inRS: implementing the indicator function of NURBS-shaped planar domains

Alvise Sommariva[a,b,*], Marco Vianello[a,b]

[a] *University of Padova, Italy*
[b] *Member of the INdAM Research group GNCS*

Abstract

We provide an algorithm that implements the indicator function of NURBS-shaped planar domains, tailored to the fast computation on huge point clouds, together with the corresponding Matlab code.

Keywords: NURBS-shaped planar domains, indicator function, crossing number.
2020 MSC: 65D07, 65D17, 65D18.

1. Introduction

NURBS-shaped domains produced by CAGD algorithms play a central role in digital design and modelling processes. The capability of locating quasi-uniform or random sample points in such domains can be useful in a vast range of applications, for example within several meshfree bivariate approximation algorithms developed in the last twenty years, among which we may quote (without any pretence of completeness) kernel-based and partition-of-unity collocation methods [5, 8, 11], construction of algebraic cubature formulas [12, 22, 23] potentially useful for curved FEM/VEM elements [1, 21], numerical differentiation by scattered data [6, 7, 9], compressed MC/QMC integration [2, 13], compressed polynomial regression [4, 19].

Though the efficient computation of the indicator function of general linear polygons has received much attention in the algorithmic literature and deserved sophisticated implementations (cf. e.g. [15]), such as the Matlab INPOLYGON function (and the ISINTERIOR function in the POLYSHAPE envinronment), or the recent INPOLY2 function [10], the same cannot be apparently said concerning NURBS-shaped curved polygons (at least in Matlab), despite their relevance in applications.

Extending and improving an approach already explored in [22, 23], as well as resorting to some ideas used in [10] for linear polygons, we try to fill this lack by providing an efficient and robust Matlab implementation of the indicator function of NURBS-shaped Jordan domains, based on the topological notions of crossing number (even-odd rule) and winding number. A key tool is encapsulating the domain boundary by a finite number of Cartesian rectangles, in all of which it is the graph of a local monotone Cartesian function. We have tried to optimize all the algorithm blocks and to conveniently manage the critical situations, the present strategy being mainly tailored to fast computation of the indicator function on huge point clouds. The corresponding Matlab code [24], that could be useful in many design and modelling applications, is freely available to the scientific community.

2. NURBS-shaped indicator function

In this section we discuss an algorithm for the computation of the indicator function of bidimensional NURBS-shaped Jordan domains, based on the topological notions of crossing number and winding number,

*Corresponding author
Email addresses: alvise@math.unipd.it (Alvise Sommariva), marcov@math.unipd.it (Marco Vianello)

whose main lines have already appeared in [23]. The present implementation is more efficient and robust, and for the reader's convenience we explain in some detail the whole construction.

Consider a Jordan domain $\Omega \subset \mathbb{R}^2$, such that

- its boundary $\partial\Omega$ is a simple curve described by parametric equations

$$\Gamma(t) = (\alpha(t), \beta(t)) \ , \quad t \in I = [t_{min}, t_{max}] \ , \tag{1}$$

  where $\alpha, \beta \in C(I)$, $\Gamma(t_{min}) = \Gamma(t_{max})$;

- there are partitions $\{I_k\}_{k=1,\dots,K}$ of I, and $\{I_{k,j}\}_{j=1,\dots,m_k}$ of each $I_k = [t_k, t_{k+1}]$, such that the restrictions of α, β to each I_k are *rational splines*, w.r.t. the subintervals $\{I_{k,j}\}$.

We shall denote the local rational splines as

$$\alpha(t) = \frac{u_k(t)}{v_k(t)}, \quad \beta(t) = \frac{w_k(t)}{z_k(t)}, \quad t \in I_k \ , \tag{2}$$

where the numerators u_k, w_k as well as the denominators v_k, z_k are polynomial splines on I_k, sharing the same knots and having degree, respectively, p_k and q_k. We assume that the denominators v_k, z_k do not vanish in the closed interval I_k. Moreover, we have that $\partial\Omega = \cup_{k=1}^{M}(V_k \frown V_{k+1})$, with the convention that $V_{K+1} = V_1$, where the *vertices* $\{V_k\}$ can be corner points or even cusps of the boundary. On the other hand, in each subinterval $I_{k,j} \subset I_k$ we have that

$$\alpha(t) = \frac{u_{k,j}(t)}{v_{k,j}(t)}, \quad \beta(t) = \frac{w_{k,j}(t)}{z_{k,j}(t)}, \quad t \in I_{k,j} \ , \tag{3}$$

where the numerators $u_{k,j}, w_{k,j}$ are polynomials of degree p_k and the denominators $v_{k,j}, z_{k,j}$ are polynomials of degree q_k.

We are particularly interested in the case when the boundary is locally a p-th degree NURBS curve [20, p.117], i.e. the curvilinear side $V_k \frown V_{k+1}$ has the following parametrization

$$\Gamma(t) = \frac{\sum_{i=1}^{m_k} B_{i,p}(t)\, \lambda_{i,k} C_{i,k}}{\sum_{i=1}^{m_k} B_{i,p}(t)\, \lambda_{i,k}}, \quad t \in [t_k, t_{k+1}] \ , \tag{4}$$

where $\{C_{i,k}\}_{i=1}^{m_k} \subset \mathbb{R}^2$ are the *control points*, $\{\lambda_{i,k}\}_{i=1}^{m_k}$ are the weights and $\{B_{i,p}\}_{i=1}^{m_k}$ are the p-th degree B-spline basis functions [3, p.87] defined on a suitable knot vector. We stress that the case of standard polynomial splines (already treated in [22]) is included in (1)-(3), and that the whole construction below can be extended also to other domains with Rational Spline (RS) boundary, such as rational Bezier curves.

We base our indicator function algorithm *inRS* on the well-known *Jordan curve theorem*, which implies that a point P belongs to a Jordan domain Ω if and only if, taking a point $P^* \notin \Omega$, the segment $\overline{P^*P}$ *crosses* $\partial\Omega$ an odd number of times; cf. e.g. [16] and the nice paper [14] with the references therein. There may be some "critical" cases, for instance when $\overline{P^*P}$ touches a vertex without crossing the boundary, or when it is tangent to the boundary; cf. Fig. 1 where vertical segments are used as in our main implementation. In these cases the crossing number strategy cannot be directly applied and alternatives have to be adopted, such as computation of the winding number.

The first step of the algorithm consists in covering $\partial\Omega$ with a finite union of suitable Cartesian rectangles, in all of which the boundary is the graph of a local monotone Cartesian function. Each rectangle contains a portion of $\partial\Omega$ that is parametrized by two *rational functions*, i.e. locally $(\alpha(t), \beta(t))$ are the ratio of two polynomials. Once these covering rectangles have been determined, evaluating the crossing number $cross(P)$ (i.e. computing the indicator function at a given point P) becomes easy, requiring at most the solution of some polynomial equations.

To this purpose, first we compute in each $I_{k,j}$ the possible zeros of $\alpha'(t) = (u'_{k,j}v_{k,j} - u_{k,j}v'_{k,j})/v^2_{k,j}$ and $\beta'(t) = (w'_{k,j}z_{k,j} - w_{k,j}z'_{k,j})/z^2_{k,j}$, obtaining a finer partition, say $I_{k,j,s}$, with such zeros as endpoints. Notice that this requires solving in each $I_{k,j}$ the two polynomial equations of degree $p_k + q_k - 1$

$$u'_{k,j}(t)v_{k,j}(t) - u_{k,j}(t)v'_{k,j}(t) = 0 \ , \quad w'_{k,j}(t)z_{k,j}(t) - w_{k,j}(t)z'_{k,j}(t) = 0 \ , \tag{5}$$

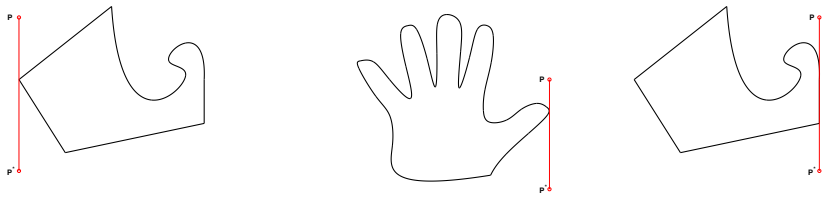

Figure 1: Critical situations for the crossing number on curvilinear domains.

which can be conveniently done in Matlab by the ROOTS function, that comptes the eigenvalues of the companion matrix.

Now, $\alpha(t)$ and $\beta(t)$ being polynomials, if not constant are strictly monotone in each $I_{k,j,s}$, and thus the boundary curve is there the graph of a strictly monotone Cartesian function, with local bounding box

$$\mathcal{B}_{k,j,s} = [a_{k,j,s}, b_{k,j,s}] \times [c_{k,j,s}, d_{k,j,s}] \tag{6}$$

$$a_{k,j,s} = \min_{t \in I_{k,j,s}} \alpha(t) \ , \ b_{k,j,s} = \max_{t \in I_{k,j,s}} \alpha(t)$$

$$c_{k,j,s} = \min_{t \in I_{k,j,s}} \beta(t) \ , \ d_{k,j,s} = \max_{t \in I_{k,j,s}} \beta(t)$$

that we call a *monotone box*. The local minima and maxima can be determined via the values of α, β at the endpoints of $I_{k,j,s}$. The case of $\alpha(t)$ or $\beta(t)$ constant are treated separately and correspond to bounding boxes degenerating into a vertical or horizontal segment, respectively. Clearly, the union of such monotone boxes, that may overlap, covers the whole boundary $\partial\Omega$; cf. Fig. 2. Moreover, a global bounding box for the whole Ω can be immediately determined by the upmost, downmost, leftmost and rightmost monotone boxes.

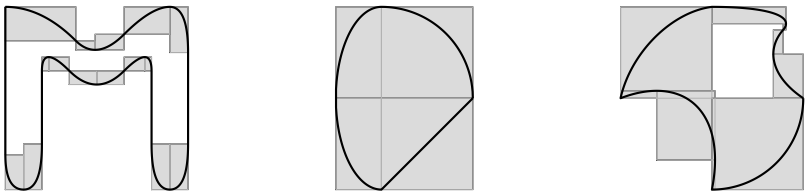

Figure 2: Three NURBS-shaped domains whose boundary may contain arc of circles, ellipses, segments as well as other NURBS blocks, together with the corresponding monotone boxes. Notice that the latter may degenerate into a segment (left figure) and may overlap (right figure).

Now, take a point $P = (\overline{x}, \overline{y}) \in \mathbb{R}^2$. If P is out of the global bounding box, then clearly the indicator function at P is null, $\chi_\Omega(P) = 0$. If P is in the global bounding box, consider the vertical "downward" ray $\{x = \overline{x}, y \leq \overline{y}\}$ and the monotone boxes $\mathcal{B}_\ell$ corresponding to the triples

$$\{\ell = (k, j, s) : \ a_{k,j,s} \leq \overline{x} < b_{k,j,s} \ , \ \ \overline{y} \geq c_{k,j,s}\} \ , \tag{7}$$

that are the boxes that do intersect the ray and can be effectively determined by a fast vectorized search. Notice that if $\overline{y} > d_\ell$ (that is P is over $\mathcal{B}_\ell$) the ray surely intersects at one point the boundary portion pertaining to such a monotone box, whereas if P is in $\mathcal{B}_\ell$ the possible intersection can be ascertained by solving the polynomial equation of degree $\max\{p_k, q_k\}$

$$u_{k,j}(t) - \overline{x}\, v_{k,j}(t) = 0 \ , \ \ t \in I_\ell \ , \tag{8}$$

which again can be conveniently done in Matlab by the ROOTS function. Indeed, let t_ℓ be the unique solution. Then if $\beta(t_\ell) = w_{k,j}(t_\ell)/z_{k,j}(t_\ell) \le \overline{y}$ the ray intersects the boundary at $(\alpha(t_\ell), \beta(t_\ell))$ and the intersection counting must be increased by 1, whereas if $\beta(t_\ell) > \overline{y}$ it does not.

At this point, let $cross(P)$ be the overall number of such intersections. If none of them is a critical point, $cross(P)$ is just the crossing number of Jordan curve theorem. This can be easily ascertained by checking that $\overline{x}$ is not the abscissa of a boundary point where α' vanishes (*point of vertical tangency*), or of a vertex $V_i = (\alpha(t_i), \beta(t_i))$ such that $\alpha'(t_i^-)\alpha'(t_i^+) < 0$, i.e. where the boundary curve turns form left to right or conversely (*x-turning vertex*), both being possible intersections without crossing. Then P belongs to Ω if and only if $cross(P)$ is odd, i.e. the indicator function at P is

$$\chi_\Omega(P) = cross(P) \bmod 2 \, . \tag{9}$$

In addition, one may also know whether the point is on the boundary or in the interior of Ω, by checking whether one of such intersections coincides with P (up to a suitable numerical tolerance).

Whenever critical points are present among the intersections, one may use the same procedure working with an horizontal ray. In the rare case of another failure still due to critical points, one can resort to a different topological index, i.e. the winding number $wind(P) \in \mathbb{Z}$ (cf. e.g. [18])

$$wind(P) = \frac{1}{\text{length}(I)} \int_I \frac{\beta'(t)\,(\alpha(t) - \overline{x}) - \alpha'(t)\,(\beta(t) - \overline{y})}{(\alpha(t) - \overline{x})^2 + (\beta(t) - \overline{y})^2}\, dt \tag{10}$$

$$= \frac{1}{\text{length}(I)} \sum_{k,j} \int_{I_{k,j}} \frac{\beta'(t)\,(\alpha(t) - \overline{x}) - \alpha'(t)\,(\beta(t) - \overline{y})}{(\alpha(t) - \overline{x})^2 + (\beta(t) - \overline{y})^2}\, dt \, , \quad P = (\overline{x}, \overline{y}) \notin \partial\Omega \, ,$$

since (assuming that the boundary is counterclockwise oriented) the indicator function at P is

$$\chi_\Omega(P) = wind(P) \, . \tag{11}$$

We observe that the evaluation of the integral above, for example by high-precision Gaussian quadrature on each subinterval in view of the fact that $\alpha, \beta \in C^\infty(I_{k,j})$, can be difficult when P is close to the boundary. In practice, however, there is no need to compute such a quantity with a small error, recalling that $wind(P)$ is an integer (so that an error strictly less than $1/2$ would suffice).

Remark 2.1. One may cover the boundary with a larger number of monotone boxes, by further partitioning each $I_{k,j,s}$ into a number of subintervals $I_{k,j,s,\tau} = [a_{k,j,s,\tau}, b_{k,j,s,\tau}]$. In such a way the boxes become clearly smaller and we have a finer approximation of the NURBS-shaped boundary: see Fig. 2. Given a point $P = (\overline{x}, \overline{y})$ the pertaining boxes correspond to the quadruples

$$\{\ell = (k, j, s, \tau) : \; a_{k,j,s,\tau} \le \overline{x} \le b_{k,j,s,\tau} \, , \;\; \overline{y} \ge c_{k,j,s,\tau}\} \, , \tag{12}$$

where $c_{k,j,s,\tau} = \min_{t \in I_{k,j,s,\tau}} \beta(t) = \min\{\beta(a_{k,j,s,\tau}), \beta(b_{k,j,s,\tau})\}$.

On the other hand, the smaller the boxes the smaller the probability that a random point is inside some box, and thus we can substantially reduce the number of equations like (8) to be solved by ROOTS in order to find the intersections and speed-up the whole procedure, when a huge set of points has to be located. In practice, however, there is a suitable threshold for the number of sub-boxes, over which refining is no more convenient (such a threshold can roughly determined experimentally).

To fix ideas, the guidelines of the implementation based on the discussion above can be roughly summarized by the following

Algorithm inRS: NURBS-shaped indicator function

input: a point cloud S, the NURBS boundary parametrization $\Gamma(t) = (\alpha(t), \beta(t))$

(i) determine the monotone boxes by computing the zeros of $\alpha'(t)$ and of $\beta'(t)$ as in (5)-(6)

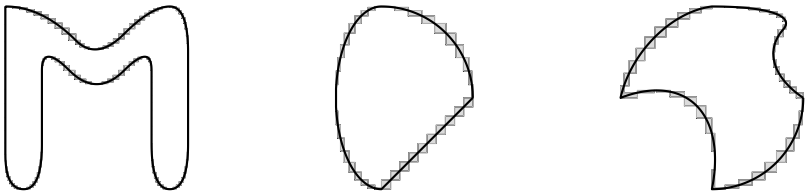

Figure 3: Box refinement for the three NURBS-shaped domains of Fig. 2.

(*ii*) compute a bounding box B for Ω by the extremal monotone boxes and collect the critical boundary points: vertical ($\alpha' = 0$) or horizontal ($\beta' = 0$) tangency points, x-turning or y-turning vertices

(*iii*) for all $P \in S \cap B^c$: set $inRS(P) = 0$

(*iv*) for all $P \in S \cap B$:

(*v*) compute $cross(P)$, the number of boundary intersections of a vertical ray from P in the monotone boxes $\mathcal{B}_\ell$, with ℓ given in (7) or (12) in case of box refinement (solving explicitly (8) only if $P \in \mathcal{B}_\ell$)

(*vi*) if the intersections do not include critical points (cf. Fig. 1)
then $inRS(P) = cross(P) \bmod 2$
else

(*vii*) repeat step (*v*) with an horizontal ray

(*viii*) if the intersections do not include critical points
then $inRS(P) = cross(P) \bmod 2$
else compute $inRS(P) = wind(P)$

output: for all $P \in S$, $inRS(P) = 1$ if $P \in S \cap \Omega$, $inRS(P) = 0$ otherwise

It is worth stressing some relevant features. First, if P is a pointset instead of a single point, steps $(i) - (ii)$ can be clearly done once. Steps (i), (v) and (vii) require solving polynomial equations, that can be conveniently done by the Matlab ROOTS function. Computation of $wind(P)$ in $(viii)$ has been implemented by Gaussian quadrature along the boundary but is more costly than $cross(P)$ (whenever the latter is feasible), so it has been reserved to dubious cases.

In addition, we have tried optimizing all the algorithm blocks, conveniently managing the critical situations and implementing more features than those present in the polynomial spline version [22] and in the rational splines alpha-version used in [23], that we call $inRS1$ below. For example, before resorting to $wind(P)$ we have provided the horizontal ray step after the vertical one, and we have added the possibility of knowing whether the point is in the domain interior or on the boundary (up to a suitable numerical tolerance).

The main improvement with respect to [22] is however in the implementation of (iv). Indeed, in [22] the algorithm loops over the points in $S \cap B$ and for each point loops over the monotone boxes. With M points and N boxes the computational complexity is then $\mathcal{O}(MN)$.

Differently, in the present version of the algorithm we chose the strategy already used in [10] for linear polygons, based on point ordering and binary search, as follows:

- orders the points in $S \cap B$ w.r.t. the x-variable
- loops over the monotone boxes
  - for each monotone box finds all the points P satisfying (7) (or (12)) by binary search followed by local linear search

– for all such points performs (v) and possibly updates $cross(P)$ whenever a noncritical intersection is found

eventually, for every point $P \in S \cap B$, either the overall number of crossings is computed and (vi) can be done, or P is recognized as "dubious"

- performs (vii)-$(viii)$ for all dubious points

Algorithm $inRS$ with the implementation just described will be called $inRS2$ below. With M points and N boxes the computational complexity is now reduced to $\mathcal{O}(M \log_2(M))$ for the ordering, plus $\mathcal{O}(N \log_2(M))$ for the binary searches, plus $\mathcal{O}(M\nu)$ where ν is the mean number of monotone boxes to which a point in $S \cap B$ belongs (when the boxes are small and the points quasi-uniformly distributed this number, which corresponds to the number of equations like (8) to be solved by ROOTS, can be very small).

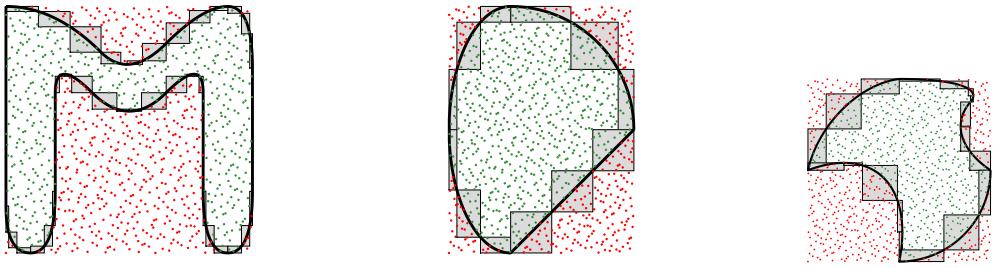

Figure 4: Halton points inclusion for the three NURBS-shaped domains of Fig. 2.

A graphical example concerning the inclusion check on 1000 Halton points of the domain bounding box is given in Fig. 4. We stress that equations like (8) have to be solved only for the points belonging to the monotone boxes. In order to give an illustration of the algorithm performance, we display Table 1 below, where we locate 10^i Halton points of the global bounding box, $i = 3, 4, 5$, on the three NURBS-shaped domains of Fig. 2; the speed-up is rounded to two significant figures and to manage CPU time fluctuations we have taken the median over 100 runs of the algorithms. As expected, the speed-up of $inRS2$ over $inRS1$ increases by increasing the cardinality, taking however into account that both have a fixed cost due to the construction of monotone boxes which is relevant at the lower cardinalities.

Similar results are obtained with the same number of grid points or random points. The numerical tests of the present paper have been done on a M1-chip PC with 16 GB of RAM, running Matlab R2021b. All the Matlab codes and demos of the present version are freely available at [24].

Remark 2.2. We stress that suitable user-friendly data structures have been used, that allow to manage as boundary either a NURBS curve (a feature apparently missing or at least complicated within basic Matlab) or a polynomial spline curve or even a rational/polynomial Bezier curve.

It is also worth pointing out what is the main purpose of this work. Given a boundary tracked by polynomial splines or NURBS, for example an object obtained by CAD in industrial design, we provide an algorithm to compute the indicator function with a great accuracy. On the other hand, often in applications only a discretization of the boundary is available as an ordered set of points, or even the domain itself is known only in fully discrete form as a point cloud.

We have added in the Matlab package suitable demos, which show how to reconstruct the boundary from a discrete ordered sample, by splines, composite Bezier curves or NURBS. In addition, the case of noisy boundary points can be treated for example by smoothing periodic parametric splines. On the other hand, if only a point cloud is known, we provide a demo where the boundary is first reconstructed piecewise linearly by Matlab ALPHASHAPE and then tracked more accurately using the polygonal vertices as NURBS control points, for specific order, weights and knots. For the sake of concision, we do not report here demo results and figures concerning such accessory features of the package, that can be conveniently run by interested users downloading [24].

#	algorithm	Fig. 2-left	Fig. 2-center	Fig. 2-right
10^3	$inRS1$	$4.3e{-}03s$	$1.9e{-}03s$	$2.1e{-}03s$
	$inRS2$	$2.8e{-}03s$	$1.3e{-}03s$	$1.3e{-}03s$
	speed-up	1.5	1.5	1.6
10^4	$inRS1$	$1.8e{-}02s$	$8.8e{-}03s$	$9.5e{-}03s$
	$inRS2$	$4.4e{-}03s$	$3.2e{-}03s$	$3.2e{-}03s$
	speed-up	4.1	2.8	3.0
10^5	$inRS1$	$1.6e{-}01s$	$7.3e{-}02s$	$8.0e{-}02s$
	$inRS2$	$1.7e{-}02s$	$2.1e{-}02s$	$2.0e{-}02s$
	speed-up	9.4	3.5	4.0

Table 1: CPU time of $inRS$ on the three NURBS-shaped domains of Fig. 2-3 with # Halton points of the corresponding bounding box; $inRS1$ is the alpha-version in [23] whereas $inRS2$ is the present version.

Remark 2.3. It is worth recalling the fact that all inequalities in the algorithm above, as usual in computational geometry, are checked up to a given tolerance, so that ultimately we only know whether a point P is in or out a suitable neighborhood of the NURBS-shaped domain (we have used a default tolerance of 10^{-12} where necessary).

Since there are clever implementations of the in-domain check for linear polygons, one may think that it could be better to approximate the NURBS-shaped boundary by a piecewise linear curve with a very high number of sides up to the given tolerance, and then to use such fast point-in-polygon algorithms (roughly, if a tolerance ε is given, the number of sides is $O\left(\varepsilon^{-1/2}\right)$ since the error of linear approximation is $O(\Delta t^2)$ for a piecewise C^2 NURBS parametrization).

In practice, however, for such small tolerances our implementation of the NURBS-shaped indicator function is faster, as it can be realized by Table 2 below, where we compare the INPOLY2 Matlab function [10, 17] (that largely overcomes the standard INPOLYGON on a large number of trial points) applied to a polygonal approximation up to a 10^{-10} tolerance (order of 10^5 sides), with our $inRS2$ on the three NURBS-shaped domains of Fig. 2-3. Notice that the speed-up of $inRS2$ over INPOLY2 tends to decrease by increasing the cardinality of the point set to be located. The dynamics of CPU times in the present range of cardinalities can be explained observing that, while that of $inRS2$ is substantially ruled by the number of points (the number of boxes being relatively small with respect to it) up to the fixed initial cost of the boxes construction, that of INPOLY2 is instead substantially ruled by the huge number of polygon sides.

#	algorithm	Fig. 2-left	Fig. 2-center	Fig. 2-right
10^3	INPOLY2	$2.1e{-}01s$	$5.2e{-}02s$	$6.5e{-}02s$
	$inRS2$	$2.8e{-}03s$	$1.3e{-}03s$	$1.3e{-}03s$
	speed-up	75	40	50
10^4	INPOLY2	$2.5e{-}01s$	$6.4e{-}02s$	$8.0e{-}02s$
	$inRS2$	$4.4e{-}03s$	$3.2e{-}03s$	$3.2e{-}03s$
	speed-up	58	20	25
10^5	INPOLY2	$3.1e{-}01s$	$8.7e{-}02s$	$1.1e{-}01s$
	$inRS2$	$1.7e{-}02s$	$2.1e{-}02s$	$2.0e{-}02s$
	speed-up	18	4.1	5.5

Table 2: CPU time of the $inRS2$ algorithm on the three NURBS-shaped domains of Fig. 2-3 with # Halton points of the corresponding bounding box, compared with the Matlab INPOLY2 function applied to an approximating polygon up to a 10^{-10} tolerance (order of 10^5 sides).

Acknowledgements

This work was partially supported by the DOR funds and the biennial project BIRD 192932 of the University of Padova, and by the INdAM-GNCS, and has been accomplished within the RITA Research ITalian network on Approximation and the UMI Group TAA Approximation Theory and Applications.

References

[1] L. Beirão da Veiga, A. Russo and G. Vacca, The Virtual Element Method with curved edges, ESAIM Math. Model. Numer. Anal., 53, 2019.

[2] L. Bittante, S. De Marchi and G. Elefante, A new quasi-Monte Carlo technique based on nonnegative least-squares and approximate Fekete points, Numer. Math. TMA., 9, 2016.

[3] C. de Boor, A Practical Guide to Splines, Rev.ed. Springer-Verlag, New York, 2001.

[4] L. Bos, F. Piazzon and M. Vianello, Near G-optimal Tchakaloff designs, Comput. Statistics, 35, 2020.

[5] J.-S. Chen and T. Belytschko, Meshless and Meshfree Methods, in: B. Engquist Ed., Encyclopedia of Applied and Computational Mathematics, Springer, 2015, pp. 886–894.

[6] O. Davydov and R. Schaback, Error bounds for kernel-based numerical differentiation. Numer. Math., 132, 2016.

[7] O. Davydov and R. Schaback, Minimal numerical differentiation formulas, Numer. Math., 140, 2018.

[8] F. Dell'Accio, F. Di Tommaso, O. Nouisser and N. Siar, Solving Poisson equation with Dirichlet conditions through multinode Shepard operators, Comput. Math. Appl., 98, 2021.

[9] F. Dell'Accio, F. Di Tommaso, N. Siar and M. Vianello, Numerical differentiation on scattered data through multivariate polynomial interpolation, BIT Numer. Math., published online 7 October 2021.

[10] D. Engwirda, INPOLY: A fast points-in-polygon test, GitHub, 2021: `https://github.com/dengwirda/inpoly`.

[11] G.E. Fasshauer and M.J. McCourt, Kernel-based Approximation Methods using Matlab, Interdisciplinary Mathematical Sciences, Vol. 19, World Scientific Publishing Co., Singapore, 2015.

[12] D. Gunderman, K. Weiss and J.A. Evans, Spectral mesh-free quadrature for planar regions bounded by rational parametric curves, Computer-Aided Design, 130, 2021.

[13] S. Hayakawa, Monte Carlo cubature construction, Jpn. J. Ind. Appl. Math., 38, 2021.

[14] T.C. Hales, Jordan's Proof of the Jordan Curve Theorem, Studies in Logic, Grammar and Rethoric, 10 (23), 2007.

[15] K. Hormann and A. Agathos, The point in polygon problem for arbitrary polygons, Comput. Geom., 20, 2001.

[16] C. Jordan, Course d'analyse de l'École Polytechnique, Gauthier-Villars, Paris, 1893.

[17] J. Kepner, A. Kipf, D. Engwirda, & al., Fast Mapping onto Census Blocks, 2020 IEEE High Performance Extreme Computing Conference (HPEC), arXiv:2005.03156v2.

[18] S.G. Krantz, The Index or Winding Number of a Curve about a Point, §4.4.4 in Handbook of Complex Variables, Birkhäuser, Boston, 1999.

[19] F. Piazzon, A. Sommariva and M. Vianello, Caratheodory-Tchakaloff Least Squares, Sampling Theory and Applications 2017, IEEE Xplore Digital Library, DOI: 10.1109/SAMPTA.2017.8024337.

[20] L. Piegl, The NURBS Book, Second Edition, Springer-Verlag, Berlin-Heidelberg, 1997.

[21] R. Sevilla and S. Fernández-Méndez, Numerical integration over 2D NURBS-shaped domains with applications to NURBS-enhanced FEM, Finite Elements in Analysis and Design, 47, 2011.

[22] A. Sommariva and M. Vianello, Computing Tchakaloff-like cubature rules on spline curvilinear polygons, Dolomites Res. Notes Approx. DRNA, 14, 2021.

[23] A. Sommariva and M. Vianello, Low-cardinality Positive Interior cubature on NURBS-shaped domains, preprint, 2021, available online at: `https://www.math.unipd.it/~marcov/pdf/nurbscatch.pdf`.

[24] A. Sommariva and M. Vianello, inRS: a Matlab code for the indicator function of NURBS-shaped planar domains, available online at: `https://www.math.unipd.it/~alvise/software.html`.