M. Vianello
Dept. of Pure and Applied Mathematics
University of Padova
35122 Padova
Italy