Padua2D: Lagrange Interpolation at Padua Points on Bivariate Domains

MARCO CALIARI and STEFANO DE MARCHI
University of Verona
and
MARCO VIANELLO
University of Padua

We present a stable and efficient Fortran implementation of polynomial interpolation at the "Padua points" on the square $[-1,1]^2$. These points are unisolvent and their Lebesgue constant has minimal order of growth (log square of the degree). The algorithm is based on the representation of the Lagrange interpolation formula in a suitable orthogonal basis, and takes advantage of a new matrix formulation together with the machine-specific optimized BLAS subroutine `DGEMM` for the matrix-matrix product. Extension to interpolation on rectangles, triangles and ellipses is also described.

Categories and Subject Descriptors: D.3.2 [**Programming Languages**]: Languages Classifications—*Fortran 77*; G.1.1 [**Numerical Analysis**]: Interpolation; G.1.2 [**Numerical Analysis**]: Approximation; G.4 [**Mathematics of Computing**]: Mathematical Software

General Terms: Algorithms

Additional Key Words and Phrases: Bivariate Lagrange interpolation, Padua points, bivariate Chebyshev orthogonal basis

1. INTRODUCTION

The problem of choosing "good" nodes on a given compact set is a central one in multivariate polynomial interpolation. Besides unisolvence, which is by no means an easy problem, for practical purposes one needs slow growth of the Lebesgue constant and computational efficiency; see, e.g., [Bojanov and Xu 2003; Carnicer et al. 2006; de Boor et al. 2000; Gasca and Sauer 2000; Sauer 1995; Xu 1996] and references therein.

In [Caliari et al. 2005] a new set of points for polynomial interpolation on the square $[-1,1]^2$, called "Padua points", was introduced and experimentally studied. In [Bos et al. 2006b; Bos et al. 2007] it has been proved that they are unisolvent in the full polynomial space Π_n^2, n the polynomial degree, and that they give the first

Authors' addresses: M. Caliari and S. De Marchi, Department of Computer Science, University of Verona, Strada Le Grazie 15, 37134 Verona (Italy); email: marco.caliari@univr.it and stefano.demarchi@univr.it; M. Vianello, Department of Pure and Applied Mathematics, University of Padua, Via Trieste 63, 35121 Padova (Italy); email: marcov@math.unipd.it.

known example of non tensor-product optimal interpolation in two variables, since their Lebesgue constant has minimal order of growth of $\mathcal{O}((\log n)^2)$. Moreover, an explicit representation of their fundamental Lagrange polynomials was given, which is based on the reproducing kernel of the space $\Pi_n^2([-1,1]^2)$ equipped with the bivariate Chebyshev inner product (cf. [Dunkl and Xu 2001; Reimer 2003]).

In this paper we present a stable and efficient Fortran implementation of the Lagrange interpolation formula at the Padua points, with cost $\mathcal{O}(n^3)$ flops for the evaluation (once and for all) of the coefficients of the interpolation polynomial represented in the Chebyshev orthonormal basis, plus an additional cost of $\mathcal{O}(n^2)$ flops for the evaluation at each target point. Remarkable speedups with respect to the existing algorithm used in [Caliari et al. 2007] are obtained by a new matrix formulation. The effect of machine-specific optimized versus standard BLAS libraries is also investigated.

2. PADUA POINTS

There are four families of Padua points (cf. [Caliari et al. 2007]), which correspond to successive 90 degrees rotations (clockwise for degree n even and counterclockwise for n odd) of the square $[-1,1]^2$. Here we describe one of them (also termed the "first family" of Padua points) as well as the associated Lagrange interpolation formula. Interpolation for the other families can be easily obtained by composition with the corresponding rotation, as described in Sec. 3.4.

The $N = (n+1)(n+2)/2 = \dim\left(\Pi_n^2\right)$ Padua points corresponding to degree n are the set of points

$$\mathrm{Pad}_n := \{\boldsymbol{\xi} = (\xi_1, \xi_2)\} := \left\{ \gamma\left(\frac{k\pi}{n(n+1)}\right), \quad k = 0, \ldots, n(n+1) \right\} , \qquad (1)$$

where $\gamma(t)$ is their "generating curve" (cf. [Bos et al. 2006b])

$$\gamma(t) := (-\cos((n+1)t), -\cos(nt)), \quad t \in [0, \pi] . \qquad (2)$$

Notice that two of the points are consecutive vertices of the square, $2n-1$ other points lie on the edges of the square, and the remaining (interior) points are double points corresponding to self-intersections of the generating curve (see Fig. 1). Clearly, such structure remains invariant for the other three families up to the corresponding rotation.

Denote by C_{n+1} the set of the $n+1$ Chebyshev–Gauss–Lobatto points

$$C_{n+1} := \{z_j^n = \cos(j\pi/n),\ j = 0, \ldots, n\} . \qquad (3)$$

The Padua points (for n even) were first introduced in [Caliari et al. 2005, formula (9)] (in that formula there is a misprint, $n-1$ has to be replaced by $n+1$). For the sake of clarity, we give again that definition:

$$\mathrm{Pad}_n := \{\boldsymbol{\xi} = (\mu_j, \eta_k),\ 0 \le j \le n;\ 0 \le k \le n/2\}$$

where

$$\mu_j = z_j^n, \quad \eta_k = \begin{cases} z_{2k}^{n+1} & j \text{ odd} \\ z_{2k+1}^{n+1} & j \text{ even} \end{cases}$$

It is clear that $\mu_j \in C_{n+1}$, $\eta_k \in C_{n+2}$, and denoting by C_{n+1}^{even}, C_{n+1}^{odd} the restrictions of C_{n+1} to even and odd indexes, then

$$\text{Pad}_n = (C_{n+1}^{\text{odd}} \times C_{n+2}^{\text{even}}) \cup (C_{n+1}^{\text{even}} \times C_{n+2}^{\text{odd}}) \subset C_{n+1} \times C_{n+2} \ ,$$

which is valid also for n odd.

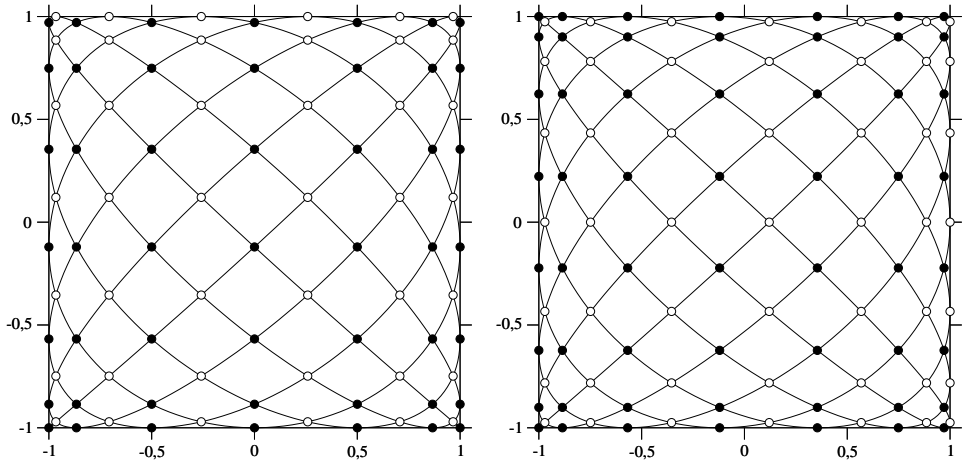

Fig. 1. The Padua points with their generating curve for $n = 12$ (left, 91 points) and $n = 13$ (right, 105 points), also as union of two Chebyshev-like grids (open bullets and filled bullets).

The fundamental Lagrange polynomials of the Padua points are

$$L_{\boldsymbol{\xi}}(\boldsymbol{x}) = w_{\boldsymbol{\xi}}(K_n(\boldsymbol{\xi}, \boldsymbol{x}) - T_n(\xi_1)T_n(x_1)) \ , \quad L_{\boldsymbol{\xi}}(\boldsymbol{\eta}) = \delta_{\boldsymbol{\xi}\boldsymbol{\eta}} \ , \quad \boldsymbol{\xi}, \boldsymbol{\eta} \in \text{Pad}_n \ , \tag{4}$$

where $K_n(\boldsymbol{x}, \boldsymbol{y})$, $\boldsymbol{x} = (x_1, x_2)$ and $\boldsymbol{y} = (y_1, y_2)$, is the reproducing kernel of the space $\Pi_n^2([-1, 1]^2)$ equipped with the inner product

$$\langle f, g \rangle := \frac{1}{\pi^2} \int_{[-1,1]^2} f(x_1, x_2) g(x_1, x_2) \frac{\mathrm{d}x_1}{\sqrt{1 - x_1^2}} \frac{\mathrm{d}x_2}{\sqrt{1 - x_2^2}} \ , \tag{5}$$

that is

$$K_n(\boldsymbol{x}, \boldsymbol{y}) = \sum_{k=0}^{n} \sum_{j=0}^{k} \hat{T}_j(x_1) \hat{T}_{k-j}(x_2) \hat{T}_j(y_1) \hat{T}_{k-j}(y_2) \ . \tag{6}$$

Here $\hat{T}_j$ denotes the normalized Chebyshev polynomial of degree j, i.e. $\hat{T}_0 = T_0 \equiv 1$, $\hat{T}_p = \sqrt{2} T_p$, $T_p(\cdot) = \cos(p \arccos(\cdot))$. Moreover, the weights $w_{\boldsymbol{\xi}}$ are

$$w_{\boldsymbol{\xi}} := \frac{1}{n(n+1)} \cdot \begin{cases} 1/2 & \text{if } \boldsymbol{\xi} \text{ is a vertex point} \\ 1 & \text{if } \boldsymbol{\xi} \text{ is an edge point} \\ 2 & \text{if } \boldsymbol{\xi} \text{ is an interior point} \end{cases} \tag{7}$$

We notice that the $\{w_{\boldsymbol{\xi}}\}$ are indeed weights of a cubature formula for the product Chebyshev measure, which is exact on "almost all" polynomials in $\Pi_{2n}^2([-1, 1]^2)$, namely on all polynomials orthogonal to $T_{2n}(x_1)$. Such a cubature formula stems

from quadrature along the generating curve and is the key to obtaining the fundamental Lagrange polynomials (4); cf. [Bos et al. 2006b]. A more abstract approach to the construction of the Lagrange polynomials, based on the theory of polynomial ideals and multivariate orthogonal polynomials, can be found in [Bos et al. 2007].

The polynomial interpolation formula can be written, in view of (4) and (6), in the bivariate Chebyshev orthonormal basis as

$$\begin{aligned}\mathcal{L}_n f(\boldsymbol{x}) &= \sum_{\boldsymbol{\xi}\in \mathrm{Pad}_n} f(\boldsymbol{\xi}) w_{\boldsymbol{\xi}} \left(K_n(\boldsymbol{\xi},\boldsymbol{x}) - T_n(\xi_1)T_n(x_1)\right) \\ &= \sum_{k=0}^{n}\sum_{j=0}^{k} c_{j,k-j}\hat{T}_j(x_1)\hat{T}_{k-j}(x_2) - \sum_{\boldsymbol{\xi}\in \mathrm{Pad}_n} f(\boldsymbol{\xi}) w_{\boldsymbol{\xi}} \hat{T}_n(\xi_1)\frac{\hat{T}_0(\xi_2)}{\sqrt{2}}\hat{T}_n(x_1)\frac{\hat{T}_0(x_2)}{\sqrt{2}} \\ &= \sum_{k=0}^{n}\sum_{j=0}^{k} c_{j,k-j}\hat{T}_j(x_1)\hat{T}_{k-j}(x_2) - \frac{c_{n,0}}{2}\hat{T}_n(x_1)\hat{T}_0(x_2)\ ,\end{aligned} \tag{8}$$

where the coefficients

$$c_{j,k-j} := \sum_{\boldsymbol{\xi}\in \mathrm{Pad}_n} f(\boldsymbol{\xi}) w_{\boldsymbol{\xi}} \hat{T}_j(\xi_1)\hat{T}_{k-j}(\xi_2), \quad 0\le j\le k\le n \tag{9}$$

can be computed once and for all.

3. THE INTERPOLATION ALGORITHM

The following definition will be useful below. Given a vector $S=(s_1,\dots,s_m)\in[-1,1]^m$, define the rectangular Chebyshev matrix

$$\mathbb{T}(S) := \begin{pmatrix} \hat{T}_0(s_1) & \cdots & \hat{T}_0(s_m) \\ \vdots & \cdots & \vdots \\ \hat{T}_n(s_1) & \cdots & \hat{T}_n(s_m) \end{pmatrix} \in \mathbb{R}^{(n+1)\times m}\ . \tag{10}$$

3.1 Old algorithm

In the recent paper [Caliari et al. 2007] the interpolation formula (8)–(9) has been implemented by the following factorization of the coefficient matrix, which was already used for hyperinterpolation at the Morrow–Patterson–Xu points in [Caliari et al. 2006b]. Considering the vectors of the one-dimensional projections of the Padua points and the corresponding Chebyshev matrices

$$\mathbb{T}_i := \mathbb{T}(\mathrm{Pad}_n^i) \in \mathbb{R}^{(n+1)\times N}\ , \quad \mathrm{Pad}_n^i := (\xi_i)_{\boldsymbol{\xi}\in\mathrm{Pad}_n}\ , \quad i=1,2\ , \tag{11}$$

and the diagonal matrix

$$\mathbb{D}(f) := \mathrm{diag}\left((w_{\boldsymbol{\xi}} f(\boldsymbol{\xi}),\ \boldsymbol{\xi}\in\mathrm{Pad}_n)\right) \in \mathbb{R}^{N\times N}\ , \tag{12}$$

following [Caliari et al. 2006b], it is easy to prove that the coefficients $c_{j,k-j}$ in (9) are the upper-left triangular part of the matrix

$$(c_{p,q}) = \mathbb{C}(f) := \mathbb{T}_1\mathbb{D}(f)\mathbb{T}_2^{\mathrm{t}} \in \mathbb{R}^{(n+1)\times(n+1)}\ . \tag{13}$$

Indeed, we have that $c_{p,q}$ in (13) is exactly $c_{j,k-j}$ in (9) with $j=p$, $k=p+q$, $0\le j\le k\le n$. Notice that the multiplication by the diagonal matrix $\mathbb{D}(f)$ is

simply a scaling of the columns of $\mathbb{T}_1$ or of the rows of $\mathbb{T}_2$. Then, defining the Fourier–Chebyshev coefficient matrix

$$\mathbb{C}_0(f) := \begin{pmatrix} c_{0,0} & c_{0,1} & \cdots & \cdots & c_{0,n} \\ c_{1,0} & c_{1,1} & \cdots & c_{1,n-1} & 0 \\ \vdots & \vdots & \iddots & \iddots & \vdots \\ c_{n-1,0} & c_{n-1,1} & 0 & \cdots & 0 \\ c_{n,0}/2 & 0 & \cdots & 0 & 0 \end{pmatrix} \in \mathbb{R}^{(n+1)\times(n+1)} , \tag{14}$$

it is easy to see that the polynomial interpolation formula (8) can be evaluated at any $\boldsymbol{x} = (x_1, x_2) \in [-1, 1]^2$ by

$$\mathcal{L}_n f(\boldsymbol{x}) = (\mathbb{T}(x_1))^{\mathrm{t}}\, \mathbb{C}_0(f) \mathbb{T}(x_2) , \tag{15}$$

cf. (10), taking suitably into account the triangular structure of $\mathbb{C}_0(f)$ (notice that $\mathbb{T}(x_i)$, $i = 1, 2$, is a column vector).

3.2 New algorithm

A substantial improvement can be obtained by a different matrix factorization. With a little abuse of notation, we denote by $C_{n+1} = (z_0^n, \ldots, z_n^n)$ the ordered vector of the Chebyshev–Gauss–Lobatto points, too. Consider the matrices (cf. (10))

$$\mathbb{P}_1 := \mathbb{T}(C_{n+1}) \in \mathbb{R}^{(n+1)\times(n+1)}, \quad \mathbb{P}_2 := \mathbb{T}(C_{n+2}) \in \mathbb{R}^{(n+1)\times(n+2)} , \tag{16}$$

and define the $(n+1)\times(n+2)$ matrix computed correspondingly to the Chebyshev-like grid $C_{n+1} \times C_{n+2}$

$$\mathbb{G}(f) = (g_{r,s}) := \begin{cases} w_{\boldsymbol{\eta}} f(z_r^n, z_s^{n+1}) & \text{if } \boldsymbol{\eta} = (z_r^n, z_s^{n+1}) \in \mathrm{Pad}_n \\ 0 & \text{if } \boldsymbol{\eta} = (z_r^n, z_s^{n+1}) \in (C_{n+1} \times C_{n+2}) \setminus \mathrm{Pad}_n \end{cases} \tag{17}$$

Then, it is easy to check that the matrix $\mathbb{C}(f)$ in (13) can also be factorized as

$$\mathbb{C}(f) = \mathbb{P}_1 \mathbb{G}(f) \mathbb{P}_2^{\mathrm{t}} . \tag{18}$$

The evaluation of the polynomial interpolant is now performed as in (15), via the coefficient matrix $\mathbb{C}_0(f)$ in (14).

The computational cost of the coefficient matrix by (18) is $\mathcal{O}(n^3)$, whereas that of (13) is $\mathcal{O}(n^4)$. Such an improvement is due to the fact that the relevant matrices have $\mathcal{O}(n^2)$ instead of $\mathcal{O}(n^3)$ elements. The matrix formulation (15)–(18) of the interpolation problem will be the computational key for the Fortran implementation described in Sec. 4.

3.3 Error estimates

In view of the growth estimate for the Lebesgue constant proved in [Bos et al. 2006b] and of the multivariate extension of Jackson's theorem (cf., e.g., [Bagby et al. 2002] and references therein), we have that for $f \in C^p([-1, 1]^2)$, $p > 0$,

$$\|f - \mathcal{L}_n f\|_\infty \le (1 + \Lambda_n)\, E_n(f) \le \mathcal{C}(f; p)\, (1 + \Lambda_n)\, n^{-p} = \mathcal{O}\left(n^{-p} (\log n)^2\right) ,$$

where $\mathcal{C}$ is a suitable constant (with respect to n), dependent on f and p. However, this a priori estimate is essentially qualitative.

A desirable feature is the availability of a reliable and, if possible, a posteriori quantitative estimate of the error. To this purpose, we have used the fact that interpolation at Padua points is "close" to a discretized truncated Fourier–Chebyshev expansion. We then obtain the following a posteriori estimate of the interpolation error (cf. (8)–(9))

$$\|f - \mathcal{L}_n f\|_\infty \approx \|f - \mathcal{S}_n f\|_\infty \approx 2 \sum_{k=n-2}^{n} \sum_{j=0}^{k} |c_{j,k-j}| \,, \tag{19}$$

where $\mathcal{S}_n f$ is the orthogonal projection of f on $\Pi_n^2([-1,1]^2)$ with respect to the inner product (5).

The first approximation can be rigorously justified (cf. [Caliari et al. 2007]), whereas the second is somewhat "empirical", but reminiscent of popular error estimates for one-dimensional Chebyshev expansions. In practice, indeed, as in the one-dimensional case it happens that (19) tends to be an overestimate for smooth functions, and an underestimate for functions of low regularity (due to the fast/slow decay of the Fourier–Chebyshev coefficients). Application to classical test sets for interpolation has shown that the behavior of this error estimate is satisfactory (see [Caliari et al. 2007]), and thus it is provided as an output of our interpolation subroutine.

In order to give a flavour of the numerical performance of the interpolation algorithm and reliability of the error estimate (19), we report in Table I some results on the classical Franke test suite (cf. [Franke 1982]) taken from [Caliari et al. 2007]. We recall that all errors are normalized to the maximum deviation of the function from its mean, and that the "true" errors have been computed on a 100×100 uniform control grid.

degree n	F_1	F_2	F_3	F_4	F_5	F_6
20	2E-2	6E-2	1E-5	7E-10	6E-5	4E-8
	(2E-2)	(8E-2)	(8E-5)	(1E-7)	(8E-4)	(5E-7)
40	2E-6	2E-3	2E-11	8E-15	4E-13	1E-14
	(1E-5)	(2E-3)	(2E-10)	(1E-15)	(2E-11)	(1E-13)
60	2E-11	6E-5	3E-14	2E-14	8E-15	3E-14
	(3E-10)	(7E-5)	(5E-15)	(5E-15)	(1E-15)	(6E-15)

Table I. "True" and estimated (in parentheses) normalized errors of interpolation of Padua points for the classical Franke test suite.

3.4 Mapping the square

The interpolation formula (15) can be extended to: a) the other three families of Padua points and b) other domains, by means of a suitable mapping of the square. Indeed, when a smooth and surjective transformation

$$\boldsymbol{\sigma} \colon [-1,1]^2 \to \Omega \,, \quad \boldsymbol{t} \mapsto \boldsymbol{x} = \boldsymbol{\sigma}(\boldsymbol{t}) \,, \quad \Omega \subset \mathbb{R}^2 \tag{20}$$

is given, we can construct the (in general nonpolynomial) interpolation formula on Ω,

$$\mathcal{L}_n f(\boldsymbol{x}) = \mathcal{L}_n f(\boldsymbol{x}; \boldsymbol{\sigma}, \Omega) := \mathbb{T}\left(\sigma_1^{-1}(\boldsymbol{x})\right)^{\mathrm{t}} \, \mathbb{C}_0(f \circ \boldsymbol{\sigma}) \, \mathbb{T}\left(\sigma_2^{-1}(\boldsymbol{x})\right) \,. \tag{21}$$

Here we denote by $\sigma_i^{-1}(\boldsymbol{x})$, $i = 1, 2$ the components of the "inverse" transformation, which in general has to be defined by choosing a point in the inverse image of $\boldsymbol{x}$ (recall that $\boldsymbol{\sigma}$ is assumed to be surjective but not necessarily injective). Any particular choice will give a valid interpolation formula.

For example, polynomial interpolation over any nondegenerate parallelogram can be obtained by a suitable invertible affine map such as

$$\boldsymbol{\sigma}(\boldsymbol{t}) := A\boldsymbol{t} + \boldsymbol{v} \ , \quad \boldsymbol{\sigma}^{-1}(\boldsymbol{x}) = A^{-1}(\boldsymbol{x} - \boldsymbol{v}) \ , \quad A \in \mathbb{R}^{2\times 2} \ . \tag{22}$$

As a special case, interpolation over the reference square rotated by an angle θ can be immediately obtained from

$$A := \begin{pmatrix} \cos\theta & -\sin\theta \\ \\ \sin\theta & \cos\theta \end{pmatrix} \ , \quad A^{-1} = A^{\mathrm{t}} \ , \quad \boldsymbol{v} = \boldsymbol{0} \ . \tag{23}$$

In particular, Lagrange interpolation at the other three families of Padua points (say, the "second", "third" and "fourth" family), corresponds to $\theta = -\pi/2$, $-\pi$, $-3\pi/2$ for even degree n, and $\theta = \pi/2$, π, $3\pi/2$ for odd degree, respectively (here $\Omega = [-1, 1]^2$). On the other hand, interpolation of a function f defined on a rectangle such as $\Omega = R(\boldsymbol{a}, \boldsymbol{b}) := [a_1, b_1] \times [a_2, b_2]$, corresponds to

$$A := \frac{1}{2} \begin{pmatrix} b_1 - a_1 & 0 \\ \\ 0 & b_2 - a_2 \end{pmatrix} \ , \quad \boldsymbol{v} = \frac{1}{2}(\boldsymbol{b} + \boldsymbol{a}) \ . \tag{24}$$

The examples above concern transformations that are globally invertible and still generate a polynomial interpolant. But it is also possible to use tranformations into bivariate domains with different geometric structures, provided that they are surjective and smooth in order to preserve as much as possible the polynomial approximation quality. These are the cases, for example, of triangles, generalized rectangles and generalized sectors, where we finally get a nonpolynomial interpolant.

As for the case of a nondegenerate triangle $\Omega = T(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$ with vertices $\boldsymbol{u} = (u_1, u_2)$, $\boldsymbol{v} = (v_1, v_2)$ and $\boldsymbol{w} = (w_1, w_2)$, it is convenient to use the Proriol (also known as the Duffy) quadratic map

$$\boldsymbol{\sigma}(\boldsymbol{t}) := \frac{1}{4}(\boldsymbol{v} - \boldsymbol{u})(1 + t_1)(1 - t_2) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{u})(1 + t_2) + \boldsymbol{u} \ . \tag{25}$$

Notice that this is non injective, since it maps the whole upper side of the square ($t_2 = 1$) to the vertex $\boldsymbol{w}$. An "inverse" can be defined via the auxiliary map $\boldsymbol{\rho}(\boldsymbol{x})$ from $T(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$ to the unit triangle with vertices $(0, 0)$, $(1, 0)$ and $(0, 1)$

$$\rho_1(\boldsymbol{x}) := \frac{x_1(w_2 - u_2) - x_2(w_1 - u_1) + (w_1 - u_1)u_2 - (w_2 - u_2)u_1}{(v_1 - u_1)(w_2 - u_2) - (v_2 - u_2)(w_1 - u_1)}$$
$$\rho_2(\boldsymbol{x}) := \frac{x_1(v_2 - u_2) - x_2(v_1 - u_1) + (v_1 - u_1)u_2 - (v_2 - u_2)u_1}{(w_1 - u_1)(v_2 - u_2) - (w_2 - u_2)(v_1 - u_1)} \ ,$$

and by choosing a point in the inverse image of $\boldsymbol{w}$, e.g. as

$$\begin{gathered}\boldsymbol{\sigma}^{-1}\colon T(\boldsymbol{u},\boldsymbol{v},\boldsymbol{w}) \to [-1,1]^2 \\ \boldsymbol{\sigma}^{-1}(\boldsymbol{x}) := \begin{cases} (0,1) & \boldsymbol{x} = \boldsymbol{w} \ (\rho_2 = 1) \\ \left(\dfrac{2\rho_1}{1-\rho_2} - 1, 2\rho_2 - 1\right) & \boldsymbol{x} \neq \boldsymbol{w} \end{cases}\end{gathered} \tag{26}$$

The resulting interpolant (21) at the Padua points mapped into the triangle becomes rational.

As an example of the use of polar-like coordinates, consider an ellipse $\Omega = E(\boldsymbol{c},\alpha,\beta)$ centered in $\boldsymbol{c} = (c_1,c_2)$ and with x_1-semiaxis α and x_2-semiaxis β. It is convenient (cf. [Bos et al. 2006a]) to define the starlike-polar map

$$\sigma_1(\boldsymbol{t}) := c_1 - \alpha t_2 \sin\left(\frac{\pi}{2} t_1\right) \ , \quad \sigma_2(\boldsymbol{t}) := c_2 + \beta t_2 \cos\left(\frac{\pi}{2} t_1\right) \ , \tag{27}$$

which distributes the interpolation points in a more symmetric way with respect to standard polar coordinates (that cluster points near the right horizontal semiaxis). The map is not globally invertible, since the whole segment $t_2 = 0$ is mapped to the ellipse center $\boldsymbol{c}$, and $\boldsymbol{\sigma}(1,t_2) = \boldsymbol{\sigma}(-1,-t_2) = (c_1 - \alpha t_2, c_2)$ for every $t_2 \in [-1,1]$. An inverse can be defined via the angle

$$\theta(\boldsymbol{x}) := \arctan\left(\frac{\beta}{\alpha}\frac{c_1 - x_1}{x_2 - c_2}\right) \ , \quad \boldsymbol{x} \neq \boldsymbol{c} \ , \tag{28}$$

and by choosing a point in the relevant inverse images, e.g. as

$$\begin{gathered}\boldsymbol{\sigma}^{-1}\colon E(\boldsymbol{c},\alpha,\beta) \to [-1,1]^2 \\ \boldsymbol{\sigma}^{-1}(\boldsymbol{x}) := \begin{cases} \left(1, \dfrac{c_1 - x_1}{\alpha}\right) & x_2 = c_2 \\ \left(\dfrac{2}{\pi}\,\theta, \dfrac{x_2 - c_2}{\beta\cos\theta}\right) & x_2 \neq c_2 \end{cases}\end{gathered} \tag{29}$$

From the identity $1/\cos^2\theta = 1 + \tan^2\theta$, we can rewrite the last row of (29) in the numerically more stable form

$$\begin{aligned} \sigma_1^{-1}(\boldsymbol{x}) &= \frac{2}{\pi}\arctan\left(\frac{\beta}{\alpha}\frac{c_1 - x_1}{x_2 - c_2}\right) \ , \\ \sigma_2^{-1}(\boldsymbol{x}) &= \operatorname{sign}(x_2 - c_2)\,\frac{\sqrt{\beta^2(x_1 - c_1)^2 + \alpha^2(x_2 - c_2)^2}}{\alpha\beta} \ . \end{aligned} \tag{30}$$

4. CODE

The software, which implements interpolation at the Padua points by the algorithm described in Sec. 3.2, is written in ANSI Fortran 77 and uses double precision. It depends on the BLAS routines `DDOT`, `DGEMM` (for the matrix-matrix product (18)), `DSCAL`, `LSAME` and `XERBLA`. Interpolation at degree n, i.e. by $N = (n+1)(n+2)/2$ Padua points, has the following storage requirements: four N-dimensional arrays, `PD1`, `PD2`, `FPD` and `WPD` containing the Padua points, the values of the function at the Padua points and the weights respectively; two auxiliary $(n+2)\times(n+2)$ arrays `RAUX1` and `RAUX2` to compute the matrix products in (18); an $(n+1)\times(n+1)$ array

`C0` for the coefficient matrix (14), for an overall allocation of about $5n^2$ floating point numbers. The user-callable subprograms are the following:

`PDPTS` Subroutine which generates the Padua points for a given degree n and the corresponding weights, see (1)–(7).

`CHEB` Subroutine which computes a Chebyshev (column) vector $\mathbb{T}(s)$, $s \in [-1, 1]$, see (10), by the three-term recurrence of the Chebyshev polynomials. It is called by `PADUA2`, see (16)–(18), and by `PD2VAL`, see (15).

`PADUA2` Subroutine which computes the coefficient matrix $\mathbb{C}_0(f)$, see Sec. 3.2.

`PD2VAL` Function which returns the value of the interpolated function $\mathcal{L}_n f(\boldsymbol{x})$ at an arbitrary target point $\boldsymbol{x}$, see (15).

The software allows to manage also interpolation on rectangles, ellipses and triangles (see Sec. 3.4). This is described in the three drivers `DRVREC, DRVTRI, DRVELL`, that carry out, as guidelines, the interpolation of the Franke test function on the unit square $[0, 1]^2$, the unit triangle $T((0, 0), (1, 0), (0, 1))$, and the disk with center $(0.5, 0.5)$ and radius 0.5, respectively. The driver `DRVREC` also shows how to interpolate at the three other families of Padua points by composition with the corresponding rotations.

4.1 Benchmarks

In this section we compare the performances of the old and new algorithms (see Sec. 3) in the computation of the coefficient matrix $\mathbb{C}_0$, both implemented by using the standard BLAS as well as the machine-specific optimized BLAS (ACML [AMD 2006] in our numerical experiments). For the relevant (double precision, general) matrix-matrix products, the BLAS subroutine `DGEMM` is used. In all cases, from Table II it is possible to see that the computation time for the evaluation of the coefficients scales as n^4 for the old algorithm (13) and as n^3 for the new algorithm (18), as expected. The speedup obtained by the new algorithm in this range of degrees (order of hundreds, that is several thousands of interpolation points) turns out to be of 2–3 orders of magnitude. On the other hand, one can appreciate that the use of optimized BLAS becomes particularly important when dealing with very high interpolation degrees.

Algorithm	Library	$n = 100$	$n = 200$	$n = 300$	$n = 400$
Old	BLAS	0.11E1	0.15E2	0.73E2	0.23E3
	ACML	0.18E0	0.14E1	0.58E1	0.16E2
New	BLAS	0.60E-2	0.48E-1	0.94E0	0.22E1
	ACML	0.30E-2	0.19E-1	0.56E-1	0.12E0

Table II. CPU times (seconds) for the computation of the coefficient matrix $\mathbb{C}_0(f)$ in (14) by the old and the new algorithms in Sec. 3.

The detailed environment of our numerical experiments is as follows:

CPU	AMD Athlon MP 2800+
OS	GNU/Linux 2.4.21
Compiler	GNU Fortran (GCC) 3.3.1

Compiler options	`-march=athlon-xp -msse -mfpmath=sse -mno-sse2 -O3`
BLAS	BLAS (Basic Linear Algebra Subprograms) (cf. [Blackford et al. 2002])
ACML	AMD Core Math Library (ACML) 3.1.0 (cf. [AMD 2006])

REFERENCES

AMD. 2006. AMD Core Math Library (ACML). Version 3.1.0. Available at `http://developer.amd.com/acml.aspx`.

Bagby, T., Bos, L., and Levenberg, N. 2002. Multivariate simultaneous approximation. *Constr. Approx. 18*, 569–577.

Blackford, L. S., Demmel, J., Dongarra, J., Duff, I., Hammarling, S., Henry, G., Heroux, M., Kaufman, L., Lumsdaine, A., Petitet, A., Pozo, R., Remington, K., and Whaley, R. C. 2002. An Updated Set of Basic Linear Algebra Subprograms (BLAS). *ACM Trans. Math. Soft. 28,* 2, 135–151. Available at `http://www.netlib.org/blas/`.

Bojanov, B. and Xu, Y. 2003. On polynomial interpolation of two variables. *J. Approx. Theory 120*, 267–282.

Bos, L., Caliari, M., De Marchi, S., and Vianello, M. 2006a. Bivariate interpolation at Xu points: results, extensions and applications. *Electron. Trans. Numer. Anal. 25*, 1–16.

Bos, L., Caliari, M., De Marchi, S., Vianello, M., and Xu, Y. 2006b Bivariate Lagrange interpolation at the Padua points: the generating curve approach. *J. Approx. Theory 143*, 15–25.

Bos, L., De Marchi, S., Vianello, M., and Xu, Y. 2007. Bivariate Lagrange interpolation at the Padua points: the ideal theory approach. *Numer. Math.*, available online 5 October 2007.

Caliari, M., De Marchi, S., and Vianello, M. 2005. Bivariate polynomial interpolation on the square at new nodal sets. *Appl. Math. Comput. 165,* 2, 261–274.

Caliari, M., De Marchi, S., and Vianello, M. 2007. Bivariate Lagrange interpolation at the Padua points: computational aspects. *J. Comput. Appl. Math.*, available online 23 October 2007.

Caliari, M., Vianello, M., De Marchi, S., and Montagna, R. 2006b. HYPER2D: a numerical code for hyperinterpolation at Xu points on rectangles. *Appl. Math. Comp. 183*, 1138–1147.

Carnicer, J. M., Gasca, M., and Sauer, T. 2006. Interpolation lattices in several variables. *Numer. Math. 102*, 559–581.

de Boor, C., Dyn, N., and Ron, A. 2000. Polynomial interpolation to data on flats in R^d. *J. Approx. Theory 105,* 2, 313–343.

Dunkl, C. F. and Xu, Y. 2001. *Orthogonal Polynomials of Several Variables*. Encyclopedia of Mathematics and its Applications, vol. 81. Cambridge University Press, Cambridge.

Franke, R. 1982. Scattered data interpolation: tests of some methods. *Math. Comp. 38*, 181–200.

Gasca, M. and Sauer, T. 2000. Polynomial interpolation in several variables. *Adv. Comput. Math. 12*, 377–410.

Reimer, M. 2003. *Multivariate Polynomial Approximation*. International Series of Numerical Mathematics, vol. 144. Birkhäuser, Basel.

Sauer, T. 1995. Computational aspects of multivariate polynomial interpolation. *Adv. Comput. Math. 3*, 219–238.

Xu, Y. 1996. Lagrange interpolation on Chebyshev points of two variables. *J. Approx. Theory 87*, 220–238.