Available online at www.sciencedirect.com
ScienceDirect
Journal of Computational and Applied Mathematics 221 (2008) 284–292
JOURNAL OF COMPUTATIONAL AND APPLIED MATHEMATICS
www.elsevier.com/locate/cam

# Bivariate Lagrange interpolation at the Padua points: Computational aspects☆

Marco Caliari[a], Stefano De Marchi[b,*], Marco Vianello[a]

[a] *Department of Pure and Applied Mathematics, University of Padua, via Trieste 63, 35121 Padova, Italy*
[b] *Department of Computer Science, University of Verona S.da Le Grazie 15, 37134 Verona, Italy*

Received 19 December 2006; received in revised form 25 May 2007

**Abstract**

The so-called "Padua points" give a simple, geometric and explicit construction of bivariate polynomial interpolation in the square. Moreover, the associated Lebesgue constant has minimal order of growth $\mathcal{O}(\log^2(n))$. Here we show four families of Padua points for interpolation at any even or odd degree $n$, and we present a stable and efficient implementation of the corresponding Lagrange interpolation formula, based on the representation in a suitable orthogonal basis. We also discuss extension of (non-polynomial) Padua-like interpolation to other domains, such as triangles and ellipses; we give complexity and error estimates, and several numerical tests.


*Keywords:* Bivariate polynomial interpolation; Square; Padua points; Bivariate Chebyshev orthogonal polynomials; Reproducing kernel

## 1. Introduction

Finding "good" nodes is a challenging problem in multivariate polynomial interpolation. Besides unisolvence, which is itself a difficult topic (see, e.g., [3,11,14]), in order to get stability and convergence one seeks slow growth of the Lebesgue constant.

In some recent papers, we studied a new set of points for bivariate polynomial interpolation in the square $[-1, 1]^2$, nicknamed "Padua points" (cf. [10,4,7]). Such points allow us to give a simple, geometric and explicit construction of the interpolation formula, since the Lagrange polynomials are written in terms of the reproducing kernel corresponding to the product Chebyshev measure. Moreover, the Padua points have a Lebesgue constant with minimal order of growth $\mathcal{O}(\log^2(n))$, as has been rigorously proved in [4] for the upper bound and in [12,16] for the exact order of growth.

In this paper, we exploit the explicit formula of the Lagrange polynomials to obtain a stable and efficient representation of the interpolation polynomial at the Padua points, in terms of a classical orthonormal basis associated with the product Chebyshev measure.

☆ Work supported by the ex-60% funds of the Universities of Padova and Verona, and by the GNCS-INdAM.
* Corresponding author.
*E-mail addresses:* stefano.demarchi@univr.it, demarchi@sci.univr.it (S. De Marchi).

0377-0427/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.cam.2007.10.027

The paper is organized as follows. In the next section we list four families of Padua points, which are here displayed together explicitly for the first time, and we recall the associated interpolation formulas. In Section 3 we describe in detail a stable and efficient implementation of interpolation at Padua points on rectangles, and we analyze its computational cost and a related a posteriori error estimate. Moreover, we discuss extension to rectangles and to non-polynomial Padua-like interpolation on domains with different geometric structures, like triangles and ellipses. Finally, in Section 4 we show the behavior of the interpolation formula on a classical test set.

## 2. Interpolation at the Padua points

The Padua points were introduced [10] for even degrees as unions of two Chebyshev-like grids, and their properties in bivariate interpolation studied numerically. In [4] their Lagrange polynomials were constructed explicitly, using the fact that the points lie on an algebraic curve, the "generating curve", and that they provide a cubature formula of high algebraic degree of exactness. On the other hand, in [7] the problem of interpolation at the Padua points has been faced in an abstract algebraic setting (polynomial ideal theory and multivariate orthogonal polynomials).

The points considered in [4], however, are not the original Padua points, but correspond to a rotation of 90 degrees. In fact, there are four families of Padua points, obtainable from one another by a suitable rotation of 90 or 180 degrees. Below we list them together for the first time, and for each family we give the corresponding generating curve as well as the description (for both even and odd degrees) as a union of two Chebyshev-like grids.

We observe that:

- For each family, the Padua points are the self-intersections and boundary contacts of the generating curve in $[-1,1]^2$, and they match exactly the dimension of $\Pi_n^2$, the space of polynomials of degree at most $n$. In particular there are two points lying on consecutive vertices of the square (the "top", "bottom", "left" and "right" pairs of vertices), other $2n-1$ points lying on the edges of the square, the remaining points being self-intersections of the corresponding generating curve.
- The Padua points are nodes of a cubature formula for the product Chebyshev measure which is exact for all polynomials in a suitable subspace of $\Pi_{2n}^2$, containing $\Pi_{2n-1}^2$ (cf. [4]). Given a Padua point, say $\boldsymbol{\xi}$, the corresponding cubature weight is

$$w_{\boldsymbol{\xi}} = \frac{1}{n(n+1)} \cdot \begin{cases} 1/2 & \text{if } \boldsymbol{\xi} \text{ is a vertex point} \\ 1 & \text{if } \boldsymbol{\xi} \text{ is an edge point} \\ 2 & \text{if } \boldsymbol{\xi} \text{ is an interior point} \end{cases} \tag{1}$$

- The first family is that of the original Padua points; the others correspond to successive rotations of 90 degrees, clockwise for even degrees and counterclockwise for odd degrees.

In order to describe the four families and the corresponding interpolation formulas, we need the following notation:

$$z_j^d = \cos\frac{j\pi}{d}, \quad j = 0,\dots,d; \qquad \mathrm{Pad}_n^s = \{\boldsymbol{\xi} = (\xi_1,\xi_2)\} = A^s \cup B^s,$$

where $A^s$ and $B^s$ are two grids of points that will be defined below for each family.

$$N = \mathrm{card}\left(\mathrm{Pad}_n^s\right) = \dim\left(\Pi_n^2\right) = \frac{(n+1)(n+2)}{2}, \quad s = 1,2,3,4, \tag{2}$$

and we recall that the reproducing kernel of $\Pi_n^2([-1,1]^2)$ corresponding to the inner product generated by the product Chebyshev measure can be written as

$$K_n(\boldsymbol{x},\boldsymbol{y}) = \sum_{k=0}^{n}\sum_{j=0}^{k} \hat{T}_j(x_1)\hat{T}_{k-j}(x_2)\hat{T}_j(y_1)\hat{T}_{k-j}(y_2), \tag{3}$$

where $\hat{T}_p$ is the normalized Chebyshev polynomials of degree $p$ (i.e., $\hat{T}_0 = 1$, $\hat{T}_p = \sqrt{2}T_p$, $T_p(\cdot) = \cos(p\arccos(\cdot))$ being the usual Chebyshev polynomial of degree $p$); see, e.g., [13].

First family: $\mathrm{Pad}_n^1$:

generating curve: $T_n(x) + T_{n+1}(y) = 0$;

parametrization: $\gamma_1(t) = [-\cos((n+1)t), -\cos(nt)]$, $0 \le t \le \pi$;

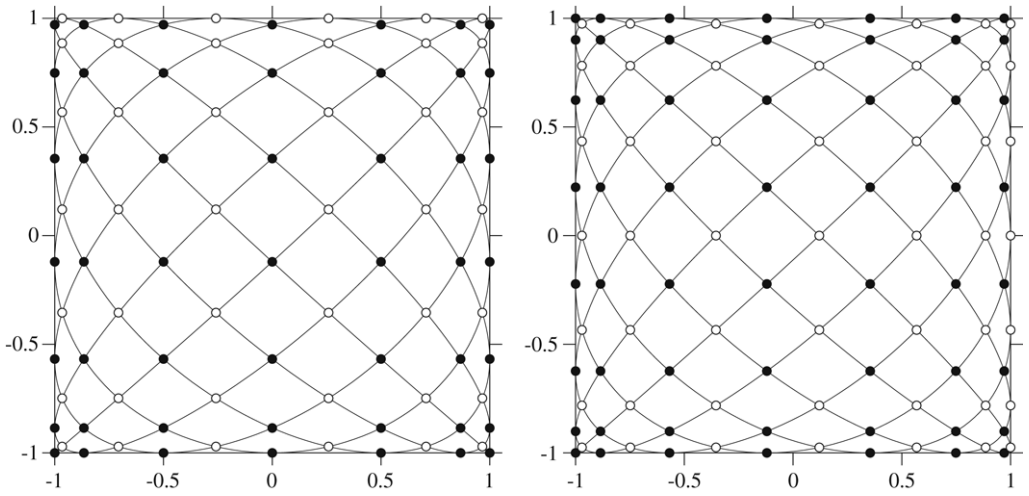


Fig. 1. The first family of Padua points with the generating curves for $n = 12$ (left, 91 points) and $n = 13$ (right, 105 points), also as a union of two Chebyshev-like grids, $A$ (empty bullets) and $B$ (full bullets).

- $n$ even, $n = 2m$ (see Fig. 1-left),

$$\begin{cases} A^1_{\text{even}} = \{(z^n_{2i+1}, z^{n+1}_{2j}),\ 0 \le i \le m-1,\ 0 \le j \le m\} \\ B^1_{\text{even}} = \{(z^n_{2i}, z^{n+1}_{2j+1}),\ 0 \le i \le m,\ 0 \le j \le m\}. \end{cases} \tag{4a}$$

  These correspond to the points defined in [10, formula (9)] (in that formula there is a misprint; $n-1$ has to be replaced by $n+1$).
- $n$ odd, $n = 2m+1$ (see Fig. 1-right),

$$\begin{cases} A^1_{\text{odd}} = \{(z^n_{2i+1}, z^{n+1}_{2j}),\ 0 \le i \le m,\ 0 \le j \le m+1\} \\ B^1_{\text{odd}} = \{(z^n_{2i}, z^{n+1}_{2j+1}),\ 0 \le i \le m,\ 0 \le j \le m\}. \end{cases} \tag{4b}$$

- Lagrange interpolant:

$$\mathcal{L}_{\text{Pad}^1_n} f(\boldsymbol{x}) = \sum_{\boldsymbol{\xi} \in \text{Pad}^1_n} f(\boldsymbol{\xi}) w_{\boldsymbol{\xi}} \left(K_n(\boldsymbol{x}, \boldsymbol{\xi}) - T_n(x_1) T_n(\xi_1)\right). \tag{4c}$$

Second family: $\text{Pad}^2_n$:

generating curve: $T_{n+1}(x) + T_n(y) = 0$;
parametrization: $\gamma_2(t) = [-\cos(nt), -\cos((n+1)t)]$, $0 \le t \le \pi$;

- $n$ even, $n = 2m$,

$$\begin{cases} A^2_{\text{even}} = \{(z^{n+1}_{2i+1}, z^n_{2j}),\ 0 \le i \le m,\ 0 \le j \le m\} \\ B^2_{\text{even}} = \{(z^{n+1}_{2i}, z^n_{2j+1}),\ 0 \le i \le m,\ 0 \le j \le m-1\}. \end{cases} \tag{5a}$$

- $n$ odd, $n = 2m+1$,

$$\begin{cases} A^2_{\text{odd}} = \{(z^{n+1}_{2i+1}, z^n_{2j}),\ 0 \le i \le m,\ 0 \le j \le m\} \\ B^2_{\text{odd}} = \{(z^{n+1}_{2i}, z^n_{2j+1}),\ 0 \le i \le m+1,\ 0 \le j \le m\}. \end{cases} \tag{5b}$$

- Lagrange interpolant:

$$\mathcal{L}_{\text{Pad}^2_n} f(\boldsymbol{x}) = \sum_{\boldsymbol{\xi} \in \text{Pad}^2_n} f(\boldsymbol{\xi}) w_{\boldsymbol{\xi}} \left(K_n(\boldsymbol{x}, \boldsymbol{\xi}) - T_n(x_2) T_n(\xi_2)\right). \tag{5c}$$

Third family: $\mathrm{Pad}_n^3$:

generating curve: $T_n(x) - T_{n+1}(y) = 0$;
parametrization: $\gamma_3(t) = [\cos((n+1)t), \cos(nt)]$, $0 \le t \le \pi$;

- $n$ even, $n = 2m$,

$$\begin{cases} A^3_{\mathrm{even}} = \{(z^n_{2i}, z^{n+1}_{2j}),\ 0 \le i \le m,\ 0 \le j \le m\} \\ B^3_{\mathrm{even}} = \{(z^n_{2i+1}, z^{n+1}_{2j+1}),\ 0 \le i \le m-1,\ 0 \le j \le m\}. \end{cases} \tag{6a}$$

- $n$ odd, $n = 2m+1$,

$$\begin{cases} A^3_{\mathrm{odd}} = \{(z^n_{2i}, z^{n+1}_{2j}),\ 0 \le i \le m,\ 0 \le j \le m+1\} \\ B^3_{\mathrm{odd}} = \{(z^n_{2i+1}, z^{n+1}_{2j+1}),\ 0 \le i \le m,\ 0 \le j \le m\}. \end{cases} \tag{6b}$$

- Lagrange interpolant:

$$\mathcal{L}_{\mathrm{Pad}_n^3} f(\boldsymbol{x}) = \sum_{\boldsymbol{\xi} \in \mathrm{Pad}_n^3} f(\boldsymbol{\xi}) w_{\boldsymbol{\xi}} \left( K_n(\boldsymbol{x}, \boldsymbol{\xi}) - T_n(x_1) T_n(\xi_1) \right). \tag{6c}$$

Fourth family: $\mathrm{Pad}_n^4$:

generating curve: $T_{n+1}(x) - T_n(y) = 0$;
parametrization: $\gamma_4(t) = [\cos(nt), \cos((n+1)t)]$, $0 \le t \le \pi$;

- $n$ even, $n = 2m$,

$$\begin{cases} A^4_{\mathrm{even}} = \{(z^{n+1}_{2i}, z^n_{2j}),\ 0 \le i \le m,\ 0 \le j \le m\} \\ B^4_{\mathrm{even}} = \{(z^{n+1}_{2i+1}, z^n_{2j+1}),\ 0 \le i \le m,\ 0 \le j \le m-1\}. \end{cases} \tag{7a}$$

- $n$ odd, $n = 2m+1$,

$$\begin{cases} A^4_{\mathrm{odd}} = \{(z^{n+1}_{2i}, z^n_{2j}),\ 0 \le i \le m+1,\ 0 \le j \le m\} \\ B^4_{\mathrm{odd}} = \{(z^{n+1}_{2i+1}, z^n_{2j+1}),\ 0 \le i \le m,\ 0 \le j \le m\}. \end{cases} \tag{7b}$$

- Lagrange interpolant:

$$\mathcal{L}_{\mathrm{Pad}_n^4} f(\boldsymbol{x}) = \sum_{\boldsymbol{\xi} \in \mathrm{Pad}_n^4} f(\boldsymbol{\xi}) w_{\boldsymbol{\xi}} \left( K_n(\boldsymbol{x}, \boldsymbol{\xi}) - T_n(x_2) T_n(\xi_2) \right). \tag{7c}$$

**Remark 1** (*Convergence Rate*). The Lebesgue constant of interpolation at the Padua points has optimal order of growth $\Lambda_{\mathrm{Pad}_n^s} = \|\mathcal{L}_{\mathrm{Pad}_n^s}\| = \mathcal{O}(\log^2(n))$, $s = 1, 2, 3, 4$, as has been rigorously proved in [4,12,16]. In view of the multivariate extension of Jackson's theorem (cf., e.g., [1] and references therein), we have that for $f \in C^p([-1,1]^2)$, $0 < p < \infty$,

$$\|f - \mathcal{L}_{\mathrm{Pad}_n^s} f\|_\infty \le \left(1 + \Lambda_{\mathrm{Pad}_n^s}\right) E_n(f) \le c(f; p) \log^2(n)\, n^{-p}, \tag{8}$$

where $c$ is a suitable constant (with $n$), dependent on $f$ and $p$.

## 3. Implementation

In view of the explicit representations above, the computational core of interpolation at the Padua points is given by an efficient treatment of the reproducing kernel. In [17,18], an elegant compact trigonometric formula for such a kernel was given, which has been key for bounding rigorously the Lebesgue constant in [4].

Unfortunately, such a formula turns out to be severely ill-conditioned, and has to be stabilized. This has been done in [6] in the context of interpolation at the Xu points (cf. [18]). Applied in the present framework to the interpolation formulas (4c)–(7c), this method leads to a pointwise evaluation cost for the interpolant at the Padua points of the order of $24 c_{\sin} N \approx 12 c_{\sin} n^2$ flops for degrees $n$ up to the hundreds, $c_{\sin}$ denoting the average cost of the sine function.

On the other hand, in view of (3) there is another natural way of writing and computing the interpolant at the Padua points, i.e. via its representation in the basis $\{\hat{T}_j(x_1)\hat{T}_{k-j}(x_2)\}$, $0 \le j \le k \le n$, which is orthonormal with respect to the product Chebyshev measure. In fact, considering for simplicity only the family $\mathrm{Pad}_n^1$, in view of (3) and (4c) we have that

$$\mathcal{L}_{\mathrm{Pad}_n^1} f(\boldsymbol{x}) = \sum_{k=0}^{n} \sum_{j=0}^{k} c_{j,k-j} \hat{T}_j(x_1) \hat{T}_{k-j}(x_2), \tag{9}$$

where

$$c_{j,k-j} = c_{j,k-j}(f) = \sum_{\boldsymbol{\xi} \in \mathrm{Pad}_n^1} f(\boldsymbol{\xi}) w_{\boldsymbol{\xi}} \hat{T}_j(\xi_1) \hat{T}_{k-j}(\xi_2), \quad (k, j) \neq (n, n),$$

$$c_{n,0} = c_{n,0}(f) = \frac{1}{2} \sum_{\boldsymbol{\xi} \in \mathrm{Pad}_n^1} f(\boldsymbol{\xi}) w_{\boldsymbol{\xi}} \hat{T}_n(\xi_1). \tag{10}$$

Clearly, for $f \in \Pi_n^2$ these are exactly the Fourier(–Chebyshev) coefficients, i.e.

$$c_{j,k-j}(f) = \varphi_{j,k-j}(f) = \frac{1}{\pi^2} \int_{[-1,1]^2} f(x_1, x_2) \hat{T}_j(x_1) \hat{T}_{k-j}(x_2) \frac{\mathrm{d}x_1 \mathrm{d}x_2}{\sqrt{1 - x_1^2}\sqrt{1 - x_2^2}},$$

$$\forall f \in \Pi_n^2, \quad \text{and} \quad \forall (k, j),\ 0 \le j \le k \le n. \tag{11}$$

Concerning the other families of Padua points, the construction is completely analogous. We only observe that the coefficient to be halved is again $c_{n,0}$ for the third family $\mathrm{Pad}_n^3$, while it is $c_{0,n}$ for the second and the fourth, $\mathrm{Pad}_n^2$ and $\mathrm{Pad}_n^4$.

The Fourier–Chebyshev representation (9) and (10) is more suitable for computation than that discussed above, which relies on the stabilized compact formula for the reproducing kernel. Moreover, it admits a natural matrix formulation, which allows one to design a simple and effective Matlab implementation (since Matlab bottlenecks like recurrences and iteration loops are avoided; cf. [8]), or to use conveniently machine-specific optimized BLAS (Basic Linear Algebra Subprograms) even in a Fortran (or $C$) implementation (cf. [9]). Another useful feature of the Fourier–Chebyshev representation is the possibility of estimating a posteriori the interpolation error by the size of some coefficients, as we shall see below.

### 3.1. Matrix formulation

For $s = 1, 2, 3, 4$, consider the matrices

$$D = D(\mathrm{Pad}_n^s, f) = \mathrm{diag}\big([w_{\boldsymbol{\xi}} f(\boldsymbol{\xi}), \boldsymbol{\xi} \in \mathrm{Pad}_n^s]\big) \in \mathbb{R}^{N \times N}, \tag{12a}$$

$$\Theta_i = \mathcal{T}_i(\mathrm{Pad}_n^s) = \underbrace{\begin{bmatrix} \cdots & \hat{T}_0(\xi_i) & \cdots \\ \vdots & \vdots & \vdots \\ \cdots & \hat{T}_n(\xi_i) & \cdots \end{bmatrix}}_{\boldsymbol{\xi} \in \mathrm{Pad}_n^s} \in \mathbb{R}^{(n+1) \times N}, \quad i = 1, 2, \tag{12b}$$

$$C_0 = C_0(\mathrm{Pad}_n^s, f) = \begin{bmatrix} c_{0,0} & c_{0,1} & \cdots & \cdots & c_{0,n} \\ c_{1,0} & c_{1,1} & \cdots & c_{1,n-1} & 0 \\ \vdots & \vdots & \iddots & \iddots & \vdots \\ c_{n-1,0} & c_{n-1,1} & 0 & \cdots & 0 \\ c_{n,0} & 0 & \cdots & 0 & 0 \end{bmatrix} \in \mathbb{R}^{(n+1) \times (n+1)}, \tag{12c}$$

the latter being the upper left triangular part of

$$B = B(\mathrm{Pad}_n^s, f) = \Theta_1 D \Theta_2^t \in \mathbb{R}^{(n+1)\times(n+1)} \tag{13}$$

with the modification $c_{0,n} = b_{0,n}$, $c_{n,0} = b_{n,0}/2$ for the first and the third family, and $c_{0,n} = b_{0,n}/2$, $c_{n,0} = b_{n,0}$ for the second and the fourth family. Then it is easy to see that, setting

$$\tau_i(\boldsymbol{x}) = [\hat{T}_0(x_i), \ldots, \hat{T}_n(x_i)]^t \text{ (column vector)}, \quad i = 1, 2, \tag{14}$$

the interpolant at the Padua points can be computed in the form (9) as

$$\mathcal{L}_{\mathrm{Pad}_n^s} f(\boldsymbol{x}) = (\tau_1(\boldsymbol{x}))^t C_0 \tau_2(\boldsymbol{x}). \tag{15}$$

Even with an array of $M$ target points, say $X$, the interpolant can be computed through matrix operations, avoiding iteration loops completely (which is essential in Matlab). In fact, setting

$$\mathcal{T}_i(X) = \underbrace{\begin{bmatrix} \cdots & \hat{T}_0(x_i) & \cdots \\ \vdots & \vdots & \vdots \\ \cdots & \hat{T}_n(x_i) & \cdots \end{bmatrix}}_{\boldsymbol{x}\in X} \in \mathbb{R}^{(n+1)\times M}, \quad i = 1, 2,$$

we have that

$$\mathcal{L}_{\mathrm{Pad}_n^s} f(X) = [\mathcal{L}_{\mathrm{Pad}_n^s} f(\boldsymbol{x})]_{\boldsymbol{x}\in X} = \text{diagonal of } (\mathcal{T}_1(X))^t C_0 \mathcal{T}_2(X). \tag{16}$$

**Remark 2** (*Beyond the Square*). Clearly, we can immediately extend interpolation at the Padua points to a function defined on a generic *rectangle* $R(\boldsymbol{a}, \boldsymbol{b}) = [a_1, b_1] \times [a_2, b_2]$, via the affine mapping $\boldsymbol{\sigma}: [-1, 1]^2 \to R(\boldsymbol{a}, \boldsymbol{b})$, $\sigma_i(t_1, t_2) = (b_i - a_i)t_i/2 + (b_i + a_i)/2$, $i = 1, 2$. Indeed, the interpolation formula becomes simply

$$\mathcal{L}_{\mathrm{Pad}_n^s} f(\boldsymbol{x}) = (\tau_1(\boldsymbol{\sigma}^{-1}(\boldsymbol{x})))^t C_0(\mathrm{Pad}_n^s, f \circ \boldsymbol{\sigma}) \tau_2(\boldsymbol{\sigma}^{-1}(\boldsymbol{x})). \tag{17}$$

It is also possible to construct non-polynomial interpolation formulas at Padua-like points on bivariate domains with different geometric structures (such as triangles, generalized rectangles, generalized sectors) by means of suitable (as smooth as possible) surjective transformations of the square (cf. [5]). For example, for the *triangle* $T(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$ with vertices $\boldsymbol{u} = (u_1, u_2)$, $\boldsymbol{v} = (v_1, v_2)$ and $\boldsymbol{w} = (w_1, w_2)$ it is possible to use the Proriol (also known as Duffy) map $\boldsymbol{\sigma}: [-1, 1]^2 \to T(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$, $\sigma_i(t_1, t_2) = (v_i - u_i)(1+t_1)(1-t_2)/4 + (w_i - u_i)(1+t_2)/2 + u_i$, $i = 1, 2$, and for the *ellipse* $E(\boldsymbol{c}, \alpha, \beta)$ centered at $\boldsymbol{c} = (c_1, c_2)$ with $x_1$-semiaxis $\alpha$ and $x_2$-semiaxis $\beta$, it is convenient to use the starlike polar map $\boldsymbol{\sigma}: [-1, 1]^2 \to E(\boldsymbol{c}, \alpha, \beta)$, $\sigma_1(t_1, t_2) = c_1 - \alpha t_2 \sin(\pi t_1/2)$, $\sigma_2(t_1, t_2) = c_2 + \beta t_2 \cos(\pi t_1/2)$. Then we have an interpolation formula like (17), where, since $\boldsymbol{\sigma}$ is surjective but non-invertible, we denote by $\boldsymbol{\sigma}^{-1}(\boldsymbol{x})$ a suitable choice (when necessary) in the inverse image of $\boldsymbol{x}$ through $\boldsymbol{\sigma}$ (e.g., $\boldsymbol{\sigma}^{-1}(\boldsymbol{w}) = (0, 1)$ for the triangle, $\boldsymbol{\sigma}^{-1}(\boldsymbol{c}) = (0, 0)$ for the ellipse). Observe that the starlike polar map for the ellipse distributes the interpolation points more symmetrically with respect to the usual polar coordinates, and shows a better interpolation error (cf. [5]).

**Remark 3** (*Computational Cost*). First, we observe that a simple analysis of the matrix-like interpolation algorithm gives the following complexity estimates for construction (excluding evaluation of the function $f$ at the Padua points), and evaluation at a set of $M$ target points:

- *construction*: cost of (12b) + cost of (12c) $\approx 2c_T nN + 2(n+1)^2 N$ flops;
- *evaluation*: $M\times$ (cost of (14) + cost of (15)) $\approx M(2c_T n + 4(n+1)^2)$ flops,

where $N$ is the number of Padua points (cf. (2)), and $c_T$ denotes the average evaluation cost of a single Chebyshev polynomial via its trigonometric representation (suitable in Matlab), or via the three-term recurrence (suitable in Fortran: here, $c_T \approx 2$). This complexity estimate shows that on a large number of evaluation points, say $M \gg N$, the present implementation is more convenient than that based on the stabilized Xu formula for the reproducing kernel (cf. [6]), whose cost is of the order of $24c_{\sin}NM$ flops, since $2c_T n + 4(n+1)^2 \ll 24c_{\sin}N = 12c_{\sin}(n+1)(n+2)$ already for relatively small values of $n$.

**Remark 4** (*Error Estimate*). An important feature in the practical use of polynomial interpolation is the possibility of having a reliable and if possible a posteriori estimate of the interpolation error. To this end, representation of the interpolant in a suitable orthogonal basis is useful. This is the reason, for example, why in their recent paper on the univariate Chebfun system [2], Battles and Trefethen switch from the barycentric Lagrange form of the interpolant (suitable for computation) to its representation in the Chebyshev orthogonal basis (suitable for estimating the error). Here, the Fourier–Chebyshev representation (9)–(11) is suitable for both purposes. In fact, consider $c_{j,k-j}$ and $\varphi_{j,k-j}$ as linear functionals on $C([-1,1]^2, \|\cdot\|_\infty)$. Then, in view of (11) and the fact that the $\{w_\xi\}$ are weights of a cubature formula exact on constants, denoting by $p_n^*$ the best uniform polynomial approximation of degree $n$ to $f \in C([-1,1]^2)$,

$$\begin{aligned} |c_{j,k-j}(f) - \varphi_{j,k-j}(f)| &\le (\|c_{j,k-j}\| + \|\varphi_{j,k-j}\|)\, \|f - p_n^*\|_\infty \\ &\le 2\left(\sum_{\xi \in \mathrm{Pad}_n^s} w_\xi + \frac{1}{\pi^2}\int_{[-1,1]^2} \frac{\mathrm{d}x_1\mathrm{d}x_2}{\sqrt{1-x_1^2}\sqrt{1-x_2^2}}\right)\|f - p_n^*\|_\infty = 4E_n(f), \end{aligned} \tag{18}$$

for every $(k, j)$, $0 \le j \le k \le n$. Moreover, indicating by $\mathcal{S}_n f$ the truncated Fourier–Chebyshev expansion of $f$, $\mathcal{S}_n f(\boldsymbol{x}) = \sum_{k=0}^n \sum_{j=0}^k \varphi_{j,k-j}(f)\hat{T}_j(x_1)\hat{T}_{k-j}(x_2)$, and observing that $\mathcal{L}_{\mathrm{Pad}_n^s}$ and $\mathcal{S}_n$ are both projection operators on $\Pi_n^2$, we have

$$\|\mathcal{L}_{\mathrm{Pad}_n^s} f - \mathcal{S}_n f\|_\infty \le (\|\mathcal{L}_{\mathrm{Pad}_n^s}\| + \|\mathcal{S}_n\|)\|f - p_n^*\|_\infty = \mathcal{O}\left(\log^2(n) E_n(f)\right). \tag{19}$$

The growth estimate $\|\mathcal{L}_{\mathrm{Pad}_n^s}\| = \mathcal{O}(\log^2(n))$ has been proved in [4], whereas the fact that $\|\mathcal{S}_n\| = \mathcal{O}(\log^2(n))$ is within the much more general results of [16]. From the inequality (19) we immediately get

$$\|f - \mathcal{L}_{\mathrm{Pad}_n^s} f\|_\infty \le a_n,\ a_n = \|f - \mathcal{S}_n f\|_\infty + \mathcal{O}\left(\log^2(n)\, E_n(f)\right).$$

Thus, we can conjecture the following a posteriori error estimate (which is substantially validated by the numerical results):

$$\begin{aligned} \|f - \mathcal{L}_{\mathrm{Pad}_n^s} f\|_\infty \le a_n \approx \|f - \mathcal{S}_n f\|_\infty &\le 2\sum_{k=n+1}^{\infty}\sum_{j=0}^{k} \left|\varphi_{j,k-j}(f)\right| \\ &\approx 2\sum_{k=n-2}^{n}\sum_{j=0}^{k} \left|\varphi_{j,k-j}(f)\right| \approx 2\sum_{k=n-2}^{n}\sum_{j=0}^{k} \left|c_{j,k-j}(f)\right|. \end{aligned} \tag{20}$$

The passage from the first to the second row in (20), though empirical, is reminiscent of popular error estimates for one-dimensional Chebyshev expansions, based on the size of the last two or three coefficients (cf., e.g., [2]). Here, we resort indeed to the coefficients corresponding to the last three degrees $k$, namely $k = n-2, n-1, n$. Notice that the first and the last approximation in (20) are justified by (8) and (19), and (18), respectively. The latter, in particular, seems to give an overestimate of the final error by an order of $\mathcal{O}(nE_n(f))$. In practice, however, as in the one-dimensional case it happens that (20) tends to be an overestimate for smooth functions, and an underestimate for functions of low regularity (due to fast/slow decay of the Fourier–Chebyshev coefficients). The behavior of this error estimate has been satisfactory in almost all our numerical tests (see the next section).

## 4. Numerical tests

In Table 1 we show the errors of interpolation at the first family of Padua points in the max-norm normalized to the max deviation of the function from its mean, at a sequence of degrees, $n = 10, 20, \ldots, 60$, on a well-known suite of 10 test functions used in [15]. The corresponding "true" errors have been computed on a $100 \times 100$ uniform control grid. In the table we report also (in parentheses) the a posteriori empirical error estimate given by the last term of (20), normalized as above. The tests have been done using the Fortran code in [9].

The last four functions are considered more challenging for the testing of interpolation methods at scattered points, due to their multiple features and abrupt transitions. Here, we can see that only $F_{10}$ and, much less severely,

Table 1
"True" and estimated (in parentheses) normalized errors of interpolation at the Padua points for the test set in [15]

| $n$ | $F_1$ | $F_2$ | $F_3$ | $F_4$ | $F_5$ | $F_6$ | $F_7$ | $F_8$ | $F_9$ | $F_{10}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 10 | 9E−2 | 4E−1 | 8E−3 | 4E−4 | 4E−2 | 1E−4 | 3E−1 | 1E−1 | 3E−1 | 5E−1 |
| | (2E−1) | (6E−1) | (6E−2) | (2E−2) | (2E−1) | (2E−3) | (1E+0) | (4E−1) | (1E+0) | (8E−1) |
| 20 | 7E−3 | 6E−2 | 1E−5 | 7E−10 | 6E−5 | 4E−8 | 8E−6 | 3E−3 | 7E−3 | 1E−1 |
| | (2E−2) | (8E−2) | (8E−5) | (1E−7) | (8E−4) | (4E−7) | (2E−4) | (1E−2) | (4E−2) | (6E−2) |
| 30 | 1E−4 | 1E−2 | 2E−8 | 2E−14 | 1E−8 | 2E−11 | 7E−13 | 2E−5 | 4E−5 | 6E−2 |
| | (8E−4) | (1E−2) | (1E−7) | (4E−14) | (2E−7) | (2E−10) | (2E−11) | (1E−4) | (2E−4) | (2E−2) |
| 40 | 3E−6 | 2E−3 | 2E−11 | 4E−14 | 4E−13 | 6E−14 | 4E−14 | 6E−8 | 1E−7 | 4E−2 |
| | (1E−5) | (2E−3) | (2E−10) | (1E−14) | (2E−11) | (1E−13) | (8E−15) | (6E−7) | (6E−7) | (8E−3) |
| 50 | 1E−8 | 4E−4 | 1E−13 | 6E−14 | 1E−15 | 1E−13 | 7E−14 | 5E−11 | 2E−10 | 3E−2 |
| | (8E−8) | (4E−4) | (4E−13) | (1E−14) | (1E−15) | (2E−14) | (1E−14) | (6E−10) | (1E−9) | (6E−3) |
| 60 | 4E−11 | 6E−5 | 2E−13 | 7E−14 | 1E−15 | 1E−13 | 1E−13 | 6E−14 | 2E−13 | 2E−2 |
| | (2E−10) | (6E−5) | (2E−14) | (2E−14) | (1E−15) | (2E−14) | (1E−14) | (4E−13) | (1E−12) | (4E−3) |

Table 2
CPU times (in seconds) for construction of the interpolation polynomial at the Padua points (excluding evaluation of $f$) and evaluation at $M = 10\,000$ target points, using Matlab and Fortran (with optimized BLAS) implementations

| | $n = 10$ | $n = 20$ | $n = 30$ | $n = 40$ | $n = 50$ | $n = 60$ |
|---|---|---|---|---|---|---|
| MATLAB | | | | | | |
| Constr. | 0.0025 | 0.0045 | 0.0110 | 0.0282 | 0.0556 | 0.0955 |
| Eval. | 0.1465 | 0.2523 | 0.3721 | 0.5022 | 0.6346 | 0.7676 |
| FORTRAN-oBLAS | | | | | | |
| Constr. | 0.0010 | 0.0050 | 0.0110 | 0.0200 | 0.0350 | 0.0550 |
| Eval. | 0.0050 | 0.0140 | 0.0240 | 0.0360 | 0.0510 | 0.0670 |

$F_2$ are really "difficult" for interpolation at the Padua points. In all the other cases the approximation behavior of the interpolation polynomial is quite satisfactory. It is interesting to observe that with the oscillating function $F_7(x_1, x_2) = 2\cos(10x_1)\sin(10x_2) + \sin(10x_1x_2)$, the error starts decaying rapidly as soon as the degree $n$ allows us to recover the oscillations. On the other hand, the troubles with $F_{10}$ are natural, since it has a gradient discontinuity in the center of the square, whereas the Padua points cluster at the boundary. As for the empirical error estimates, we can see that they tend to overestimate (at least far from machine precision), except for $F_{10}$, where they underestimate the true errors. In general, we can consider the behavior of the (normalized) a posteriori estimate (20) satisfactory. We stress also that our implementation of interpolation at the Padua points is very stable. Indeed, we could interpolate at much higher degrees without drawbacks. For example, we can take $n = 300$ ($N = 45\,451$ Padua points), obtaining an error of 9E−12 for the test function $F_2$.

In Table 2 we have reported the CPU times corresponding to our implementations in Matlab and Fortran, with optimized BLAS (oBLAS for short) libraries. The tests have been made on an AMD Athlon 2800 + processor machine with 2 Gb RAM. The resulting interpolation method is very fast. Notice that, due to optimized linear algebra, in both implementations the times scale differently (and more favourably) than what would be predictable from the flops counting in Remark 3.

## References

[1] T. Bagby, L. Bos, N. Levenberg, Multivariate simultaneous approximation, Constr. Approx. 18 (2002) 569–577.
[2] Z. Battles, L.N. Trefethen, An extension of MATLAB to continuous functions and operators, SIAM J. Sci. Comput. 25 (2004) 1743–1770.
[3] B. Bojanov, Y. Xu, On polynomial interpolation of two variables, J. Approx. Theory 120 (2003) 267–282.
[4] L. Bos, M. Caliari, S. De Marchi, M. Vianello, Y. Xu, Bivariate Lagrange interpolation at the Padua points: The generating curve approach, J. Approx. Theory 143 (1) (2006) 15–25.
[5] L. Bos, M. Caliari, S. De Marchi, M. Vianello, Bivariate interpolation at Xu points: Results, extensions and applications, Electron. Trans. Numer. Anal. 25 (2006) 1–16.
[6] L. Bos, M. Caliari, S. De Marchi, M. Vianello, A numerical study of the Xu polynomial interpolation formula, Computing 76 (2005) 311–324.

[7] L. Bos, S. De Marchi, M. Vianello, Y. Xu, Bivariate Lagrange interpolation at the Padua points: The ideal theory approach, Numer. Math. (2007). Available on line, DOI 10.1007/s00211-007-0112-z.
[8] M. Caliari, S. De Marchi, R. Montagna, M. Vianello, XuPad2D: A Matlab code for hyperinterpolation/interpolation at Xu/Padua points on rectangles, available at http://www.math.unipd.it/~marcov/software.html.
[9] M. Caliari, S. De Marchi, M. Vianello, Padua2D: A Fortran code for bivariate Lagrange interpolation at Padua points, available at http://www.math.unipd.it/~marcov/software.html.
[10] M. Caliari, S. De Marchi, M. Vianello, Bivariate polynomial interpolation on the square at new nodal sets, Appl. Math. Comput. 165 (2005) 261–274.
[11] J.M. Carnicer, M. Gasca, T. Sauer, Interpolation lattices in several variables, Numer. Math. 102 (2006) 559–581.
[12] B. Della Vecchia, G. Mastroianni, P. Vertesi, Exact order of the Lebesgue constants for bivariate Lagrange interpolation at certain node systems, Studia Sci. Math. Hungar. (2007) (in press).
[13] C.F. Dunkl, Y. Xu, Orthogonal Polynomials of Several Variables, in: Encyclopedia of Mathematics and its Applications, vol. 81, Cambridge University Press, Cambridge, 2001.
[14] M. Gasca, T. Sauer, Polynomial interpolation in several variables, Adv. Comput. Math. 12 (2000) 377–410.
[15] R.J. Renka, R. Brown, Algorithm 792: Accuracy tests of ACM algorithms for interpolation of scattered data in the plane, ACM Trans. Math. Software 25 (1999) 79–93.
[16] L. Szili, P. Vertesi, On multivariate projection operators (2007) (in press).
[17] Y. Xu, Christoffel functions and Fourier series for multivariate orthogonal polynomials, J. Approx. Theory 82 (1995) 205–239.
[18] Y. Xu, Lagrange interpolation on Chebyshev points of two variables, J. Approx. Theory 87 (1996) 220–238.