

A numerical code for fast interpolation and cubature at the Padua points

M. Caliari¹, S. De Marchi¹, A. Sommariva² and M. Vianello²

¹ *Department of Computer Science, University of Verona*

² *Department of Pure and Applied Mathematics, University of Padua*

emails: marco.caliari@univr.it, stefano.demarchi@univr.it,
alvise@math.unipd.it, marcov@math.unipd.it

Abstract

We present a numerical code in Matlab/Octave that implements fast versions of the Lagrange interpolation formula, and of the corresponding algebraic cubature formula, at the so-called Padua points in rectangles.

Key words: Padua points, fast algorithms, Lagrange interpolation, Fast Fourier Transform, algebraic cubature

1 Introduction

In this talk we discuss an efficient implementation in Matlab/Octave of bivariate interpolation and cubature at the so-called Padua points. Such points are the first known example of optimal points for total degree polynomial interpolation in two variables, with a Lebesgue constant increasing like log square of the degree; see [1, 2, 4, 5]. Moreover, the associated algebraic cubature formula has shown a very good behavior, comparable to that of the one-dimensional Clenshaw–Curtis rule, cf. [9].

The $N = (n + 1)(n + 2)/2 = \dim(\mathbb{P}_n^2)$ Padua points, $n > 0$, are the set

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

where $\gamma(t)$ is their “generating curve” (cf. [1])

$$\gamma(t) = (-\cos((n+1)t), -\cos(nt)), \quad t \in [0, \pi] \tag{1}$$

The Padua points (for n even) were introduced for the first time in [4, formula (9)] (in that formula there is a misprint, $n - 1$ has to be replaced by $n + 1$). Denoting by C_{n+1} the set of the $n + 1$ Chebyshev–Gauss–Lobatto points

$$C_{n+1} = \{z_j^n = \cos((j-1)\pi/n), \quad j = 1, \dots, n+1\}$$

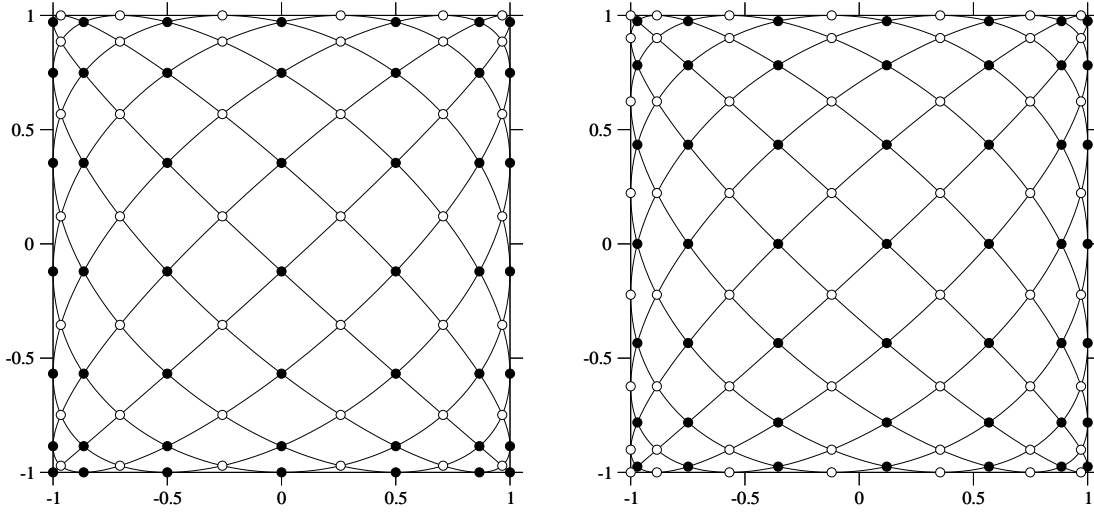


Figure 1: The Padua points with their generating curve for $n = 12$ (left, 91 points) and $n = 13$ (right, 105 points), also as union of two Chebyshev-like grids: filled bullets = $C_{n+1}^E \times C_{n+2}^O$, open bullets = $C_{n+1}^O \times C_{n+2}^E$.

and

$$\begin{aligned} C_{n+1}^E &= \{z_j^n \in C_{n+1}, j-1 \text{ even}\} \\ C_{n+1}^O &= \{z_j^n \in C_{n+1}, j-1 \text{ odd}\} \end{aligned}$$

then

$$\text{Pad}_n = (C_{n+1}^E \times C_{n+2}^O) \cup (C_{n+1}^O \times C_{n+2}^E) \subset C_{n+1} \times C_{n+2}$$

which is valid also for n odd (see Fig. 1).

The fundamental Lagrange polynomials of the Padua points are

$$L_{\boldsymbol{\xi}}(\mathbf{x}) = w_{\boldsymbol{\xi}} \left(K_n(\boldsymbol{\xi}, \mathbf{x}) - \frac{1}{2} \hat{T}_n(\xi_1) \hat{T}_n(x_1) \right) \quad (2)$$

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

$$\langle f, g \rangle = \frac{1}{\pi^2} \int_{[-1, 1]^2} f(x_1, x_2) g(x_1, x_2) \frac{dx_1}{\sqrt{1-x_1^2}} \frac{dx_2}{\sqrt{1-x_2^2}}$$

that is

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

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

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

We notice that the $\{w_{\boldsymbol{\xi}}\}$ are indeed weights of a cubature formula for the product Chebyshev measure. Such a cubature formula stems from quadrature along the generating curve and is the key to obtaining the Lagrange polynomials (2); cf. [1].

2 Fast interpolation

The polynomial interpolation formula can be written in the bivariate Chebyshev orthonormal basis as

$$\begin{aligned} \mathcal{L}_n f(\mathbf{x}) &= \sum_{\boldsymbol{\xi} \in \text{Pad}_n} f(\boldsymbol{\xi}) w_{\boldsymbol{\xi}} \left(K_n(\boldsymbol{\xi}, \mathbf{x}) - \frac{1}{2} \hat{T}_n(\xi_1) \hat{T}_n(x_1) \right) \\ &= \sum_{k=0}^n \sum_{j=0}^k c_{j,k-j} \hat{T}_j(x_1) \hat{T}_{k-j}(x_2) - \frac{1}{2} \sum_{\boldsymbol{\xi} \in \text{Pad}_n} f(\boldsymbol{\xi}) w_{\boldsymbol{\xi}} \hat{T}_n(\xi_1) \hat{T}_0(\xi_2) \hat{T}_n(x_1) \hat{T}_0(x_2) \\ &= \sum_{k=0}^n \sum_{j=0}^k c_{j,k-j} \hat{T}_j(x_1) \hat{T}_{k-j}(x_2) - \frac{c_{n,0}}{2} \hat{T}_n(x_1) \hat{T}_0(x_2) \end{aligned} \quad (3)$$

where the coefficients are defined as

$$c_{j,k-j} = \sum_{\boldsymbol{\xi} \in \text{Pad}_n} f(\boldsymbol{\xi}) w_{\boldsymbol{\xi}} \hat{T}_j(\xi_1) \hat{T}_{k-j}(\xi_2), \quad 0 \leq j \leq k \leq n \quad (4)$$

and can be computed once and for all. First we define the $(n+1) \times (n+2)$ matrix computed corresponding to the Chebyshev-like grid $C_{n+1} \times C_{n+2}$ with entries

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

Then, given a vector $S = (s_1, \dots, s_m) \in [-1, 1]^m$, we define the rectangular Chebyshev matrix

$$\mathbb{T}(S) = \begin{pmatrix} \hat{T}_0(s_1) & \cdots & \hat{T}_0(s_m) \\ \vdots & \cdots & \vdots \\ \hat{T}_n(s_1) & \cdots & \hat{T}_n(s_m) \end{pmatrix} \in \mathbb{R}^{(n+1) \times m} \quad (5)$$

Then it is easy to check (see [5]) that the coefficients $c_{j,l}$, $0 \leq j \leq n$, $0 \leq l \leq n-j$ are the entries of the upper-left triangular part of the matrix

$$\mathbb{C}(f) = \mathbb{T}(C_{n+1}) \mathbb{G}(f) (\mathbb{T}(C_{n+2}))^t \quad (6)$$

where now $C_{n+1} = (z_1^n, \dots, z_{n+1}^n)$ is the *vector* of the Chebyshev–Gauss–Lobatto points. A slightly more refined algorithm can be obtained, by exploiting the fact that the Padua points are union of two Chebyshev subgrids. Indeed, defining the two matrices

$$\begin{aligned} \mathbb{G}_1(f) &= (w_{\boldsymbol{\xi}} f(\boldsymbol{\xi}), \boldsymbol{\xi} = (z_r^n, z_s^{n+1}) \in C_{n+1}^E \times C_{n+2}^O) \\ \mathbb{G}_2(f) &= (w_{\boldsymbol{\xi}} f(\boldsymbol{\xi}), \boldsymbol{\xi} = (z_r^n, z_s^{n+1}) \in C_{n+1}^O \times C_{n+2}^E) \end{aligned}$$

then we can compute the coefficient matrix as

$$\mathbb{C}(f) = \mathbb{T}(C_{n+1}^E) \mathbb{G}_1(f) (\mathbb{T}(C_{n+2}^O))^t + \mathbb{T}(C_{n+1}^O) \mathbb{G}_2(f) (\mathbb{T}(C_{n+2}^E))^t$$

by multiplying matrices of smaller dimension than those in (6). We term this approach MM (Matrix Multiplication) in the numerical tests.

Here, we pursue an alternative computational strategy, based on the special structure of the Padua points. Indeed, the coefficients $c_{j,l}$ can be rewritten as

$$\begin{aligned} c_{j,l} &= \sum_{\boldsymbol{\xi} \in \text{Pad}_n} f(\boldsymbol{\xi}) w_{\boldsymbol{\xi}} \hat{T}_j(\xi_1) \hat{T}_l(\xi_2) = \sum_{r=0}^n \sum_{s=0}^{n+1} g_{r,s} \hat{T}_j(z_r^n) \hat{T}_l(z_s^{n+1}) \\ &= \beta_{j,l} \sum_{r=0}^n \sum_{s=0}^{n+1} g_{r,s} \cos \frac{jr\pi}{n} \cos \frac{ls\pi}{n+1} = \beta_{j,l} \sum_{s=0}^{M-1} \left(\sum_{r=0}^{N-1} g_{r,s}^0 \cos \frac{2jr\pi}{N} \right) \cos \frac{2ls\pi}{M} \end{aligned}$$

where $N = 2n$, $M = 2(n+1)$ and

$$\beta_{j,l} = \begin{cases} 1 & j = l = 0 \\ 2 & j \neq 0, l \neq 0 \\ \sqrt{2} & \text{otherwise} \end{cases} \quad g_{r,s}^0 = \begin{cases} g_{r,s} & 0 \leq r \leq n \text{ and } 0 \leq s \leq n+1 \\ 0 & r > n \text{ or } s > n+1 \end{cases}$$

Then, it is possible to recover the coefficients $c_{j,l}$ by the Discrete Fourier Transform

$$\begin{aligned} \hat{g}_{j,s} &= \text{REAL} \left(\sum_{r=0}^{N-1} g_{r,s}^0 e^{-2\pi i jr/N} \right), \quad 0 \leq j \leq n, \quad 0 \leq s \leq M-1 \\ \frac{c_{j,l}}{\beta_{j,l}} &= \hat{g}_{j,l} = \text{REAL} \left(\sum_{s=0}^{M-1} \hat{g}_{j,s} e^{-2\pi i ls/M} \right), \quad 0 \leq j \leq n, \quad 0 \leq l \leq n-j \end{aligned} \quad (7)$$

According to [5], we call $\mathbb{C}_0(f)$ the interpolation coefficients matrix

$$\mathbb{C}_0(f) = (c'_{j,l}) = \begin{pmatrix} c_{0,0} & c_{0,1} & \cdots & \cdots & c_{0,n} \\ c_{1,0} & c_{1,1} & \cdots & c_{1,n-1} & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ c_{n-1,0} & c_{n-1,1} & 0 & \cdots & 0 \\ \frac{c_{n,0}}{2} & 0 & \cdots & 0 & 0 \end{pmatrix} \in \mathbb{R}^{(n+1) \times (n+1)} \quad (8)$$

2.1 Evaluation of the interpolant

It is easy to see that the polynomial interpolation formula (3) can be evaluated at any $\mathbf{x} = (x_1, x_2) \in [-1, 1]^2$ by

$$\mathcal{L}_n f(\mathbf{x}) = (\mathbb{T}(x_1))^t \mathbb{C}_0(f) \mathbb{T}(x_2).$$

It is also possible to evaluate the polynomial interpolation formula on a set \mathbf{X} of target points, at the same time. Given the vector X_1 of the first components of a set of target points and the vector X_2 of the corresponding second components, then

$$\mathcal{L}_n f(\mathbf{X}) = \text{diag}((\mathbb{T}(X_1))^t \mathbb{C}_0(f) \mathbb{T}(X_2)) . \quad (9)$$

The result $\mathcal{L}_n f(\mathbf{X})$ is a (column) vector containing the evaluation of the interpolation polynomial at the corresponding target points.

If the target points are a Cartesian grid $\mathbf{X} = X_1 \times X_2$, then it is possible to evaluate the polynomial interpolation in a more compact form

$$\mathcal{L}_n f(\mathbf{X}) = ((\mathbb{T}(X_1))^t \mathbb{C}_0(f) \mathbb{T}(X_2))^t . \quad (10)$$

3 Fast cubature

In a recent paper [9], the interpolatory cubature formula corresponding to the Padua points has been studied. It has been termed “nontensorial Clenshaw–Curtis cubature” since it is a bivariate analogous of the classical Clenshaw–Curtis quadrature formula (cf. [6]). From the results of the previous section, we can write

$$\begin{aligned} \int_{[-1,1]^2} f(\mathbf{x}) d\mathbf{x} &\approx I_n(f) = \int_{[-1,1]^2} \mathcal{L}_n f(\mathbf{x}) d\mathbf{x} = \sum_{k=0}^n \sum_{j=0}^k c'_{j,k-j} m_{j,k-j} \\ &= \sum_{j=0}^n \sum_{l=0}^n c'_{j,l} m_{j,l} = \sum_{j \text{ even}}^n \sum_{l \text{ even}}^n c'_{j,l} m_{j,l} \end{aligned} \quad (11)$$

where the *moments* $m_{j,l}$ are

$$m_{j,l} = \int_{-1}^1 \hat{T}_j(t) dt \int_{-1}^1 \hat{T}_l(t) dt$$

with

$$\int_{-1}^1 \hat{T}_j(t) dt = \begin{cases} 2 & j = 0 \\ 0 & j \text{ odd} \\ \frac{2\sqrt{2}}{1-j^2} & j \text{ even} \end{cases}$$

It is often desirable to have a cubature formula that involves only the function values at the nodes and the corresponding cubature weights. A simple matrix formulation is still available. First, observe that

$$I_n(f) = \sum_{j \text{ even}}^n \sum_{l \text{ even}}^n c'_{j,l} m_{j,l} = \sum_{j \text{ even}}^n \sum_{l \text{ even}}^n c_{j,l} m'_{j,l}$$

with

$$\mathbb{M}_0 = (m'_{j,l}) = \begin{pmatrix} m_{0,0} & m_{0,2} & \cdots & \cdots & m_{0,p_n} \\ m_{2,0} & m_{2,2} & \cdots & m_{2,p_n-2} & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ m_{p_n-2,0} & m_{p_n-2,2} & 0 & \cdots & 0 \\ m'_{p_n,0} & 0 & \cdots & 0 & 0 \end{pmatrix} \in \mathbb{R}^{([\frac{n}{2}]+1) \times ([\frac{n}{2}]+1)}$$

where $p_n = n$ and $m'_{p_n,0} = m_{p_n,0}/2$ for n even, $p_n = n-1$ and $m'_{p_n,0} = m_{p_n,0}$ for n odd. Now, using the formula for the coefficients (4) we can write

$$\begin{aligned} I_n(f) &= \sum_{\boldsymbol{\xi} \in \text{Pad}_n} \lambda_{\boldsymbol{\xi}} f(\boldsymbol{\xi}) \\ &= \sum_{\boldsymbol{\xi} \in C_{n+1}^E \times C_{n+2}^O} \lambda_{\boldsymbol{\xi}} f(\boldsymbol{\xi}) + \sum_{\boldsymbol{\xi} \in C_{n+1}^O \times C_{n+2}^E} \lambda_{\boldsymbol{\xi}} f(\boldsymbol{\xi}) \end{aligned}$$

where

$$\lambda_{\boldsymbol{\xi}} = w_{\boldsymbol{\xi}} \sum_{j \text{ even}}^n \sum_{l \text{ even}}^n m'_{j,l} \hat{T}_j(\xi_1) \hat{T}_l(\xi_2) \quad (12)$$

Defining the Chebyshev matrix corresponding to even degrees

$$\mathbb{T}^E(S) = \begin{pmatrix} \hat{T}_0(s_1) & \cdots & \hat{T}_0(s_m) \\ \hat{T}_2(s_1) & \cdots & \hat{T}_2(s_m) \\ \vdots & \cdots & \vdots \\ \hat{T}_{p_n}(s_1) & \cdots & \hat{T}_{p_n}(s_m) \end{pmatrix} \in \mathbb{R}^{([\frac{n}{2}]+1) \times m}$$

and the matrices of interpolation weights on the subgrids of Padua points, $\mathbb{W}_1 = (w_{\boldsymbol{\xi}}, \boldsymbol{\xi} \in C_{n+1}^E \times C_{n+2}^O)^t$, $\mathbb{W}_2 = (w_{\boldsymbol{\xi}}, \boldsymbol{\xi} \in C_{n+1}^O \times C_{n+2}^E)^t$ it is then easy to show that the cubature weights $\{\lambda_{\boldsymbol{\xi}}\}$ can be computed in the matrix form

$$\mathbb{L}_1 = (\lambda_{\boldsymbol{\xi}}, \boldsymbol{\xi} \in C_{n+1}^E \times C_{n+2}^O)^t = \mathbb{W}_1 \cdot (\mathbb{T}^E(C_{n+1}^E))^t \mathbb{M}_0 \mathbb{T}^E(C_{n+2}^O)^t$$

$$\mathbb{L}_2 = (\lambda_{\boldsymbol{\xi}}, \boldsymbol{\xi} \in C_{n+1}^O \times C_{n+2}^E)^t = \mathbb{W}_2 \cdot (\mathbb{T}^E(C_{n+1}^O))^t \mathbb{M}_0 \mathbb{T}^E(C_{n+2}^E)^t$$

where the dot means that the final product is made componentwise.

An alternative approach is based on the observation that (12) itself is a Discrete Fourier Transform, the roles of the points and of the indexes being interchanged. An FFT-based implementation is then feasible, as in the univariate case (cf. [11]).

It is worth recalling that the cubature weights are not all positive, but the negative ones are few and of small size. Indeed, the cubature formula is stable and convergent for every continuous integrand, since, as it has been proved in [9],

$$\lim_{n \rightarrow \infty} \sum_{\boldsymbol{\xi} \in \text{Pad}_n} |\lambda_{\boldsymbol{\xi}}| = 4.$$

4 Numerical tests

In this section we present some numerical tests on the accuracy and performance of the various implementations of interpolation and cubature at the Padua points. All the experiments have been made by the Matlab/Octave package Padua2DM [3], run in Matlab 7.6.0 on an Intel Core2 Duo 2.20GHz processor.

In Tables 1 and 2 we show the CPU times (seconds) for the computation of the interpolation coefficients and cubature weights at a sequence of degrees, by the MM and the FFT-based algorithms. The results suggest that the FFT approach is preferable for interpolation, whereas the MM approach is better for cubature. Indeed, the MM algorithm is more efficient than the FFT-based one in the cubature instance, since the matrices have lower dimension due to restriction to even indexes, and is competitive with the FFT up to very high degrees.

n	20	40	60	80	100	200	300	400	500
MM	0.003	0.001	0.003	0.004	0.006	0.022	0.065	0.142	0.206
FFT	0.002	0.002	0.002	0.002	0.006	0.029	0.055	0.088	0.137

Table 1: CPU time (in seconds) for the computation of the interpolation coefficients.

n	20	40	60	80	100	200	300	400	500
MM	0.004	0.000	0.001	0.002	0.003	0.010	0.025	0.043	0.071
FFT	0.005	0.001	0.003	0.003	0.005	0.025	0.048	0.090	0.142

Table 2: CPU time (in seconds) for the computation of the cubature weights.

In Figure 2 we report the relative errors of interpolation (top) and cubature (bottom) for the classical Franke test function in $[0, 1]^2$ (versus the degree). Here a second advantage of the FFT approach for interpolation appears: it is able to arrive close to machine precision, whereas the MM algorithm stagnates around 10^{-13} . On the contrary, the MM algorithm seems more stable in the cubature than in the interpolation setting. These observations have been confirmed by many other numerical experiments.

In Figures 3 and 4 we show the interpolation and cubature errors versus the number of points (i.e., of function evaluations), for a Gaussian and a C^2 function. Interpolation and cubature at the Padua points are compared with tensorial formulas, and in the case of cubature also with the few known minimal formulas (cf. [8]).

We see two opposite situations. Concerning interpolation, the Padua points perform better than tensor-product Chebyshev–Lobatto points only on regular functions. On the other hand, nontensorial cubature at the Padua points performs always better than tensorial Clenshaw–Curtis cubature (which uses tensor-product Chebyshev–Lobatto points). However, it is less accurate than tensorial Gauss–Legendre–Lobatto and minimal formulas on analytic entire functions, whereas it appears the best one on less regular functions. This phenomenon, confirmed by many other examples (cf. [9])

and present also in 3d with nontensorial cubature at new sets of Chebyshev hyperinterpolation points (cf. [7]), is quite similar to that studied in the univariate case for the classical Clenshaw–Curtis formula (cf. [10]), but is still theoretically unexplained in the multivariate case. Nevertheless, numerical cubature at the Padua points seems to provide one of the best algebraic cubature formulas presently known for the square.

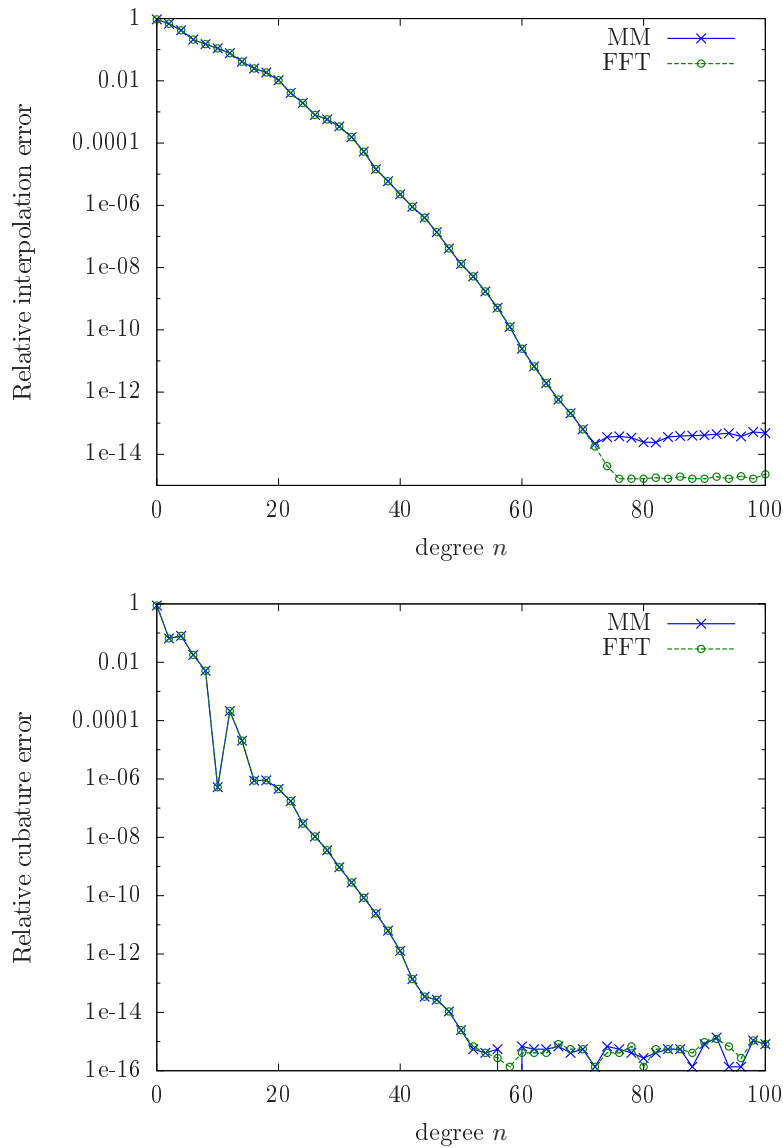


Figure 2: Errors of interpolation (top) and cubature (bottom) versus the interpolation degree for the Franke test function in $[0, 1]^2$, by the MM and the FFT-based algorithms.

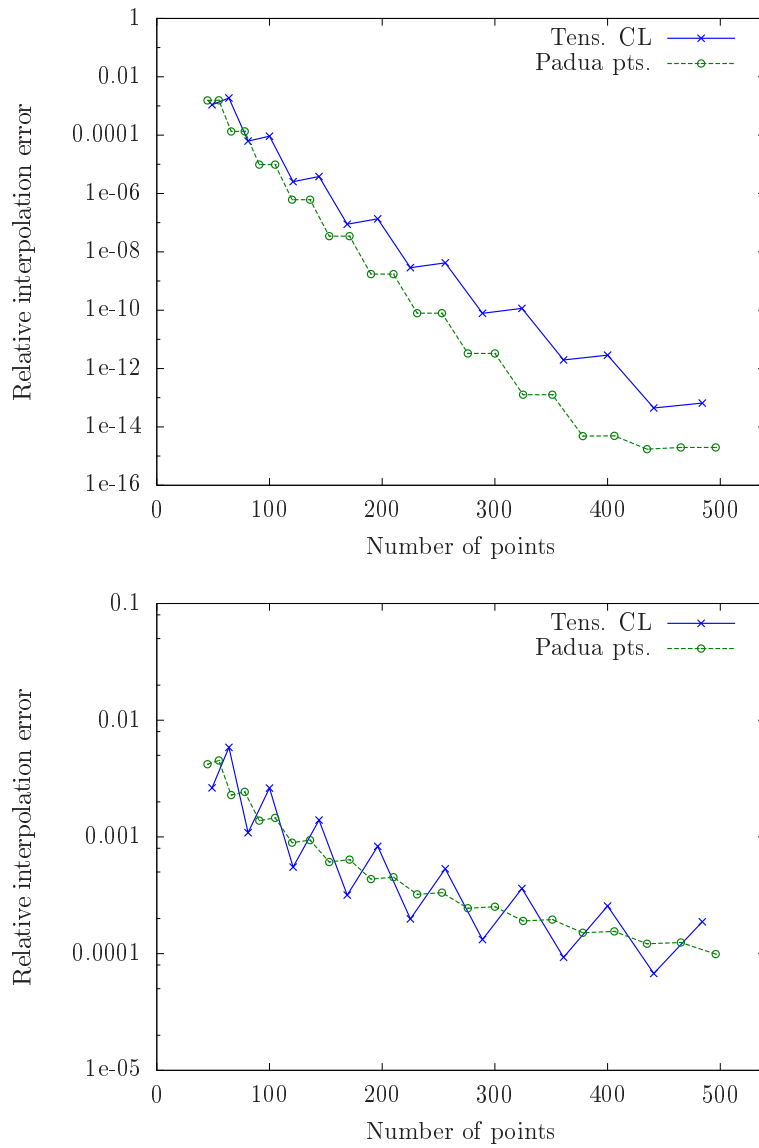


Figure 3: Relative interpolation errors versus the number of interpolation points for the Gaussian $f(\mathbf{x}) = \exp(-|\mathbf{x}|^2)$ (top) and the C^2 function $f(\mathbf{x}) = |\mathbf{x}|^3$ (bottom) in $[-1, 1]^2$; Tens. CL = Tensorial Chebyshev–Lobatto interpolation.

References

- [1] L. Bos, M. Caliarì, S. De Marchi, M. Vianello, and Y. Xu: Bivariate Lagrange interpolation at the Padua points: the generating curve approach. *J. Approx. Theory* **143** (2006) 15–25.
- [2] L. Bos, S. De Marchi, M. Vianello, and Y. Xu: Bivariate Lagrange interpolation

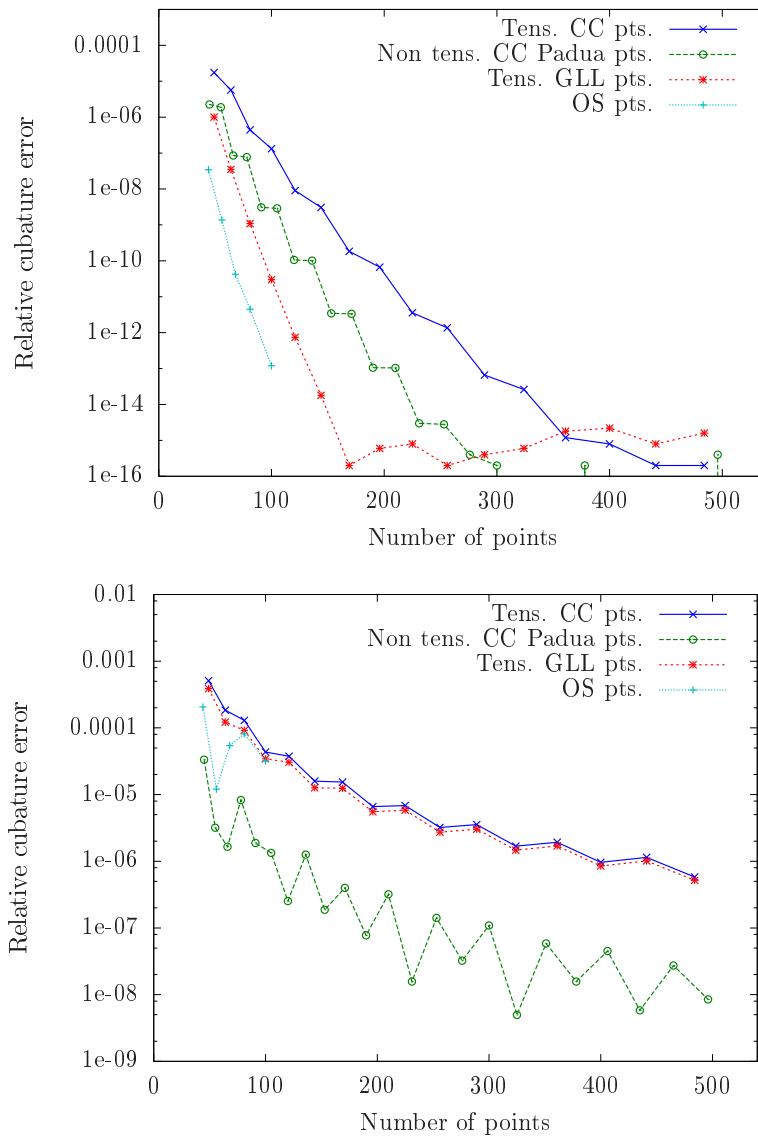


Figure 4: Relative cubature errors versus the number of cubature points (CC = Clenshaw–Curtis, GLL = Gauss–Legendre–Lobatto, OS = Omelyan–Solovyan) for the Gaussian $f(\mathbf{x}) = \exp(-|\mathbf{x}|^2)$ (top) and the C^2 function $f(\mathbf{x}) = |\mathbf{x}|^3$ (bottom) in $[-1, 1]^2$.

at the Padua points: the ideal theory approach. Numer. Math. **108** (2007) 43–57.

- [3] M. Caliari, S. De Marchi, A. Sommariva, and M. Vianello: Padua2DM (a Matlab/Octave code for interpolation and cubature at the Padua points), software available at <http://profs.sci.univr.it/~caliari/software> (2009).

- [4] M. Caliari, S. De Marchi, and M. Vianello: Bivariate polynomial interpolation on the square at new nodal sets. *Appl. Math. Comput.* **165** (2005) 261–274.
- [5] M. Caliari, S. De Marchi, and M. Vianello: Algorithm 886: Padua2D: Lagrange Interpolation at Padua Points on Bivariate Domains. *ACM Trans. Math. Software* **35-3** (2008).
- [6] C.W. Clenshaw, A.R. Curtis: A method for numerical integration on an automatic computer. *Numer. Math.* **2** (1960) 197–205
- [7] S. De Marchi, M. Vianello, and Y. Xu: New cubature formulae and hyperinterpolation in three variables. *BIT Numerical Mathematics* **49(1)** (2009) 55–73.
- [8] I.P. Omelyan and V.B. Solovyan: Improved cubature formulae of high degrees of exactness for the square. *J. Comput. Appl. Math.* **188** (2006) 190–204.
- [9] A. Sommariva, M. Vianello, and R. Zanghè, R.: Nontensorial Clenshaw–Curtis cubature. *Numer. Algorithms* **49** (2008) 409–427.
- [10] L.N. Trefethen: Is Gauss quadrature better than Clenshaw–Curtis?. *SIAM Rev.* **50** (2008) 67–87.
- [11] J. Waldvogel: Fast construction of the Fejér and Clenshaw–Curtis quadrature rules. *BIT Numerical Mathematics* **46** (2006) 195–202.