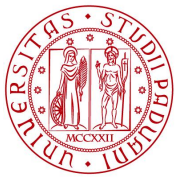


UNIVERSITÀ DEGLI STUDI DI PADOVA

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"

Corso di Laurea Triennale in Matematica

Tesi di Laurea

Formule di cubatura a bassa cardinalità di tipo Positive Interior su domini molteplicemente connessi con bordo di tipo NURBS

Relatore
Prof. **Alvise Sommariva**

Laureanda
Chiara Zara
Matricola: 1201459

Correlatore
Dott. **Giacomo Elefante**

Anno Accademico 2023/2024
19 Luglio 2024

Indice

Introduzione

In questo lavoro viene introdotto un algoritmo che calcola una formula di cubatura algebrica a bassa cardinalità di grado n su poligoni curvilinei definiti da funzioni razionali a tratti, anche non semplicemente connessi.

Lo scopo è di estendere l'approccio utilizzato per poligoni curvilinei a spline sviluppato in [6] e [7], fornendo un algoritmo (implementato in MATLAB) che calcoli una formula di cubatura di tipo PI (acronimo di *Positive weights and Interior nodes*) a bassa cardinalità di grado algebrico di esattezza ADE $= n$, con al massimo $\frac{(n+1)(n+2)}{2} = \dim(P_n^2)$ nodi, su poligoni curvilinei definiti da funzioni razionali a tratti (qui e di seguito $\dim(P_n^d)$ indica lo spazio dei polinomi d-variati con grado totale non superiore a n).

In particolare verranno presi in considerazione domini il cui bordo è definito a tratti da curve NURBS ("Non-Uniform Rational B-Splines").

Elementi chiave per tale analisi sono un teorema del 1976 di Wilhelmsen, un algoritmo di *in-domain* specifico nel dominio per tali poligoni curvilinei e un altro per il calcolo della soluzione sparsa e non negativa di sistemi di corrispondenza di momenti sotto-determinati da parte del risolutore di minimi quadrati non negativi di Lawson-Hanson.

La routine implementata, dopo aver calcolato i momenti γ mediante l'utilizzo della formula di Gauss-Green, partendo da una mesh sufficientemente densa di punti nella bounding-box del dominio non semplicemente connesso $S = S^{\text{ext}} \backslash S^{\text{in}}$, determinerá i punti che stanno in S^{ext} ma non in S^{in} e quindi cercherá di determinare mediante l'algoritmo di Lawson-Hanson la formula richiesta. In caso di insuccesso, ripeterá la procedura da una mesh piú densa.

Nella parte di test, illustriamo su 4 domini forati con bordi definiti da NURBS i risultati ottenuti, evidenziando per gradi di precisione fino a 10 la bontá dell'approccio intrapreso.

Capitolo 1

Nozioni preliminari

1.1 Cubatura

Il proposito di una *formula di cubatura* è di approssimare il valore di un integrale definito mediante valutazioni dell'integranda in un insieme discreto di punti.

Il problema della cubatura numerica consiste quindi nell'approssimare

$$I_w(f) = \int_\Omega f(x)w(x)\,\mathrm{d}x \tag{1.1}$$

mediante

$$Q_n(f) := \sum_{j=1}^{n} w_j f(x_j), \quad w_j \in \mathbb{R}, x_j \in \mathbb{R}^d \quad (2) \tag{1.2}$$

Dove sono dati $f : \Omega \subset \mathbb{R}^d \to \mathbb{R}$ funzione continua, Ω dominio compatto e $w : \mathbb{R}^d \to \mathbb{R}_{\geq 0}$ una funzione peso in Ω ovvero una funzione:

- non negativa, con $\Omega \subset \mathbb{R}^d$ non necessariamente limitato,
- $\int_\Omega |x|^n w(x)\ \mathrm{d}x < +\infty$ per tutti gli $n \in \mathbb{N}$;
- Se $\int_\Omega g(x)w(x)\ \mathrm{d}x = 0$ per qualche funzione continua e non negativa g, allora $g \equiv 0$ in Ω.

I termini w_j e x_j sono detti rispettivamente pesi e nodi.

In particolare tratteremo formule PI type, dove P indica *Positive weights* e I indica *Inside the domain*, ovvero per le quali i nodi sono interni alla regione Ω e i pesi di quadratura sono positivi.

Un criterio importante da considerare per la qualità delle formule di cubatura è il *grado di precisione algebrica*

Definizione 1.1. *(Grado di precisione, Radau 1880)*
Una formula

$$\int_a^b f(x)w(x)\,\mathrm{d}x \approx \sum_{i=1}^{N} w_i f(x_i) \tag{1.3}$$

ha grado di precisione almeno M se e solo se è esatta per tutti i polinomi f di grado inferiore o uguale a M, ovvero

$$\int_a^b p_M(x)w(x)\,\mathrm{d}x = \sum_{i=1}^{N} w_i\, p_M(x_i), \quad p_M \in P_M, \tag{1.4}$$

dove P_M è lo spazio dei polinomi di grado minore o uguale a M. Ha inoltre grado di precisione esattamente M se e solo se è esatta per ogni polinomio di grado inferiore o uguale a M, ed esiste un polinomio di grado $M+1$ per cui non lo sia.

Questa definizione, inizialmente valida in un intervallo, anche illimitato, è immediatamente generalizzabile ad un dominio di integrazione multivariato.

Definizione 1.2. *(Grado di precisione, dominio multivariato)*
Sia D un dominio di $\mathbb{R}^n$, $w(x)$ una funzione peso definita su D, e P_M^n è lo spazio dei polinomi di grado totale inferiore o uguale a M in n variabili. Una formula

$$\int_D f(x)w(x)\;\mathrm{d}x \approx \sum_{i=1}^{N} w_i f(x_i) \tag{1.5}$$

ha grado di precisione almeno M se e solo se è esatta per tutti i polinomi f di grado totale inferiore o uguale a M, ovvero

$$\int_D p_M(x)w(x)\;\mathrm{d}x = \sum_{i=1}^{N} w_i p_M(x_i), \quad p_M \in P_M^n \tag{1.6}$$

Ha inoltre grado di precisione esattamente M se e solo se è esatta per ogni polinomio di grado totale inferiore o uguale a M, ed esiste un polinomio di grado totale $M+1$ per cui non lo sia.

1.2 Formule di cubatura di tipo Tchakaloff

La seguente versione del teorema di Tchakaloff [8] è un risultato fondamentale per estrarre da una discretizzazione sufficientemente densa di un insieme compatto una regola di cubatura di tipo PI con bassa cardinalità (non superiore alla dimensione dello spazio polinomiale P_n^2, essendo n il grado algebrico di esattezza)

Teorema 1.1. *Sia μ una misura positiva sul dominio compatto $D \subset \mathbb{R}^d$ e sia n un intero positivo. Allora esistono $\nu \leq \dim(P_d^n(D))$ punti $\{Q_j\} \in D$ e numeri reali positivi $\{w_j\}$ tali che*

$$\int_D p(P)\ \mathrm{d}\mu = \sum_{j=1}^{\nu} w_j p(Q_j) \quad \textit{per ogni } p \in P_d^n(D) \tag{1.7}$$

dove $P_d^n(D)$ è lo spazio dei polinomi d-varianti di grado non superiore a n, ristretto a D.

Con l'obiettivo di calcolare un insieme di nodi $\{Q_j\} \subset D$ e relativi pesi positivi, soddisfacendo il Teorema 1.1, un altro risultato chiave è il seguente teorema di Wilhelmsen in [9], che estende una proposizione di Davis [3]:

Teorema 1.2. *Sia F il prodotto lineare di funzioni continue, reali, linearmente indipendenti $\{f_k\}_{k=1,\dots,N}$ definite su un insieme compatto $D \subset \mathbb{R}^d$. Supponiamo che F soddisfi la condizione di Krein (cioè esiste almeno una $f \in F$ che non si annulla su D) e che L sia un funzionale lineare positivo su F, cioè $Lf > 0$ per ogni $f \in F$, $f \geq 0$ non si annulla ovunque in D. Se $\{P_i\}_{i=1}^{\infty}$ è un sottoinsieme ovunque denso di D, allora per I sufficientemente grande, l'insieme $X = \{P_i\}_{i=1}^{I}$ è un insieme di Tchakaloff, cioè*

$$Lf = \sum_{j=1}^{\nu} w_j f(Q_j), \quad \forall f \in F \tag{1.8}$$

dove $w_j > 0$ per ogni j e $\{Q_j\}_{j=1}^{\nu} \subset X \subset D$, con $\nu = \mathrm{card}(\{Q_j\}) \leq N$.

Notiamo che possiamo applicare questo teorema al caso della cubatura algebrica, ponendo $Lf = \int_D f(P)\ \mathrm{d}\mu$ (essendo μ una misura positiva sull'insieme compatto $D \subset \mathbb{R}^d$) e $F = P_d^n(D)$. Una tale formula può essere chiamata *"Tchakaloff-like algebraic cubature rule"*

1.3 Definizione del dominio di integrazione

Con lo scopo di individuare una discretizzazione sufficentemente densa di punti bivariati, verrà presentato un algoritmo il cui obiettivo é valutare la posizione di un punto rispetto ad una classe di poligoni curvilinei S, le cui frontiere sono particolari curve di Jordan.

A tal proposito risultano utili seguenti definizioni, riprese da [1] e [2].

Definizione 1.1. Dati $k \in \mathbb{N} \cup \{\infty\}$, una curva parametrizzata di classe $C^k \subset \mathbb{R}^n$ è una applicazione $\varphi : I \to \mathbb{R}^n$ di classe C^k, dove $I \subset \mathbb{R}$ è un intervallo. L'immagine di $\varphi(I)$ sarà detta *sostegno della curva.*

Definizione 1.2. Una curva $\varphi : I \to \mathbb{R}^n$ si dice:

- *semplice* se, comunque presi due punti distinti t_1 e t_2 di I, di cui almeno uno *interno* all'intervallo I, risulta $\varphi(t_1) \neq \varphi(t_2)$;
- *chiusa* se è definita in un intervallo chiuso e limitato $I = [a, b]$ con $\varphi(a) = \varphi(b)$;
- *piana* se $\varphi(I)$ è contenuta in un piano. In particolare, una curva piana ha parametrizzazione $\varphi : I \to \mathbb{R}^2$.

Definizione 1.3. Una curva C è una *curva di Jordan* se ammette una parametrizzazione $\varphi : I \to \mathbb{R}^n$ chiusa, semplice e piana.

Verranno trattate regole numeriche su poligoni curvilinei a tratti razionali spline, seguendo i dettagli introdotti in [7].

Consideriamo domini di Jordan $S \subset \mathbb{R}^2$:

1. il cui bordo ∂S è descritto da equazioni parametriche $x = \tilde{x}(t)$, $y = \tilde{y}(t)$, $t \in [a, b]$, $\tilde{x}, \tilde{y} \in C([a, b])$, $\tilde{x}(a) = \tilde{x}(b)$ e $\tilde{y}(a) = \tilde{y}(b)$;
2. per i quali esistono partizioni $\{I^{(k)}\}_{k=1,\dots,M}$ di $[a, b]$, e $\{I_j^{(k)}\}_{j=1,\dots,m_k}$ di ciascun $I^{(k)} \equiv [t^{(k)}, t^{(k+1)}]$, tali che le restrizioni di $\tilde{x}$ e $\tilde{y}$ su ciascun intervallo chiuso $I^{(k)}$ sono spline razionali rispetto ai sottointervalli $\{I_j^{(k)}\}_{j=1,\dots,m_k}$.

Adottiamo come notazione:

$$\tilde{x}(t) = \frac{u_{k,1}(t)}{v_{k,1}(t)}, \quad \tilde{y}(t) = \frac{u_{k,2}(t)}{v_{k,2}(t)}, \quad t \in I^{(k)} \tag{1.9}$$

essendo $u_{k,1}, u_{k,2}, v_{k,1}, v_{k,2}$ spline su $I^{(k)}$, condividendo gli stessi nodi e avendo grado, rispettivamente, $\eta_{k,1}, \eta_{k,2}, \delta_{k,1}, \delta_{k,2}$, $k = 1, \dots, M$.

Si noti che, poiché $\tilde{x}, \tilde{y} \in C([a,b])$, stiamo assumendo che i denominatori $v_{k,1}, v_{k,2}, k = 1, \ldots, M$ non siano mai nulli nell'intervallo chiuso $I^{(k)}$.

Osserviamo che, dati $V_k = (\tilde{x}(t_k), \tilde{y}(t_k)) \in \mathbb{R}^2$, $k = 1, \ldots, M$, $V_{M+1} = V_1$ i vertici di un tale dominio di Jordan S, allora $\partial S := \cup_{k=1}^{M} \widehat{V_k V_{k+1}}$ e ciascun lato curvilineo $\widehat{V_k V_{k+1}}$ può essere tracciato da una spline parametrica di grado δ_k, interpolando una sottosequenza ordinata di nodi $P_{1,k} = V_k, P_{2,k}, \ldots, P_{m_k-1,k}, P_{m_k,k} = V_{k+1}$ con una parametrizzazione adeguata che determina ciascun $I_j^{(k)}$ (e quindi ciascun $I^{(k)}$).

L'obiettivo è analizzare domini nei quali il bordo sia localmente una curva NURBS (Non-Uniform Rational B-Splines) [5] di grado p, cioè definita sul lato curvilineo $\widehat{V_k V_{k+1}}$ come

$$C(t) = \frac{\sum_{i=1}^{m_k} B_{i,p}(t) \lambda_{i,k} P_{i,k}}{\sum_{i=1}^{m_k} B_{i,p}(t) \lambda_{i,k}}, \quad t \in [t^{(k)}, t^{(k+1)}] \tag{1.10}$$

dove:

- $\{P_{i,k}\}_{i=1}^{m_k}$ sono i punti di controllo,
- $\{\lambda_{i,k}\}_{i=1}^{m_k}$ sono i pesi;
- $\{B_{i,p}\}_{i=1}^{m_k}$ sono le *B-spline basis functions* di grado p definite sul vettore di nodi non periodico (e non uniforme)

$$U = \left\{ \underbrace{t^{(k)}, \ldots, t^{(k)}}_{p+1}, t_{p+1}^{(k)}, \ldots, t_{m_k-(p+1)}^{(k)}, \underbrace{t^{(k+1)}, \ldots, t^{(k+1)}}_{p+1} \right\}$$

  con $t_{p+j}^{(k)} \leq t_{p+j+1}^{(k)}$, $j = 1, ..., m_k - 1$.

Per comprendere meglio la struttura dei domini analizzati procediamo con le seguenti definizioni

Definizione 1.3. *Sia $[a,b] \subset \mathbb{R}$ un intervallo chiuso e limitato, e sia $a = x_0 < x_1 < \ldots < x_n = b$. Una spline di grado m (o di ordine $m+1$) è una funzione in $C^{m-1}([a,b])$ che in ogni intervallo $[x_i, x_{i+1}]$, con $i = 0, \ldots, n-1$, è un polinomio di grado m.*

Definizione 1.4. *Consideriamo $U = (u_0, u_1, \ldots, u_m)$, una sequenza non decrescente di numeri reali, ovvero tale che $u_i \leq u_{i+1}$ per $i = 0, \ldots, m-1$. Gli elementi u_i sono detti nodi e la sequenza U è chiamata vettore dei nodi. Le B-spline basis functions di ordine p (ovvero di grado $p-1$) sono definite ricorsivamente come segue:*

$$B_{i,0}(u) = \begin{cases} 1 & \textit{se } u_i \leq u < u_{i+1} \\ 0 & \textit{altrimenti} \end{cases}$$

$$B_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} B_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} B_{i+1,p-1}(u).$$

Le NURBS sono quindi un particolare tipo di spline

- Non uniforme: I nodi del vettore dei nodi $U = (u_0, u_1, \ldots, u_m)$ possono essere distribuiti in modo non uniforme, ovvero gli intervalli tra i nodi non sono necessariamente uguali.

- Razionale: $C(t)$ è espresso come combinazione pesata dei punti di controllo $\{P_{i,k}\}_{i=1}^{m_k}$ mediante le *B-spline basis functions* $\{B_{i,p}\}_{i=1}^{m_k}$ moltiplicate per i rispettivi pesi $\{\lambda_{i,k}\}_{i=1}^{m_k}$. Poiché sia il numeratore che il denominatore del rapporto sono somme pesate, il risultato complessivo è un numero razionale.

Capitolo 2

Routine in-domain

2.1 Algoritmo

L'algoritmo che verrà delineato in questo paragrafo ha la funzione di determinare se un dato punto $P = (P_x, P_y) \in \mathbb{R}^2$ appartiene o meno a un dominio di Jordan $S \subset \mathbb{R}^2$.

Per valutare se un dato punto $P \in \mathbb{R}^2$ è interno o esterno alla frontiera ∂S del dominio, è utile richiamare la strategia in-domain basata sul seguente teorema:

Teorema 2.1 (Teorema della curva di Jordan)**.** *Un punto $P \in \mathbb{R}^2$ appartiene a un dominio di Jordan $S \subset \mathbb{R}^2$ se e solo se, dato un punto $P^* \notin S$, il segmento $\overline{P^*P}$ interseca il bordo ∂S un numero dispari $c(P)$ di volte.*

Potrebbero però esserci casi patologici, per questo inizialmente richiederemo che $\overline{P^*P}$ non contenga alcun punto critico o lato verticale. Diverse problematiche sorgono infatti se:

- $\overline{P^*P}$ tocca un vertice o include un segmento di ∂S,
- se il bordo ∂S contiene punto critico $S = (\widetilde{x}(t), \widetilde{y}(t))$ dove:

$$\lim_{t\to\gamma^-} \widetilde{x}'(t) \lim_{t\to\gamma^+} \widetilde{x}'(t) < 0,$$

  Ovvero se si verifica localmente una svolta verticale del bordo da sinistra a destra (o viceversa da destra a sinistra).

In questi casi, se l'analisi in-domain del punto P viene eseguita mediante segmenti verticali $\overline{P^*P}$, allora il suddetto teorema di Jordan non può essere applicato direttamente.

Supposto quindi che $\overline{P^*P}$ non intersechi un vertice o un punto critico né contenga un segmento di ∂S, con lo scopo di calcolare il numero di attraversamenti $c(P)$, cioè il numero di volte in cui il segmento verticale $\overline{P^*P}$ attraversa ∂S, definiamo R rettangolo cartesiano, cioè con lati paralleli agli assi cartesiani, che contiene strettamente S, P^* punto di ∂R che condivide la stessa ascissa P_x di $P = (P_x, P_y)$ ma ha ordinata strettamente inferiore di P_y. Il primo passo consiste nel coprire ∂S con l'unione di rettangoli $R_1, ..., R_L$ adatti. Ogni R_j, $j = 1, ..., L$ contiene una porzione di ∂S che non ha picchi ed è parametrizzata da due funzioni razionali, $(\widetilde{x}(t), \widetilde{y}(t))$ sono infatti localmente il rapporto di due polinomi. Una volta ottenuti questi R_j, $j = 1, ..., L$, otterremo una valutazione del numero di attraversamenti $c(P)$ che richiede al massimo la soluzione di alcune equazioni polinomiali.

Per determinare numericamente i rettangoli R_j procediamo

- Osservando che, poiché $\tilde{x}$ e $\tilde{y}$ sono spline razionali in ciascun $I^{(k)}$, allora ci sono $I_j^{(k)} = [t_j^{(k)}, t_{j+1}^{(k)}] \subseteq I^{(k)}$, $k = 1, ..., M$, $j = 1, ..., m_k - 1$, dove la restrizione di $\tilde{x}$ e $\tilde{y}$ a $I_j^{(k)}$ sono funzioni razionali, cioè

$$\tilde{x}(t) = \frac{u_{k,j,1}(t)}{v_{k,j,1}(t)}, \quad \tilde{y}(t) = \frac{u_{k,j,2}(t)}{v_{k,j,2}(t)}, \quad t \in I_j^{(k)} \tag{2.1}$$

  dove $u_{k,j,1}$, $u_{k,j,2}$, $v_{k,j,1}$, $v_{k,j,2}$ sono polinomi su $I_j^{(k)}$, di grado rispettivamente $\eta_{k,1}$, $\eta_{k,2}$, $\delta_{k,1}$, $\delta_{k,2}$ (notare che non dipendono da j ma solo dal grado locale delle spline $u_{k,1}$, $u_{k,2}$, $v_{k,1}$, $v_{k,2}$ in $I^{(k)}$).

- Definiamo, se $\tilde{x}'$ cambia segno in $(t_j^{(k)}, t_{j+1}^{(k)})$, $N_j^{(k)} = \{t_i^{(j,k)}\}_{i=1,...,l_{j,k}}$ l'insieme di tutti i punti $t_i^{(j,k)} \in (t_j^{(k)}, t_{j+1}^{(k)})$ tali che $\tilde{x}'(t_i^{(j,k)}) = 0$ (notare che la restrizione di $\tilde{x}$ a $I_j^{(k)}$ è una funzione razionale con il denominatore non nullo in nessun punto, quindi $\tilde{x}'$ esiste), altrimenti poniamo $N_j^{(k)} = \emptyset$. Successivamente, sia $T^{(j,k)} = \{t_j^{(k)}, t_{j+1}^{(k)}\} \cup N_j^{(k)}$, dove supponiamo che i suoi elementi, $T_i^{(j,k)}$, siano in ordine crescente. Osserviamo che essendo $\tilde{x}(t) = \frac{u_{k,j,1}(t)}{v_{k,j,1}(t)}$, $t \in I_j^{(k)}$, la determinazione dell'insieme $N_j^{(k)}$ richiede la soluzione di un'equazione polinomiale. Più precisamente, dato che

$$\tilde{x}'(t) = \frac{u'_{k,j,1}(t)v_{k,j,1}(t) - u_{k,j,1}(t)v'_{k,j,1}(t)}{v_{k,j,1}^2(t)}, t \in I_j^{(k)}, \tag{2.2}$$

  e $v_{k,j,1}^2(t) \neq 0$ per ogni $t \in I_j^{(k)}$, otteniamo che $\tilde{x}'(t) = 0$ se e solo se

$$u'_{k,j,1}(t)v_{k,j,1}(t) - u_{k,j,1}(t)v'_{k,j,1}(t) = 0, \tag{2.3}$$

e di conseguenza la determinazione di $N_j^{(k)}$ richiede la soluzione di un'equazione polinomiale di grado $\eta_{k,1} + \delta_{k,1} - 1$.

- Definiamo i rettangoli $B_i^{(j,k)}$, chiamati *monotone boxes* in [6]:

$$B_i^{(j,k)} := [\min_{t \in I_i^{(j,k)}} \tilde{x}(t), \max_{t \in I_i^{(j,k)}} \tilde{x}(t)] \times [\min_{t \in I_i^{(j,k)}} \tilde{y}(t), \max_{t \in I_i^{(j,k)}} \tilde{y}(t)],$$

  Dove $I_i^{(j,k)} := [T_i^{(j,k)}, T_{i+1}^{(j,k)}]$

- Notiamo che, per definizione, se $N_j^{(k)} = \emptyset$, allora c'è solo la *monotone box* $B_1^{(j,k)}$. Poiché $\tilde{y}$ ristretta a $[T_i^{(j,k)}, T_{i+1}^{(j,k)}]$ è una funzione razionale, la valutazione di:

$$\min_{t \in [T_i^{(j,k)}, T_{i+1}^{(j,k)}]} \tilde{y}(t), \max_{t \in [T_i^{(j,k)}, T_{i+1}^{(j,k)}]} \tilde{y}(t)$$

  Può essere facilmente determinata una volta ottenuta la derivata del polinomio $\tilde{y}'$, calcolando le sue radici in $[T_i^{(j,k)}, T_{i+1}^{(j,k)}]$

Alla fine di questa procedura abbiamo determinato $I_i^{(j,k)}$ in modo che le restrizioni di $\tilde{x}$ e $\tilde{y}$ su ciascun $I_i^{(j,k)} \subseteq [a, b]$ siano funzioni razionali e $\tilde{x}$ sia una funzione monotona (e quindi non ci siano punti di flesso di ∂S all'interno di ogni *monotone box* $B_i^{(j,k)}$). Definito l'insieme $\mathcal{B} := \{B_i^{(j,k)}\}$ verifichiamo l'appartenenza di $P = (P_x, P_y)$ al dominio S. Per prima cosa determiniamo l'insieme $B(P)$ che tiene conto di tutte le *monotone boxes* B_l tali che $\overline{P^*P} \cap B_l \neq \emptyset$ e che possono contribuire alla valutazione di $c(P)$:

$$B(P) = \{B = [\alpha_1, \beta_1] \times [\alpha_2, \beta_2] \in \mathcal{B} : P_x \in [\alpha_1, \beta_1], P_y \geq \alpha_2\}$$

Consideriamo la *monotone box* $B_l = [\alpha_1^{(l)}, \beta_1^{(l)}] \times [\alpha_2^{(l)}, \beta_2^{(l)}] \in B(P)$. Se $P_y > \beta_2^{(l)}$, allora il punto P non appartiene alla *monotone box* B_l e necessariamente il segmento $\overline{P^*P}$ attraversa il bordo ∂S una volta in B_l e sotto P, a causa della monotonicità di $\tilde{x}$ in B_l. Altrimenti, abbiamo che $P \in B_l$. Poiché per ipotesi $\overline{P^*P}$ non contiene un punto critico o un segmento verticale del bordo e B_l include una certa porzione di ∂S descritta parametricamente da due funzioni razionali, diciamo $\tilde{x}|_{Bl}$, $\tilde{y}|_{Bl}$, con argomenti nell'intervallo $I|_{Bl} \subseteq [a, b]$, in cui $\tilde{x}|_{Bl}$ è monotona e tale che $P_x \in \tilde{x}|_{Bl}(I|_{Bl})$, necessariamente esiste una radice unica $t^* \in I|_{Bl}$ dell'equazione polinomiale $\tilde{x}|_{Bl}(t) = P_x$. In particolare, essendo $\tilde{x}|_{Bl}(t) = \frac{u(t)}{v(t)}$, per certi polinomi u, v, allora t^* è la soluzione unica in $I|_{Bl}$ dell'equazione polinomiale $u(t) - P_x \cdot v(t) = 0$. Successivamente,

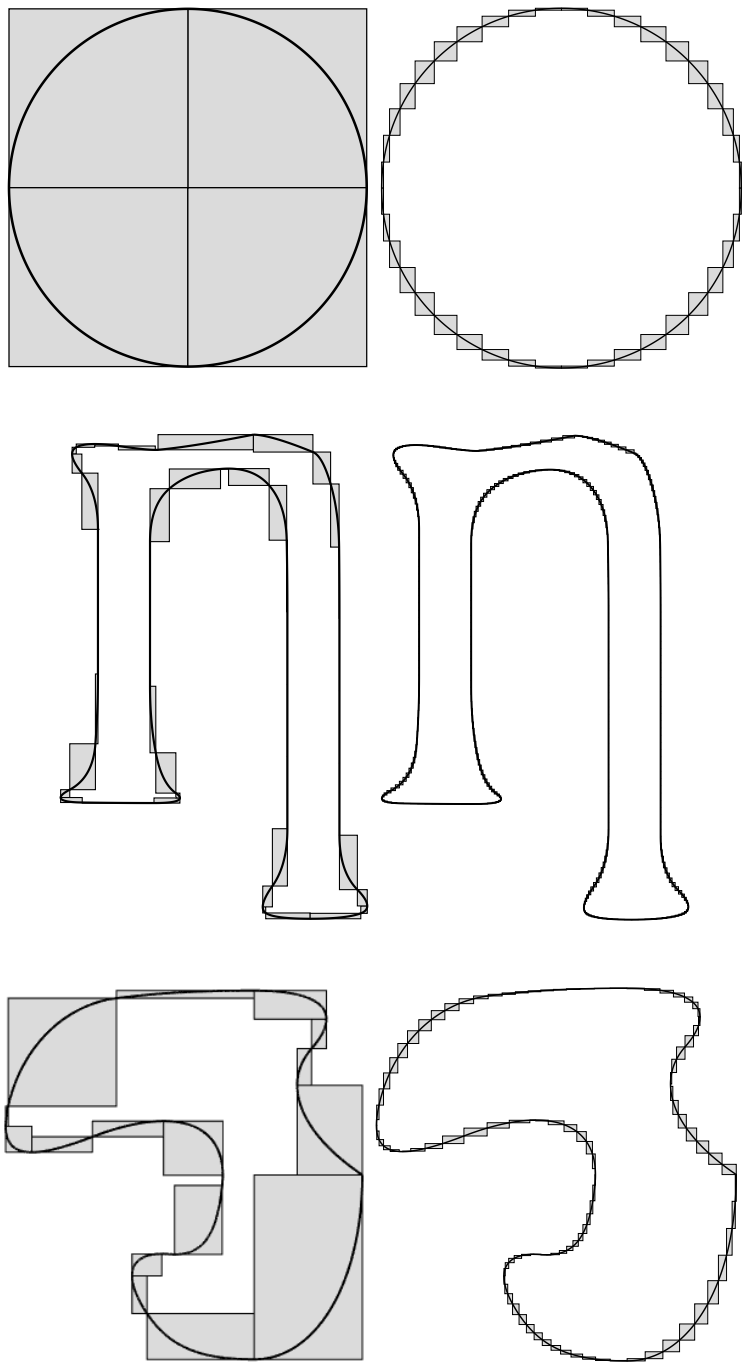


Figura 2.1: *Domini con bordo di tipo NURBS con le relative monotone boxes. A sinistra una copertura con un numero esiguo di monotone boxes, a destra con un numero maggiore.*

- Se $\tilde{y}(t^*) < P_y$ allora il segmento $\overline{P^*P}$ attraversa il bordo ∂S una volta nella *monotone box*, sotto P;
- Se $\tilde{y}(t^*) > P_y$ allora il segmento $\overline{P^*P}$ non attraversa il bordo ∂S in B, sotto P;
- Se $\tilde{y}(t^*) = P_y$ allora P è sul bordo ∂S.

Come risultato, nelle ipotesi menzionate sopra, contando tutti questi attraversamenti, determiniamo se un punto P è all'interno o all'esterno del dominio S.

Osservazione 2.1 Si può coprire il bordo con un numero maggiore di *monotone boxes* suddividendo ulteriormente ogni $I_i^{(j,k)}$ in un certo numero di sottointervalli $I_{i,\tau}^{(j,k)} = [a_{i,\tau}^{(j,k)}, b_{i,\tau}^{(j,k)}]$. In questo modo i box diventano chiaramente più piccoli e si ottiene un'approssimazione più fine del bordo del dominio. Dato un punto $P = (x, y)$, i box pertinenti corrispondono alle quadruple

$$\ell = (j, k, i, \tau) : a_{i,\tau}^{(j,k)} \leq x \leq b_{i,\tau}^{(j,k)}, \quad y \geq c_{i,\tau}^{(j,k)}$$

dove

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

D'altra parte, più piccoli sono i box, minore è la probabilità che un punto casuale si trovi all'interno di un box, e quindi possiamo ridurre sostanzialmente il numero di equazioni da risolvere tramite radici per trovare le intersezioni e velocizzare l'intera procedura, quando un insieme molto grande di punti deve essere localizzato. In pratica, tuttavia, esiste una soglia adeguata per il numero di sub-box, oltre la quale il raffinamento non è più conveniente (una tale soglia può essere determinata approssimativamente sperimentalmente).

2.2 Winding-algorithm

Rimane da stabilire se il punto P appartiene al dominio S nei casi patologici in cui il segmento $\overline{P^*P}$ interseca un vertice, un punto critico o contiene un segmento verticale di ∂S. Per risolvere questo problema, si utilizza il *winding algorithm*, una procedura che calcola, attraverso una *Gauss-Legendre shifted rule* di sufficiente grado di precisione, il *winding number* $wind(P, \tilde{x}, \tilde{y}) \in \mathbb{Z}$. Se tale quantità sarà dispari allora il punto apparterrà a S, altrimenti non sarà contenuto all'interno del dominio.

Definizione 2.1. *(Winding Number)*
Sia $\gamma : [a, b] \subseteq \mathbb{R} \to \mathbb{C}$ un cammino chiuso, e sia $P \notin \gamma$. Il Winding Number di γ rispetto a P è

$$W(\gamma, P) := \frac{1}{2\pi i} \int_\gamma \frac{1}{z - P}\, dz \tag{2.4}$$

Il *Winding Number* di γ rispetto a P rappresenta il numero di avvolgimenti di γ attorno a P e sarà quindi un numero intero

Proposizione 2.1 Il Winding Number di un cammino chiuso è una funzione a valori in $\mathbb{Z}$.

Dimostrazione. Osservo che, dati due cammini chiusi γ_1 e γ_2 omotopi e tali per cui $P \notin \gamma_1$ e $P \notin \gamma_2$, vale $W(\gamma_1, P) = W(\gamma_2, P)$; infatti $w = \frac{1}{z-P}$ è chiuso in $\mathbb{C} \setminus \{0\}$ e da ciò ne consegue che $\oint_{\gamma_1} \omega = \oint_{\gamma_2} \omega$ essendo γ_1 e γ_2 omotopi e chiusi.

Posso ora considerare i cammini chiusi come cammini omotopi a un cammino γ^k che percorre una circonferenza di raggio unitario centrata in P k-volte, $k \in \mathbb{Z}$, per quanto precedentemente dimostrato varrà quindi:

$$W(\gamma, P) = W(\gamma^k, P) \tag{2.5}$$

Resta ora da calcolare $W(\gamma^k, P)$. Consideriamo γ_k, un cammino chiuso che si avvolge k volte attorno a P, essendo γ_k una circonferenza di raggio unitario centrato in P definisco la parametrizzazione di γ_k

$$z(t) = P + e^{it}, \text{dove } t \in [0, 2\pi k]$$

L'integrale di $\frac{1}{z-P}$ lungo γ_k, essendo $\mathrm{d}z = ie^{it}\, \mathrm{d}t$ il differenziale $\mathrm{d}z$, è quindi:

$$\oint_{\gamma_k} \frac{1}{z - P}\, \mathrm{d}z = \oint_{\gamma_k} \frac{1}{e^{it}} \cdot ie^{it}\, \mathrm{d}t = \oint_{\gamma_k} i\, \mathrm{d}t = i \cdot (2\pi k) = 2\pi i \cdot k.$$

Pertanto:

$$W(\gamma^k, P) = \frac{1}{2\pi i} \int_\gamma^k \frac{1}{z - P}\, \mathrm{d}z = \frac{2\pi i \cdot k}{2\pi i} = k$$

Da cui

$$W(\gamma, P) = k \in \mathbb{Z}.$$

□

Proposizione 2.2 Sia $\gamma : [a, b] \to \mathbb{R}^2$ una curva chiusa, e sia $P = (P_x, P_y) \notin \gamma$. Allora

$$W(\gamma, P) = \frac{1}{2\pi} \oint_\gamma \frac{(x - P_x)\ \mathrm{d}y - (y - P_y)\ \mathrm{d}x}{(x - P_x)^2 + (y - P_y)^2}.$$

Dimostrazione. Dati $z = x + iy$ e $P = P_x + iP_y$, sostituiamo:

$$W(\gamma, P) := \frac{1}{2\pi i} \oint_\gamma \frac{1}{z - P}\ \mathrm{d}z = \frac{1}{2\pi i} \oint_\gamma \frac{1}{(x + iy) - (P_x + iP_y)}\ \mathrm{d}(x + iy)$$

$$= \frac{1}{2\pi i} \oint_\gamma \frac{\mathrm{d}x + i\ \mathrm{d}y}{(x - P_x) + i(y - P_y)} \cdot \frac{(x - P_x) - i(y - P_y)}{(x - P_x) - i(y - P_y)}$$

Separiamo parte reale e immaginaria

$$\frac{1}{2\pi i} \oint_\gamma \frac{\mathrm{d}x \cdot (x - P_x) + \mathrm{d}y \cdot (y - P_y)}{(x - P_x)^2 + (y - P_y)^2} + \frac{i}{2\pi i} \oint_\gamma \frac{\mathrm{d}y \cdot (x - P_x) - \mathrm{d}x \cdot (y - P_y)}{(x - P_x)^2 + (y - P_y)^2}.$$

La parte immaginaria dell'integrale si annulla perché l'integrale di dx su un percorso chiuso è zero. Otteniamo quindi:

$$W(\gamma, P) = \frac{1}{2\pi} \oint_\gamma \frac{(x - P_x)\ \mathrm{d}y - (y - P_y)\ \mathrm{d}x}{(x - P_x)^2 + (y - P_y)^2}.$$

□

Viste le cosniderazioni precedenti possiamo quindi definire $wind(P, \tilde{x}, \tilde{y}) \in \mathbb{Z}$:

$$wind(P, \tilde{x}, \tilde{y}) := \frac{1}{b - a} \int_a^b \frac{\tilde{y}'(t)(\tilde{x}(t) - P_x) - \tilde{x}'(t)(\tilde{y}(t) - P_y)}{(\tilde{x}(t) - P_x)^2 + (\tilde{y}(t) - P_y)^2}\ \mathrm{d}t.$$

la valutazione di $wind(P, \tilde{x}, \tilde{y})$ puó essere difficile quando P è vicino al bordo, ma in generale non è necessario calcolare tale quantità con alta precisione, considerando che $wind(P, \tilde{x}, \tilde{y})$ è un numero intero (per le proprietà precedentemente dimostrate).

Osservazione 2.2 Quando un punto $P = (P_x, P_y)$ ha ascissa P_x vicina a quella di un punto di inversione o di un lato verticale, è buona norma utilizzare il *winding-algorithm*. In tal senso, nell'implementazione dell'algoritmo *in-domain*, per verificare se un punto $P = (P_x, P_y)$ appartiene al bordo ∂S, abbiamo risolto un equazione polinomiale $u(t^*) - P_x \cdot v(t^*) = 0$ e poi verificato che la sua soluzione unica t^* soddisfi $\tilde{y}(t^*) = P_y$. Considerando gli errori numerici però, possiamo solo stabilire che un punto è molto vicino a ∂S, cioè $|\tilde{y}(t^*) - P_y|$ è inferiore a una tolleranza fissata dall'utente.

2.3 Domini forati

Consideriamo ora il caso di domini forati. A tale scopo definiamo il dominio S come differenza di due poligoni curvilinei, S_1 e S_2, tali che $S_2 \subset S_1$ ovvero $S = S_1 \setminus S_2$. In altri termini, il dominio S rappresenta il dominio S_1 forato dal poligono S_2, dove S_2 funge da foro.

Un punto generico $P = (P_x, P_y)$ è interno al dominio S solo se è interno al dominio S_1 e, contemporaneamente, esterno a S_2.

Nelle routine che implementeremo in questa tesi, il dominio S è descritto da un *cell array* che contiene i vari domini $\{S_i\}_{i=1,\dots,n}$ da cui è generato. Un algoritmo può accedere in ordine ad ogni elemento di tale *cell array*, compiendo una valutazione su ciascuno di essi rispetto ad un punto P o ad un insieme di punti $\{P_i\}_{i=1,\dots,k}$. Le valutazioni di appartenenza o meno ad ogni dominio S_i sono poi memorizzate nel vettore `in_all = []` secondo la seguente convenzione:

i) `in_all(j) = 1` se P_j appartiene a S_i

ii) `in_all(j) = 0` se P_j non appartiene a S_i

e sommate alle valutazioni precedenti al variare del dominio S_i. Al termine di tale processo, il vettore `in_all` presenta in posizione k un numero intero; se tale valore è dispari, allora il punto P_k è interno al dominio complessivo S, altrimenti è esterno.

Per quanto riguarda l'appartenenza alla frontiera, se la valutazione relativa ad un punto $P_k = P(k,:)$ è `on(k) = NaN` per un qualsiasi dominio S_i tra quelli che generano S, allora verrà memorizzato `on_all(k) = NaN`. Infatti, un punto vicino alla frontiera di un qualsiasi S_i è vicino alla frontiera del dominio S.

In Figura 2.2 abbiamo raffigurato 4 domini aventi geometrie complicate e molteplicemente connesse, i cui bordi sono di tipo NURBS. In ogni esperimento abbiamo considerato una nuvola di punti random nella bounding box di ogni dominio. In rosso abbiamo rappresentato i punti che l'algoritmo giudica appartenenti al dominio forato, mentre in ciano tutti i rimanenti. Risulta evidente visivamente la correttezza delle scelte operate.

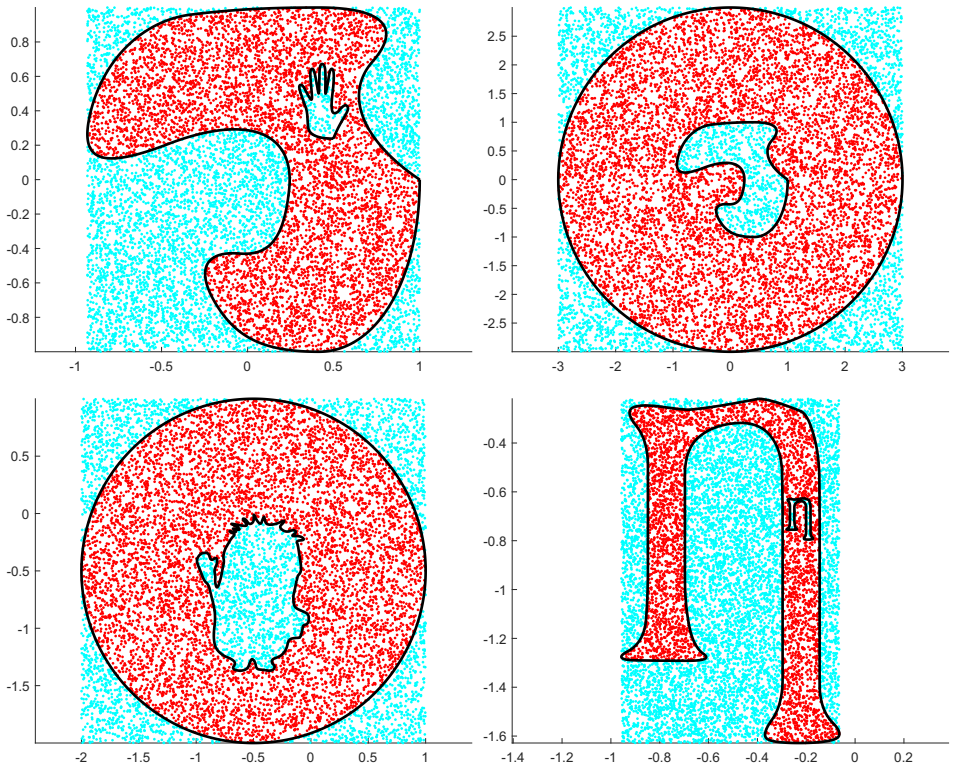


Figura 2.2: *Posizione di generici set di punti rispetto dei domini forati con bordi NURBS; i punti interni sono rappresentati in rosso mentre quelli esterni in ciano.*

Capitolo 3

Cubatura

3.1 Costruzione di regole di cubatura

Lo scopo di questa sezione è di mostrare come estrarre da una discretizzazione sufficientemente densa di un insieme compatto una regola di cubatura di tipo PI con bassa cardinalità (non superiore alla dimensione dello spazio polinomiale P_n^d, essendo n il grado algebrico di esattezza).

Applicando il Teorema 1.2 al nostro caso, cioè $D = S$, e supponendo di avere un insieme di Tchakaloff X a disposizione, data una qualsiasi base polinomiale $\{\varphi_j\}$ di P_n^d, procediamo:

- definendo la matrice di Vandermonde:

$$V = V_n(X) = (\varphi_j(P_i))_{i,j \in \mathbb{R}^{I \times N}}, \tag{3.1}$$

  dove P_i rappresentano i nodi nel dominio di integrazione S;

- denotando con $\gamma = \{\gamma_j\}$ il vettore dei momenti

$$\gamma_j = \int_S \varphi_j(P) \, \mathrm{d}\mu, \quad j = 1, \ldots, N; \tag{3.2}$$

- osservando che il sistema dei momenti (sottodeterminato) $N \times I$

$$V^T u = \gamma, \tag{3.3}$$

  ha una soluzione sparsa e non negativa u, le cui componenti non nulle (cioè, i pesi $\{w_j\} = \{u_i > 0\}$) sono al massimo $N = \dim(P_n^d)$. Inoltre, i nodi $\{Q_j\} \subset X$ sono determinati dagli indici delle componenti non nulle di u;

- determinando il vettore u, soluzione di 3.3, mediante l'algoritmo iterativo di Lawson-Hanson, che cerca automaticamente una soluzione sparsa del problema ai minimi quadrati non negativi

$$\min_{u \geq 0} \|V^T u - \gamma\|^2. \tag{3.4}$$

3.2 Implementazione delle regole di cubatura

Descriviamo quindi ora la procedura che determina una formula di cubatura algebrica di tipo Tchakaloff con nodi nel dominio di integrazione S, avente grado di esattezza algebrica (ADE) n su S, ovvero tale che:

$$\int_S p(x,y)\ \mathrm{d}x\ \mathrm{d}y = \sum_{j=1}^{\nu} w_j p(Q_j), \tag{3.5}$$

e dove

$$\nu \leq N = \dim(P_n^2) = \frac{(n+1)(n+2)}{2} \tag{3.6}$$

per qualsiasi polinomio bivariato $p \in P_n^2$.

Assumiamo che $I_s^{(k)} = [t_s^{(k)}, t_{s+1}^{(k)}]$, $k = 1, \ldots, M$, $s = 1, \ldots, m_k - 1$, sia una partizione di $[a,b]$ e che $\partial S = \{(\tilde{x}(t), \tilde{y}(t)), t \in [a,b]\}$, dove le restrizioni di $\tilde{x}, \tilde{y} \in C([a,b])$ a ciascun $I_s^{(k)}$ sono funzioni razionali.

Come precedentemente introdotto, il primo passo consiste nel calcolare i momenti γ_j di una base polinomiale opportuna $\{\phi_j\}$ di grado totale n su S, ovvero:

$$\gamma_j = \int_S \phi_j(x,y)\ \mathrm{d}x\ \mathrm{d}y, \quad j = 1, \ldots, N. \tag{3.7}$$

Come in [6], nella nostra implementazione in MATLAB abbiamo adottato come base polinomiale $\{\phi_j\}$, $1 \leq j \leq N$, la base di Chebyshev ordinata lessicograficamente, di tipo prodotto, con elementi aventi grado totale al piú n, ovvero:

$$\phi_j(x,y) = T_{h_1}(\alpha_1(x)) \cdot T_{h_2}(\alpha_2(y)), \quad 0 \leq h_1 + h_2 \leq n \tag{3.8}$$

dove j rappresenta la posizione di (h_1, h_2) nell'ordinamento e $(x,y) \in \mathbb{R}^* = [a_1, b_1] \times [a_2, b_2]$, inoltre

- $T_k(\cdot) = \cos(k \arccos(\cdot))$ è il polinomio di Chebyshev di prima specie di grado k;

- se $R^* = [a_1, b_1] \times [a_2, b_2]$ è il riquadro di delimitazione di S (talvolta detto anche *bounding box*, che corrisponde al più piccolo rettangolo cartesiano, cioè con i lati paralleli agli assi, contenente il dominio S), allora $\alpha_i(s) = (2s - b_i - a_i)/(b_i - a_i)$, $s \in [a_i, b_i]$, $i = 1, 2$; Notiamo che essendo il bordo ∂S descritto da una curva NURBS, allora R^* sarà il rettangolo cartesiano più piccolo che contiene l'involucro convesso dei punti di controllo della NURBS.

La scelta di questa base deriva dalla necessità di evitare una situazione estrema di mal condizionamento delle matrici di Vandermonde nella base standard dei monomi. In virtù del teorema di Gauss-Green:

$$\gamma_j = \int_S \phi_j(x, y)\ \mathrm{d}x\ \mathrm{d}y = \int_{\partial S} \Psi_j(x, y)\ \mathrm{d}y \tag{3.9}$$

Dove

$$\Psi_j(x, y) = \int \phi_j(x, y)\ \mathrm{d}x = T_{h_2}(\alpha_2(y)) \int T_{h_1}(\alpha_1(x))\ \mathrm{d}x \tag{3.10}$$

Osserviamo che:

$$\int T_0(\alpha_1(x))\ \mathrm{d}x = x,$$

$$\int T_1(\alpha_1(x))\ \mathrm{d}x = \frac{b_1 - a_1}{4} \cdot \alpha_1^2(x),$$

$$\int T_h(\alpha_1(x))\ \mathrm{d}x = \frac{b_1 - a_1}{2} \left(\frac{h}{h^2 - 1} T_{h+1}(\alpha_1(x)) - \frac{x}{h - 1} T_{h-1}(\alpha_1(x)) \right), \quad h \geq 2.$$

Se definiamo $P_{k,s} := (\tilde{x}(t_s^{(k)}), \tilde{y}(t_s^{(k)}))$ e $\widehat{P_{k,s}P_{k,s+1}}$ come l'arco di ∂S che unisce $P_{k,s}$ con $P_{k,s+1}$, otteniamo:

$$\gamma_j = \int_{\partial S} \Psi_j(x, y)\ \mathrm{d}y = \sum_{k,s} \int_{\widehat{P_{k,s}P_{k,s+1}}} \Psi_j(x, y)\ \mathrm{d}y$$

$$= \sum_{k,s} \int_{t_s^{(k)}}^{t_{s+1}^{(k)}} \Psi_j(\tilde{x}(t), \tilde{y}(t)) \tilde{y}'(t)\ \mathrm{d}t. \tag{3.11}$$

Essendo $\tilde{x}$ e $\tilde{y}$ funzioni razionali tale calcolo richiede l'uso di appropriate regole di Gauss-Legendre di ordine elevato [4], o routine adattive come la routine integrata in MATLAB `integral`.

Il secondo passo consiste nell'estrazione dei nodi e dei pesi positivi di una regola di cubatura algebrica di tipo Tchakaloff. A tal fine

- definiamo $P^0 = \emptyset$;
- introduciamo una sequenza di griglie tensoriali M_ℓ nel rettangolo $R^* := [a_1, b_1] \times [a_2, b_2]$ che contiene S e che diventa più fine all'aumentare di ℓ determinato dall'algoritmo in-domain;
- definiamo, all'iterazione ℓ-esima della procedura, l'insieme $P^\ell = P^{\ell-1} \cup (M_\ell \cap S)$, che include i punti delle griglie precedenti e dell'attuale, tutti appartenenti al dominio di integrazione S;
- applichiamo l'algoritmo di Lawson-Hanson per estrarre la formula di Tchakaloff con nodi $\{(x_i^{(\ell)}, y_i^{(\ell)})\}_{i=1}^{\nu_\ell}$ e pesi positivi corrispondenti $\{w_i^{(\ell)}\}_{i=1}^{\nu_\ell}$, dove $\nu_\ell \leq N$;
- verifichiamo infine che la regola ottenuta soddisfi:

$$\gamma_j^{(\ell)} = \sum_{i=1}^{\nu_\ell} w_i^{(\ell)} \phi_j(x_i^{(\ell)}, y_i^{(\ell)}), \quad j = 1, \ldots, N,$$

  e approssimi bene l'insieme dei momenti $\gamma = \{\gamma_j\}$, cioè

$$\|\gamma^{(\ell)} - \gamma\|_2 \leq \epsilon \tag{3.12}$$

  dove ϵ è una tolleranza fissata dall'utente;
- se la condizione 3.12 non è soddisfatta, iteriamo la procedura fino a quando non si verifica la convergenza o si raggiunge un numero massimo di iterazioni, fornendo un messaggio di errore in quest'ultimo caso.

È importante osservare che in aritmetica esatta, questa procedura termina finitamente poiché l'insieme P^ℓ diventa sufficientemente denso dopo un numero finito di iterazioni.

Queste regole algebriche di Tchakaloff di tipo PI sono ottimamente stabili (pesi positivi), cioè il numero di condizionamento della cubatura:

$$\text{cond}(\{w_i\}) = \frac{\sum_{i=1}^{\nu} |w_i|}{|\sum_{i=1}^{\nu} w_i|}$$

è uguale a 1, il miglior risultato possibile per regole con $ADE \geq 0$; hanno inoltre cardinalità particolarmente bassa (anche se solitamente non minima) e sono adatte quando il campionamento dell'integranda non è possibile fuori dal dominio.

Capitolo 4

Test numerici

4.1 Le routines `InRS` e `CubRS`

.

Per soddisfare le ipotesi introdotte nelle sezioni precedenti, dobbiamo

(1) Determinare le splines $u_{k,1}, u_{k,2}, v_{k,1}, v_{k,2}$ su $I^{(k)}$ definendo

$$\tilde{x}(t) = \frac{u_{k,1}(t)}{v_{k,1}(t)}, \quad \tilde{y}(t) = \frac{u_{k,2}}{v_{k,2}(t)}(t), \quad t \in I^{(k)},$$

in cui

$$u_{k,1}(t) = \sum_{i=1}^{m_k} B_{i,p}(t)\lambda_{i,k}(P_{i,k})_x, \quad u_{k,2}(t) = \sum_{i=1}^{m_k} B_{i,p}(t)\lambda_{i,k}(P_{i,k})_y,$$

$$v_{k,1}(t) = v_{k,2}(t) = \sum_{i=1}^{m_k} B_{i,p}(t)\lambda_{i,k};$$

(2) convertire dalla notazione B-spline a quella a tratti mediante il comando `fn2fm`.

Una volta descritte queste splines razionali in forma a tratti, possiamo determinare, come introdotto nelle sezioni precedenti, la routine in-domain e quella di cubatura.

La routine in-domain può essere riassunta nella seguente maniera.

Input: Insieme di punti $\mathcal{P}$, parametrizzazione del bordo NURBS $\Gamma(t) = (\alpha(t), \beta(t))$

1. Determinare le *monotone boxes* calcolando gli zeri di $\alpha'(t)$ e di $\beta'(t)$.

2. Calcolare una bounding box B per S utilizzando le *monotone boxes* estremali e raccogliere i punti critici del bordo, ovvero i punti di tangenza verticali ($\alpha' = 0$) o orizzontali ($\beta' = 0$) e i vertici di inversione in x o y.

3. Per tutti $P \in \mathcal{P} \cap B^c$: impostare $\text{inRS}(P) = 0$.

4. Per tutti $P \in \mathcal{P} \cap B$:

   (a) Calcolare $c(P)$, il numero di intersezioni del bordo con un raggio verticale da P nelle *monotone boxes* B_ℓ.

   (b) Se le intersezioni non includono punti critici allora:

   $$\text{inRS}(P) = c(P) \mod 2$$

   .

   (c) Altrimenti, ripetere il passo (a) con un raggio orizzontale.

   (d) Se le intersezioni non includono punti critici, allora:

   $$\text{inRS}(P) = c(P) \mod 2$$

   .

   (e) Altrimenti, calcolare $\text{inRS}(P) = \text{wind}(P)$.

Output: per tutti $P \in \mathcal{P}$, $\text{inRS}(P) = 1$ se $P \in \mathcal{P} \cap S$, $\text{inRS}(P) = 0$ altrimenti.

La routine di cubatura è invece implementata nel codice MATLAB `cubRS`, adeguatamente modificato in modo da considerare domini forati nel quale S è composto da una NURBS esterna S^{ext} e una NURBS interna S^{int}. In questo caso, procediamo come indicato nel seguente.

1. Definiamo una griglia sul rettangolo più piccolo $R^* = [a_1, b_1] \times [a_2, b_2]$ contenente il dominio S^{ext}; in particolare, per $ADE = n$, impostando $\tau = \lfloor n^{1.5} \rfloor$ e consideriamo i punti $P_{ij} = (x_i, y_j)$ dove

   $$x_i = a_1 + i\frac{a_2 - a_1}{\tau - 1}, \quad y_j = b_1 + j\frac{b_2 - b_1}{\tau - 1}, \quad 0 \le i, j \le \tau - 1,$$

   che corrispondono ad una griglia uniforme M_1, basata su τ punti equidistanti in ogni direzione. La scelta di τ è basata su esperimenti numerici, al fine di mantenere basso il numero di raffinamenti della griglia.

2. Determiniamo i punti di M_1 strettamente interni a S, diciamo $P_1 \subseteq M_1$, mediante l'algoritmo in-domain descritto nel Capitolo 2 e implementato nel codice MATLAB `inRS`. Iteriamo il procedimento per il dominio S^{int}, trovando così i punti P_2

3. Calcoliamo i momenti, prima su S^{ext} e successivamente su S^{int}, del prodotto di grado totale n-esimo della base di Chebyshev

$$\{T_{h_1}(\alpha_1(x))\ T_{h_2}(\alpha_2(y))\},$$

dove $(x, y) \in [a_1, b_1] \times [a_2, b_2]$, $0 \leq h_1 + h_2 \leq n$ e con

$$\alpha_i(s) = \frac{2s - b_i - a_i}{b_i - a_i}, \quad i = 1, 2,$$

mediante il teorema di Gauss-Green e regole di quadratura adeguate lungo gli archi di bordo NURBS. Definiamo poi il momento totale di S come la differenza tra i momenti di S^{ext} e S^{int}, sfruttando la linearità dell'integrale.

4. Estraiamo i nodi di una formula di tipo PI con $ADE = n$ da $P_1 - P_2$ e determiniamo i pesi relativi utilizzando l'algoritmo di Lawson-Hanson sul corrispondente problema NNLS.

Nel caso in cui la regola di cubatura non sia soddisfacente, cioè se la norma-2 dell'errore dei momenti è maggiore di una tolleranza fissata, diciamo $\varepsilon = 10^{-12}$, procediamo iterativamente definendo griglie uniformi più fini M_ℓ (aumentando il valore di τ, come $\tau_{\ell+1} = \lfloor \beta \tau_\ell \rfloor$ con ad esempio $\beta = 1.5$), determinando all'iterazione ℓ, con $\ell > 1$, quei punti appartenenti a S, diciamo $P_\ell \subseteq M_\ell$, e svolgendo il passo 4 dell'algoritmo sopra con l'insieme $\bigcup_{i=1}^{\ell} P_i$ al posto di P_1.

4.2 Test Numerici

Nella seguente sezione andremo a testare la *routine* precedentemente descritta su quattro domini molteplicemente connessi differenti.

Il primo test consiste nel valutare l'efficacia della *routine* nel determinare l'integrale di polinomi con coefficienti random e di grado uguale alla precisione algebrica della formula di quadratura. I polinomi sono quindi del seguente tipo

$$f = (a + b \cdot x + c \cdot y)^n \tag{4.1}$$

e vengono effettuate 100 prove con coefficienti a, b, c casuali distribuiti uniformemente in $(0, 1)$ e ADE $= n$ con $n = 2, 4, 6, 8, 10$. I valori di riferimento di questi integrali sono stati determinati utilizzando `Lineint`, una funzione che calcola pesi e nodi per un integrale di linea su domini di Jordan, il cui contorno è descritto in senso antiorario da spline razionali, ed è esatta per polinomi bivariati fino al grado di esattezza algebrica. In particolare, la funzione percorre gli archi del bordo del dominio rappresentati dalle spline razionali e calcola l'integrale di linea su ciascun arco. Nello specifico,

- se i denominatori delle spline sono costanti, utilizza la quadratura di Gauss-Legendre,
- altrimenti, utilizza una quadratura razionale che tiene conto dei denominatori variabili.

Procediamo con il test definendo quindi il dominio

a) S_1, caratterizzato dalla differenza di un dominio esterno e concavo a forma di τ, il quale bordo è determinato da un'unica curva NURBS di ordine 3 e con 12 punti di controllo, e da uno interno a forma di *mano*, il quale bordo è determinato da una NURBS con 35 punti di controllo;

b) S_2 costruito a sua volta come la differenza tra un dominio esterno convesso C_2, il quale bordo è definito da un'unica NURBS che costituisce una circonferenza centrata nell'origine e di raggio 3, e dal dominio τ precedentemente definito (che questa volta però rappresenta il foro e non più il dominio esterno);

c) S_3 è costituito da un dominio C_3 il cui bordo esterno è determinato da una NURBS che descrive un disco mentre quello interno da una NURBS a forma di *Minion*;

d) S_4 è definito come differenza tra due domini, η_1 e η_2, simili nella forma a una font che ricorda la lettera greca η, ma differenti per dimensione.

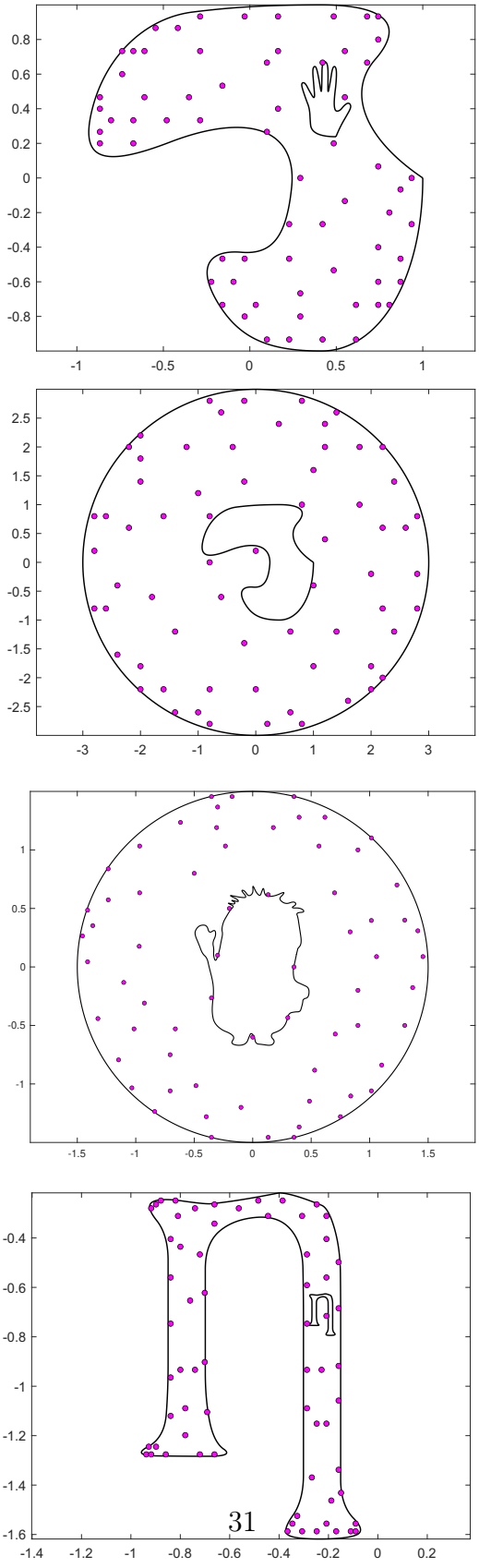
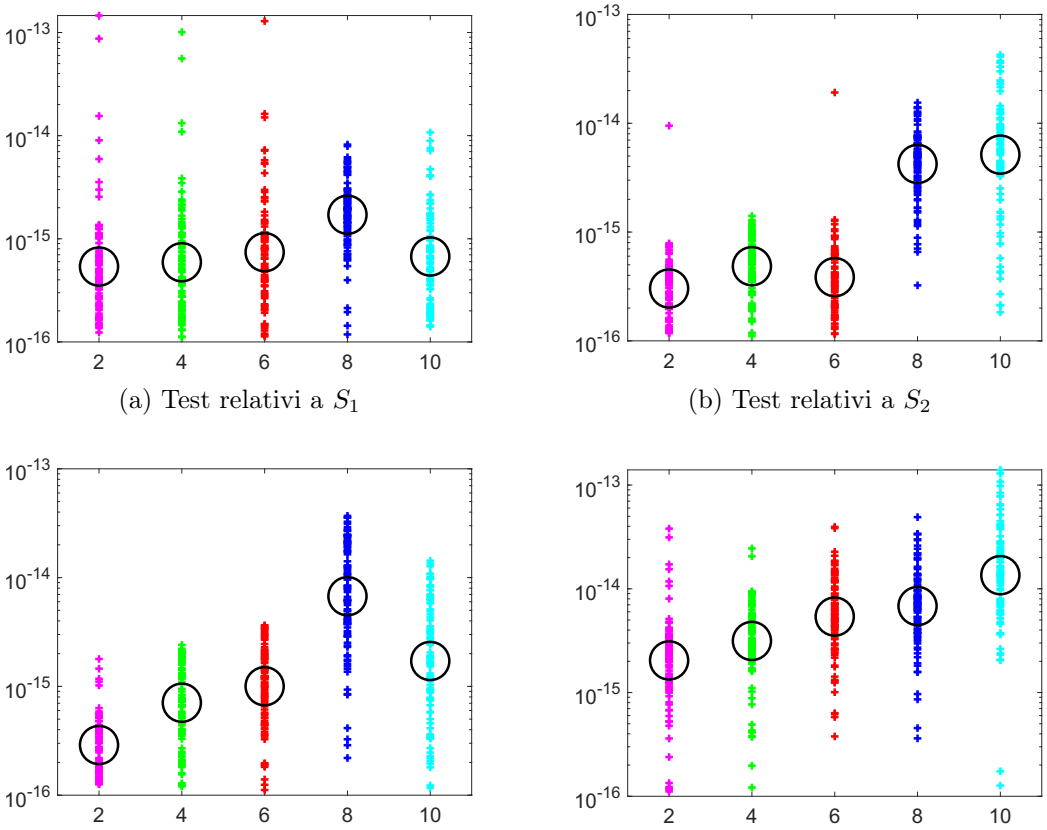


Figura 4.1: *Domini curvilinei S_i, con $i = 1, 2, 3, 4$, in cui testiamo la routine, con i nodi di integrazione evidenziati in magenta.*

Per analizzare la correttezza dei risultati proposti, studiamo l'errore relativo RE_k commesso dalla *routine* (in scala logaritmica) e l'errore logaritmico medio, definito come $10^{\left(\frac{1}{100}\sum_{k=1}^{100}\log(RE_k)\right)}$ dove RE_k è ottenuto da:

$$RE_k(n) = \left|\frac{I(f_k)_n - I(f_k)}{I(f_k)}\right| \quad \text{con} \quad I(f_k) \neq 0 \tag{4.2}$$

Dove indichiamo con $I(f_k)_n$ il valore fornito dalla *routine* e con $I(f_k)$ quello ottenuto utilizzando `Lineint`.



(a) Test relativi a S_1

(b) Test relativi a S_2

(c) Test relativi a S_3

(d) Test relativi a S_4

Figura 4.2: *Punti che rappresentano gli errori relativi RE_k (per $k = 1, \ldots, 100$) nelle cubature di polinomi casuali $f = (a + b \cdot x + c \cdot y)^{ADE}$ su S_i con $i = 1, 2, 3, 4$; cerchi che indicano l'errore logaritmico medio, definito come $10^{\left(\frac{1}{100}\sum_{k=1}^{100}\log(RE_k)\right)}$. Le ascisse rappresentano i valori di ADE della formula, ossia 2, 4, 6, 8, 10.*

Tabella 4.1: Valori di RE_{max} e RE_{log} per diversi valori di ADE per i domini S_1, S_2, S_3, S_4.

	S_1		S_2	
ADE	RE_{max}	RE_{log}	RE_{max}	RE_{log}
2	4.081e-14	6.428e-16	3.442e-15	3.330e-16
4	3.973e-14	5.280e-16	2.868e-15	5.542e-16
6	8.290e-15	7.779e-16	2.318e-15	3.577e-16
8	1.840e-14	1.916e-15	1.499e-14	4.562e-15
10	2.489e-14	6.476e-16	4.169e-14	6.680e-15

	S_3		S_4	
ADE	RE_{max}	RE_{log}	RE_{max}	RE_{log}
2	1.539e-14	2.847e-16	3.324e-12	2.654e-15
4	2.017e-14	4.294e-16	1.175e-13	3.490e-15
6	6.953e-15	1.805e-15	3.817e-14	4.421e-15
8	1.592e-14	1.212e-15	3.494e-14	6.217e-15
10	4.515e-14	5.652e-15	1.100e-13	1.458e-14

I test mostrano che, nonostante i momenti siano calcolati vicino alla precisione di macchina, c'è un lieve deterioramento dell'errore medio logaritmico per $n = 8$ e $n = 10$, mentre per gradi inferiori questo valore rimane generalmente inferiore a 10^{-14}.

Il prossimo test analizza l'errore relativo della *routine* nell'approssimare $\int_{S_i} f_k(x, y)\,\mathrm{d}x\,\mathrm{d}y$, con $i = 1, 2, 3, 4$, dove

$$f_1(x, y) = \exp\left(-(x^2 + y^2)\right), \tag{4.3}$$

$$f_2(x, y) = \left((x - x_0)^2 + (y - y_0)^2\right)^{11/2}, \tag{4.4}$$

$$f_3(x, y) = \left((x - x_0)^2 + (y - y_0)^2\right)^{1/2}, \tag{4.5}$$

con

- $(x_0, y_0) = (0, 0.4)$ per i test riguardanti S_1,

Tabella 4.2: *Errori relativi in S_i, con $i = 1, 2, 3, 4$, calcolati con $ADE = 2, 4, 6, 8, 10$*

	S_1			S_2		
ADE	f_1	f_2	f_3	f_1	f_2	f_3
2	3.0e-02	1.1e+00	1.4e-01	3.9e-01	7.4e-01	6.4e-02
4	1.9e-03	1.7e-01	3.6e-03	1.4e+00	1.1e-01	7.1e-03
6	1.1e-04	9.4e-03	1.0e-02	1.7e-01	3.3e-03	1.8e-03
8	2.9e-06	7.9e-04	6.3e-03	9.0e-02	1.8e-04	5.6e-05
10	4.3e-08	3.4e-05	9.3e-04	1.5e-02	4.9e-06	4.4e-03
	S_3			S_4		
ADE	f_1	f_2	f_3	f_1	f_2	f_3
2	9.0e-02	4.6e-01	5.5e-02	2.4e-02	3.2e-01	1.6e-02
4	3.7e-03	6.5e-02	1.0e-02	8.3e-04	1.1e-01	5.5e-03
6	1.9e-03	1.2e-02	4.0e-03	1.5e-05	7.2e-03	2.1e-03
8	4.2e-05	9.3e-05	7.7e-03	2.4e-07	3.9e-04	1.4e-03
10	1.1e-05	5.4e-06	6.0e-03	9.2e-10	1.6e-06	1.7e-04

.

- $(x_0, y_0) = (-0.2, -1.4)$ per i test riguardanti S_2 e S_4.
- $(x_0, y_0) = (0.5, -1)$ per i test riguardanti S_3,

I valori di riferimento di questi integrali sono ottenuti dalla stessa routine con ADE = 30.

Le funzioni f_k, $k = 1, 2, 3$, hanno diversa regolarità. Piú in dettaglio,

- f_1 è una funzione esponenziale, continua in tutto il dominio e in particolare $f_1 \in C^\infty(S_i)$ per $i = 1, 2, 3, 4$;
- f_2 rappresenta la distanza euclidea tra un punto (x, y) e il punto (x_0, y_0) elevata alla potenza undicesima; il fatto che $(x_0, y_0) \in S_i$, per $i = 1, 2, 3, 4$, rende (x_0, y_0) un punto singolare per la derivata sesta di f_2.
- f_3 rappresenta la distanza euclidea tra un punto (x, y) e il punto (x_0, y_0); il fatto che $(x_0, y_0) \in S_i$, per $i = 1, 2, 3, 4$, con f_3, rende (x_0, y_0) un punto singolare per la derivata prima di f_3.

Come prevedibile da un teorema di Jackson, in genere la qualità dell'approssimazione di f_3 è peggiore di quella per f_2, in quanto quest'ultima è più regolare.

É stata poi testata la rotine sui domini S_i precedentemente introdotti con lo scopo di analizzare, a parità di precisione n, il numero di nodi interni, esterni e precedenti alla compressione, prestando attenzione anche al numero di condizionamento, il residuo e al tempo impiegato dal calcolatore (CPU).

S_1	n	#	*in*	*all*	*cnd*	*moms*	*cpu*
	2	6	47	121	1	6e-16	1.3e-01
	4	15	47	121	1	1e-15	2.2e-01
	6	28	291	637	1	1e-15	4.5e-01
	8	45	235	484	1	3e-15	3.7e-01
	10	66	474	961	1	3e-15	4.6e-01
S_2	n	#	*in*	*all*	*cnd*	*moms*	*cpu*
	2	6	63	121	1	2e-14	1.6e-02
	4	15	63	121	1	1e-14	2.6e-02
	6	28	113	196	1	1e-14	4.0e-02
	8	45	309	484	1	3e-14	6.0e-02
	10	66	649	961	1	4e-14	1.3e-01
S_3	n	#	*in*	*all*	*cnd*	*moms*	*cpu*
	2	6	60	121	1	2e-15	3.1e-01
	4	15	60	121	1	4e-15	3.7e-01
	6	28	107	196	1	6e-15	4.4e-01
	8	45	290	484	1	1e-14	5.6e-01
	10	66	5182	7838	1	1e-14	7.1e-01
S_4	n	#	*in*	*all*	*cnd*	*moms*	*cpu*
	2	6	28	121	1	1e-16	1.4e-01
	4	15	100	377	1	2e-16	3.7e-01
	6	28	190	637	1	3e-16	2.5e-01
	8	45	527	1573	1	4e-16	3.0e-01
	10	66	995	3077	1	4e-16	4.4e-01

Tabella 4.3: Grado di precisione n della regola, cardinalità # dei nodi estratti, numero di nodi interni *in*, numero totale di punti utilizzati, indice di condizionamento *cnd*, residuo dei momenti *moms* e mediana del tempo di calcolo *cpu* su 50 test condotti sui domini S_i, con $i = 1, 2, 3, 4$.

Nella Tabella 4.3, variando il dominio e il grado di precisione ADE pari a n, illustriamo:

- la cardinalitá # della formula di quadratura;

- il numero di punti *in* del dominio S_k da cui viene estratta la formula di quadratura;
- il numero di punti *all* della bounding-box del dominio S_k da cui viene estratta la formula di quadratura;
- il numero di condizionamento *cnd* della formula di quadratura nel dominio S_k;
- il residuo sui momenti *moms*;
- la cputime *cpu* richiesta dall'algoritmo per ricavare la formula di quadratura.

Dai risultati dei test osserviamo che:

- la cardinalitá della formula di quadratura a grado di precisione n ha in genere $(n+1)(n+2)/2$ punti, ossia la dimensione dello spazio P_n^2;
- in tutti i test il numero di condizionamento $\mathrm{cond}(\{w_i\})$ della formula di cubatura;
$$\mathrm{cond}(\{w_i\}) = \frac{\sum_{i=1}^{\nu} |w_i|}{|\sum_{i=1}^{\nu} w_i|} \tag{4.6}$$
è, come previsto, pari a 1; ciò conferma che le regole hanno pesi positivi e quindi una stabilità ottimale;
- i valori *cmom* corrispondono ai residui sui momenti, ovvero alla quantità $cmom = \|\gamma - \gamma_{\mathrm{num}}\|_2$, dove $\gamma = \{\gamma_j\}$ sono i momenti della base di Chebyshev prodotto di grado n, mentre $\gamma_{\mathrm{num}} = \{\gamma_{\mathrm{num},j}\}$ consiste nella loro valutazione usando la regola di cubatura fornita dall'algoritmo; la loro corrispondenza vicina alla precisione di macchina conferma che le regole hanno grado di esattezza algebrico (numerico) pari a n;
- la colonna con i tempi di CPU richiesti dall'algoritmo per calcolare la regola mostra la mediana su 50 test; si osserva che i tempi contenuti, essendo al massimo dell'ordine di alcuni 10^{-1} secondi, a seconda della complessità del dominio, mostrando la rapiditá dell'algoritmo nel calcolare la formula di quadratura, per ADE non eccessivi, su domini particolarmente complicati.

Appendice A

Codici

I codici Matlab utilizzati, ed eventuali algoritmi di supporto, sono resi disponibili *open-source* sulla piattaforma *GITHUB* [10].

Qui di seguito, per facilitarne la consultazione, sono riportate le routines principali utilizzate per la realizzazione di questa tesi. Prima di tutto, verrà allegato il codice MATLAB di `cubRS`, opportunamente modificato per il caso di domini forati. Successivamente, verranno incluse le routine necessarie per l'esecuzione dei due test descritti in precedenza.

```
function [xyw,res,Z,Zin,cmom,bbox,itlevel] = cubRS(ade,
    structure_RS,...
     structure_RSH,extraction_type,Nbox,safe_mode)

%-------------------------------------
% OBJECT:
%-------------------------------------
% This routine computes an algebraic formula with positive
    weights and
% interior nodes with degree of exactness "ade" with respect
    to the
% Lebesgue/Legendre measure in a Jordan NURBS domain (or one
    more generally
% defined parametrically by piecewise rational splines) by
    the arrays of
% splines "SxN", "SxD", "SyN", "SyD", sharing the same
    checkpoints.
% This code works with boundaries defined by parametric
    splines or
% composite Bezier curves.
%-------------------------------------
% INPUT:
%-------------------------------------
```

```
% ade: Algebraic Degree of Exactness of the rule.
%
% structure_RS: array whose k-th element is a "structure"
   containing the
%    informations of the k-th portion of the boundary "dD" of
    the domain
%    "D".
%
%  * structure_RS.P: it is a N x 2 matrix of coordinates of
   control
%       points. The k-th control point is described as the k
   -th row of P.
%      In case it is a spline it represents the couples (X(i)
   ,Y(i)) that
%      the parametric splines x=x(t), y=y(t) must match.
%
%  * structure_RS.w: vector of weights; it is a row vector 1
   x N,
%       where "N" is the number of control points.
%
%  * structure_RS.knots: vector of knots;
%     1. in case the boundary is described by NURBS,
   following reference
%       [1], it is a row vector 1 x (N+L) of the form
%                   [k0*ones(1,L) u k1*ones(1,L)],
%       where "L" is the order of the NURBS and "u" is a
   vector of non
%       decreasing values, of length "N-L" where "N" is the
   number of
%       control points. Observe that
%
%      max(structure_RS{k}.knots) <= min(structure_RS{k+1}.
   knots).
%
%     2. in case the boundary is defined by a splines the
   variable
%       "structure_RS.knots" represents the break points of
   the spline.
%
%   Note that for understanding how to define the array of
   structures
%   "structure_RS", one can use the pertinent demos present
   in this
%   package.
%
%  * structure_RS.order: NURBS or spline order (i.e. "NURBS
   degree +1").
%
% A. The routine can be modified to study boundaries, that
```

```
   are defined by
% rational piecewise functions, in which the degree of the
   numerator and
% denominator is the same, or alternatively the denominator
   is equal to 1.
% This structure is typical of common splines or Composite
   Bezier curves.
%
% B. The variable "structure_RS" can be easily defined (see "
   demo").
%
% structure_RSH: as structure_RS but for the internal
   boundary;
%
% extraction_type: it sets how Tchakaloff rule is extracted
%      1: lsqnonneg, 2: lawsonhanson, 3: LHDM 4: \
%      This variable is not mandatory. If not declared or
   equal to "[]",
%      the default is LHDM.
%
% Nbox: not mandatory parameter, useful for indomain (default
   : 100). It is
%     a technical variable that if correctly set allows the
   indomain
%     routine to be faster.
%--------------------------------
% OUTPUT:
%--------------------------------
% xyw: 3-column array of cartesian coordinates of the nodes
   stored in the
%      first two columns and relative weights, stored in the
   third column.
%      Thus, the k-th node is "(xyw(k,1),xyw(k,2))" and has
   weight
%      "xyw(k,3)". Only this output variable is mandatory for
    the cubature
%      process.
%
% res: norm 2 of compressed moments vs moments, i.e.
%                           res=norm(V'*w-cmom);
%      where "V" is the Vandermonde matrix in the nodes of a
   certain Cheb.
%      tensorial basis of total degree "ade" and "cmom" is
   the vector of
%      the Chebyshev-Vandermonde moments;
%
% Zin: matrix M x 2 of M internal points from which the nodes
    are extracted
%       by Lawson-Hanson algorithm;
```

```
%
% ZL  : all analysed points from which only Zin are used in
   compression;
%
% cmom: vector of the Chebyshev-Vandermonde moments; as basis
    it is
%       considered the tensor product Chebyshev polynomials of
    total degree
%       "ade" with a suitable ordering, scaled respect to the
   aforementioned
%       rectangle "R".
%
% bbox: it is a vector that defines the smaller rectangle
   with side
%       parallel to axis, containing the domain.
%       Such rectangle is [bbox(1), bbox(2)] x [bbox(3), bbox
   (4)];
%
% itlevel: iterations made be the process; the routine
   defines at each
%       iteration a set of points from which Lawson-Hanson
   algorithm
%       attempts to extract the nodes. In case of failure, it
   adds a denser
%       set, trying again to find Tchakaloff nodes and weights
   . The higher
%       "itlevel", the higher is the cardinality of the set "Z
   " described
%       above.
%-----------------------------------
% Routines.
%-----------------------------------
% 1. chebmom (attached, as well as subroutines, except "
   legendre_10000.m");
% 2. inRS (external);
% 3. cvand (attached).
% 4. LHDM (external)
%-----------------------------------
% Copyrights.
%-----------------------------------
%% Copyright (C) 2007-2024 Chiara Zara, Alvise Sommariva,
   Marco Vianello.
%%
%% This program is free software; you can redistribute it and
   /or modify
%% it under the terms of the GNU General Public License as
   published by
%% the Free Software Foundation; either version 2 of the
   License, or
```

```
%% (at your option) any later version.
%%
%% This program is distributed in the hope that it will be
   useful,
%% but WITHOUT ANY WARRANTY; without even the implied
   warranty of
%% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
   the
%% GNU General Public License for more details.
%%
%% You should have received a copy of the GNU General Public
   License
%% along with this program; if not, write to the Free
   Software
%% Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
   02111-1307 USA
%%
%% Author:
%%          Chiara Zara
%%          Alvise Sommariva <alvise@euler.math.unipd.it>
%%          Marco Vianello   <marcov@euler.math.unipd.it>
%%
%% Date: July 8, 2024 -
%---------------------------------
% Dates
%---------------------------------
% First version: October 31, 2021;
% Checked: July 8, 2024.
%---------------------------------


% ..................... troubleshooting ...


% Extraction method: 1: lsqnonneg, 2: lawsonhanson, 3: LHDM
   4: \
if nargin < 4, extraction_type=3; end
if length(extraction_type) == 0, extraction_type=3; end

% Number of boxes inside each monotone box.
if nargin < 5, Nbox=100; end
if length(Nbox) == 0, Nbox=100; end

% Number of boxes inside each monotone box.
if nargin < 6, safe_mode=100; end
if length(safe_mode) == 0, safe_mode=1; end

% ....................... initialize .......
```

```
% Discretization parameters, defining the points of the
   initial grid and
% following ones in case of search of finer discretizations.
alpha=1.5; beta=1.5;

% The initial pointset is a tensorial grid in a small
   rectangle "R"
% containing the domain, with sides parallel to the axis. A
   tensorial rule
% of "k" equispaced points per direction is used. Thus the
   initial set will
% have "k^2=k*k" points.
k=floor(max(ade,5)^(alpha));

% Note: on the setting of restol: 10^(-14) seems fine,
   10^(-15) is often
% not achieved. This quantity is normalized with the number
   of extracted
% nodes.
restol=(1*10^(-14))*sqrt((ade+1)*(ade+2)/2);

% The variable "Z" will contain all the points used in the
   small rectangle
% "R" containing the domain, defined above.
Z=[];

% The variable "Zin" will contain all the points of the set
   defined by "Z",
% that are internal to the domain.
Zin=[];

% The matrix "V" will store the Vandermonde matrix, w.r.t. to
    the tensor
% product Chebyshev polynomials of total degree "ade", scaled
    respect
% to the aforementioned rectangle "R".
V=[];

% The variable "Lmax" defines the maximum number of
   iterations of the
% indomain process to obtain a set "Zin" good enough for
   extraction of
% Tchakaloff points.
Lmax=6;



% ..................... Determine rectangle "R" ..
```

```
% Determine NURBS domain control points.
S=length(structure_RS); P=[];
for kk=1:S
    structure_RSL=structure_RS(kk); PL=structure_RSL.P; P=[P;
   PL];
end

% Establish indomain structures.
[~,on,sp_xnV,sp_xdV,sp_ynV,sp_ydV,boxVx,singular_points_X,...
    singular_points_Y]=inRS([],structure_RS,Nbox,safe_mode);

SH=length(structure_RSH); PH=[];
for kk=1:SH
    structure_RSLH=structure_RSH(kk); PLH=structure_RSLH.P;
   PH=[PH;PLH];
end

% Establish indomain structures.
[~,onH,sp_xnVH,sp_xdVH,sp_ynVH,sp_ydVH,boxVxH,
   singular_points_XH,...
    singular_points_YH]=inRS([],structure_RSH,Nbox,safe_mode)
   ;

% Compute Tens.Chebyshev-Vandermonde moments.
[cmomE,bboxE]=chebmom(ade,sp_xnV,sp_xdV,sp_ynV,sp_ydV);
[cmomH,bboxH]=chebmom(ade,sp_xnVH,sp_xdVH,sp_ynVH,sp_ydVH,
   bboxE);
cmom=cmomE-cmomH;



% ......................... Method iterations .....


for itlevel=1:Lmax

    % ........................... indomain ..........

    % Set test point "ZL" from which we try to extract
   Tchakaloff nodes.
    x=linspace(bboxE(1),bboxE(2),k); y=linspace(bboxE(3),
   bboxE(4),k);
    [u,v]=meshgrid(x,y); ZL=[u(:) v(:)];

    % Find points of ZL in the domain and store them in "X".
    [insideE,onE]=inRS(ZL,structure_RS,Nbox,safe_mode,...
        sp_xnV,sp_xdV,sp_ynV,sp_ydV,boxVx,singular_points_X
   ,...
        singular_points_Y);
```

```
 [insideH,onH]=inRS(ZL,structure_RSH,Nbox,safe_mode,...
     sp_xnVH,sp_xdVH,sp_ynVH,sp_ydVH,boxVxH,
singular_points_XH,...
     singular_points_YH);

 % Points in which "on" is "NaN" are not too close to the
boundary,
 % i.e. with distance larger that a tolerance 10^(-12), or
 with abscissa
 % too close to vertical points.
 ind=find((insideE == 1) & (isnan(onE)) & ...
     not((insideH == 1) & (isnan(onH))) );
 X=ZL(ind,:);

 if length(X) == 0
     % Mesh refinement in case of failures (no points in
domain)
     % fprintf('\n \t * no point in domain');
     k=floor(beta*(k+1));

 else

     % The variable "Zin" contains all the points that
along the
     % iteration are determined as internal to the domain.
     Zin=[Zin; X];


     % The matrix "VL" will store the Vandermonde matrix,
w.r.t. to the
     % tensor product Chebyshev polynomials of total
degree "ade",
     % scaled in the respect to the aforementioned
rectangle "R",
     % relatively to the pointset "X".

     VL=cvand(ade,X,bboxE);
     V=[V; VL];      % Vandermonde of all points inside
the domain
     Z=[Z; ZL];      % All points: inside/outside domain
     [Q,R]=qr(V,0);  % Orthogonalize Vandermonde matrix (
more stable).

     cmom1=R'\cmom;  % Moments w.r.t. the orthog. basis.

     % ......................... compression ..........

     % Perform extraction of the compressed formula from
```

```
nodes in "Zin".
     switch extraction_type
         case 1
             w=lsqnonneg(Q',cmom1);
         case 2
             w = lawsonhanson(Q', cmom1);
         case  3
             w = LHDM(Q', cmom1);
         otherwise
             w=Q'\cmom1;
     end

     % The nodes of the Tchakaloff formula are those
points of "Zin"
     % with strictly positive weights in case a Lawson-
Hanson algorithm
     % has been used.
     % In case the "backslash" method is adopted, the
nodes are those
     % whose weights are nonzero.

     if (extraction_type == 1) | (extraction_type == 1) |
...
             (extraction_type == 3)
         ind1=find(w>0); w=w(ind1);
     else
         ind1=find(abs(w)>0); w=w(ind1);
     end

     % ......................... error analysis ..........

     % Evaluate residual in norm 2, that says hw much the
compressed
     % rule matches the moments.

     res=norm(V(ind1,:)'*w-cmom);

     % In case of failure determine the number of
equispaced points per
     % direction that determine the rule, otherwise exit
from the
     % routine.

     if res>restol | length(w) == 0
         k=floor(beta*k);
         % In case of more iterations, relax the tolerance
         %  "restol" for the norm 2 of the moment matching
.
         restol=10*restol;
```

```
        else
            xyw=[Zin(ind1,:) w];
            return;
        end

    end

end


% In case of failure, provide warnings and try to give a rule
    with internal
% nodes but possibly negative weights. In most of the cases
   the formula has
% "low" cubature condition number (i.e. is almost optimally
   stable).

fprintf(2,'\n \t Compression not completed. Moment error:
   %1.3e',res);

w=Q'\cmom1; ind1=find(abs(w)>0); w=w(ind1);
res=norm(V(ind1,:)'*w-cmom);
if res>restol | length(w) == 0
    fprintf(2,'\n \t Tried backslash (PO rule): fail. Mom.
   error: %1.3e',...
        res);
else

    ineg=length(find(w < 0));
    fprintf(2,'\n \t Tried backslash (PO rule): OK. Mom.
   error: %1.3e ',...
        res);
    fprintf(2,'\n \t NEGATIVE WEIGHTS: %8.0f ',...
        ineg);
end

xyw=[Zin(ind1,:) w];
```

```
function demo_nurbs_cubature_01(example)

%----
% OBJECT:
%----
% Demo illustrating CUBATURE:
% 1. how to define a NURBS on a composite boundary, using
   arcs of disks,
```

```
%    ellipses, segments, polygons and "free NURBS". The
   routines work on
%    piecewise NURBS of different order.
%    To this purpose see the MATLAB function "define_domain"
   at the
%    bottom of this demo.
% 2. how to get an algebraic cubature rule on the desired
   domain, via
%    "cubRS" routine.
% 3. testing the routines over random polynomials.
%---
%% Copyright (C) 2007-2024 Chiara Zara, Alvise Sommariva,
   Marco Vianello.
%%
%% This program is free software; you can redistribute it and
   /or modify
%% it under the terms of the GNU General Public License as
   published by
%% the Free Software Foundation; either version 2 of the
   License, or
%% (at your option) any later version.
%%
%% This program is distributed in the hope that it will be
   useful,
%% but WITHOUT ANY WARRANTY; without even the implied
   warranty of
%% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
   the
%% GNU General Public License for more details.
%%
%% You should have received a copy of the GNU General Public
   License
%% along with this program; if not, write to the Free
   Software
%% Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
   02111-1307 USA
%%
%% Author:
%%          Chiara Zara
%%          Alvise Sommariva <alvise@euler.math.unipd.it>
%%          Marco Vianello   <marcov@euler.math.unipd.it>
%%
%% Date: July 8, 2024 -
%---
% Dates
%---
% First version: October 31, 2021;
% Checked: July 8, 2024.
%---
```

```
warning off;

% ------------------------------ Settings ---

% Algebraic degree of precision of the cubature rule.
   Suggested values are
% mild, e.g. not larger than "ade=15". Typical "ade" in the
   applications
% are between "3" and "5".
adeV=2:2:10;

% Number of tests to determine the median cputime. We suggest
    to set
% "tests=10", while we do not recommend "tests=1" because it
   often turns
% out to provide unreliable cputimes.
tests=100;

% Cubature extraction method: the variable "extraction_type"
   sets how
% Tchakaloff rule is extracted.
%      1: lsqnonneg, 2: lawsonhanson, 3: LHDM 4: \
% If not declared or equal to "[]", the default is LHDM, see
% "dCATCH: a numerical package for d-variate near G-optimal
   Tchakaloff
% regression via fast NNLS"
% M. Dessole, F. Marcuzzi and M. Vianello
% MDPI-Mathematics 8 (2020) - Special Issue "Numerical
   Methods"
extraction_type=3;

% Nbox is a technical parameter for indomain routine; in
   doubt set 100.

Nbox=100;

% In indomain routine one must be certain that points are
   inside the
% domain. In our cubature needs we just wants some points in
   the domain.
% Consequently, if we are not fully sure that a point is in
   the domain, due
% to geometrical issues, we do not take it into account.

safe_mode=1;

% The variable "domain_type" set the domain taken into
   account.
```

```
if nargin < 1
  example=3;
end









% ------------------------------ Main code ---

%---
% 1. Make NURBS structure
%----

[geometry_NURBS,geometry_NURBSH,domain_str]=define_domain(
    example);

REV=[]; logerrV=[];
for jj=1:length(adeV)

    ade=adeV(jj);

    %---
    % 2. Compute algebraic cubature rule of the domain (
    several tests!).
    %---
    xyw = cubRS(ade,geometry_NURBS,geometry_NURBSH,
    extraction_type,Nbox,...
        safe_mode);

    %---
    % 3. Compute algebraic cubature rule of the boundary.
    %---

    [inside,in_doubts,sp_xnV,sp_xdV,sp_ynV,sp_ydV,boxVx,...
        singular_points_X,singular_points_Y]=inRS([],
    geometry_NURBS,...
        Nbox,safe_mode);
    xywL = lineint(ade+1,sp_xnV,sp_xdV,sp_ynV,sp_ydV);

    [insideH,in_doubtsH,sp_xnVH,sp_xdVH,sp_ynVH,sp_ydVH,
    boxVxH,...
        singular_points_XH,singular_points_YH]=inRS([],
    geometry_NURBSH,...
        Nbox,safe_mode);
```

```
    xywLH = lineint(ade+1,sp_xnVH,sp_xdVH,sp_ynVH,sp_ydVH);

    %---
    % 4. Making tests.
    %---

    for k=1:tests
        a=rand(1); b=rand(1); c=rand(1);
        f=@(x,y) (a+b*x+c*y).^ade;

        xx=xyw(:,1); yy=xyw(:,2); ww=xyw(:,3);
        I0=ww'*feval(f,xx,yy);

        xxL=xywL(:,1); yyL=xywL(:,2); wwL=xywL(:,3);
        g=@(x,y) (a+b*x+c*y).^(ade+1)/(b*(ade+1));
        I1E=wwL'*feval(g,xxL,yyL);

        xxLH=xywLH(:,1); yyLH=xywLH(:,2); wwLH=xywLH(:,3);
        I1H=wwLH'*feval(g,xxLH,yyLH);

        I1=I1E-I1H;

        RE(k,1)=abs((I0-I1)./I1);
    end

    %----
    % 5. Statistics.
    %----
    fprintf('\n \n \t ADE    : %2.0f',ade);
    fprintf('\n \t RE max: %1.3e',max(RE));
    kpos=find(RE > 0);
    logerr=10^(sum(log10(RE(kpos)))/length(kpos));
    fprintf('\n \t RE log: %2.3e',logerr);

    cell_tick{jj}=num2str(ade);

    REV=[REV RE];
    logerrV=[logerrV; logerr];
end

%---
% 6. Plots.
%---

h=figure(1);
f1=ishandle(h)&&strcmp(get(h,'type'),'figure'); if f1,clf(1);
   end
figure(1)
axis equal;
```

```
for jj=1:length(adeV)
    n=adeV(jj);
    reL=REV(:,jj); log_reL=logerrV(jj);
    plot_errors(jj,n,reL,log_reL);
    hold on;
end
ax=gca;
ax.XAxis.FontSize = 16;
ax.YAxis.FontSize = 16;
xlim([adeV(1)-1,adeV(end)+1]);
xticks(adeV)
hold off
fprintf('\n \n');

h=figure(2);
f2=ishandle(h)&&strcmp(get(h,'type'),'figure'); if f2,clf(2);
   end
figure(2)

axis equal;
plotNURBSPL(geometry_NURBS);
hold on;
plotNURBSPL(geometry_NURBSH);
plot(xx,yy,'mo','MarkerEdgeColor','k',...
                        'MarkerFaceColor','m',...
                        'MarkerSize',4)
hold off;
```

```
function demo_nurbs_cubature_02(example)

%---
% OBJECT:
%----
% Demo illustrating CUBATURE:
% 1. how to define a NURBS on a composite boundary, using
   arcs of disks,
%    ellipses, segments, polygons and "free NURBS". The
   routines work on
%    piecewise NURBS of different order.
%    To this purpose see the MATLAB function "define_domain"
   at the
%    bottom of this demo.
% 2. how to get an algebraic cubature rule on the desired
   domain, via
%    "cubRS" routine.
% 3. testing the routines over random polynomials.
```

```
%---
%% Copyright (C) 2007-2024 Chiara Zara, Alvise Sommariva,
   Marco Vianello.
%%
%% This program is free software; you can redistribute it and
   /or modify
%% it under the terms of the GNU General Public License as
   published by
%% the Free Software Foundation; either version 2 of the
   License, or
%% (at your option) any later version.
%%
%% This program is distributed in the hope that it will be
   useful,
%% but WITHOUT ANY WARRANTY; without even the implied
   warranty of
%% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
   the
%% GNU General Public License for more details.
%%
%% You should have received a copy of the GNU General Public
   License
%% along with this program; if not, write to the Free
   Software
%% Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
   02111-1307 USA
%%
%% Author:
%%          Chiara Zara
%%          Alvise Sommariva <alvise@euler.math.unipd.it>
%%          Marco Vianello   <marcov@euler.math.unipd.it>
%%
%% Date: July 8, 2024 -
%---
% Dates
%---
% First version: October 31, 2021;
% Checked: July 8, 2024.
%---


warning off;

% ------------------------------ Settings ---

% Algebraic degree of precision of the cubature rule.
   Suggested values are
% mild, e.g. not larger than "ade=15". Typical "ade" in the
   applications
```

```
% are between "3" and "5".
adeV=2:2:10;

% Number of tests to determine the median cputime. We suggest
    to set
% "tests=10", while we do not recommend "tests=1" because it
   often turns
% out to provide unreliable cputimes.
tests=1;

% Cubature extraction method: the variable "extraction_type"
   sets how
% Tchakaloff rule is extracted.
%      1: lsqnonneg, 2: lawsonhanson, 3: LHDM 4: \
% If not declared or equal to "[]", the default is LHDM, see
% "dCATCH: a numerical package for d-variate near G-optimal
   Tchakaloff
% regression via fast NNLS"
% M. Dessole, F. Marcuzzi and M. Vianello
% MDPI-Mathematics 8 (2020) - Special Issue "Numerical
   Methods"
extraction_type=3;

% Nbox is a technical parameter for indomain routine; in
   doubt set 100.

Nbox=3;

% In indomain routine one must be certain that points are
   inside the
% domain. In our cubature needs we just wants some points in
   the domain.
% Consequently, if we are not fully sure that a point is in
   the domain, due
% to geometrical issues, we do not take it into account.

safe_mode=1;

% The variable "domain_type" set the domains taken into
   account.
if nargin < 1
    example=1;
end
```

```
% ----------------------------- Main code ---

%---
% 1. Make NURBS structure
%---
[geometry_NURBS_EXT,geometry_NURBS_INT]=define_domain(example
    );

REV=[]; logerrV=[];

switch example
    case 1
        x0=0; y0=0.4;
    case {2,4}
        x0=-0.2; y0=-1.4;
    case 3
        x0=0.5; y0=-1;
end

% Define functions
f1=@(x,y) exp(-(x.^2+y.^2));
f2=@(x,y) ((x-x0).^2+(y-y0).^2).^(11/2);
f3=@(x,y) ((x-x0).^2+(y-y0).^2).^(1/2);

% Define high order rule
ADEref=30;
xywL = cubRS(ADEref,geometry_NURBS_EXT,geometry_NURBS_INT,0,
    Nbox,safe_mode);

xxL=xywL(:,1); yyL=xywL(:,2); wwL=xywL(:,3);
I(1)=wwL'*feval(f1,xxL,yyL);
I(2)=wwL'*feval(f2,xxL,yyL);
I(3)=wwL'*feval(f3,xxL,yyL);

Zcards=[];

for jj=1:length(adeV)

    ade=adeV(jj);

    %---
    % 2. Compute algebraic cubature rule of the domain (
    several tests!).
    %---
    [xyw,res,Z,Zin] = cubRS(ade,geometry_NURBS_EXT,
    geometry_NURBS_INT,...
```

```
        extraction_type,Nbox,safe_mode);

    Zcards(end+1,:)=[size(xyw,1) size(Zin,1) size(Z,1)];

    xx=xyw(:,1); yy=xyw(:,2); ww=xyw(:,3);

    %---
    % 3. Making tests.
    %---
    I0(1)=ww'*feval(f1,xx,yy);
    I0(2)=ww'*feval(f2,xx,yy);
    I0(3)=ww'*feval(f3,xx,yy);

    RE=abs((I-I0)./I);

    %---
    % 4. Statistics.
    %---

    fprintf('\n \t %2.0f | %1.1e | %1.1e | %1.1e | ',ade,RE
   (1),RE(2),RE(3));


end

fprintf('\n \n \t ..........Cardinalities .............. '
   );

for k=1:size(Zcards,1)
    fprintf('\n \t %2.0d | %4d | %4d  | %4d  | ',adeV(k),
   Zcards(k,1),...
        Zcards(k,2),Zcards(k,3));
end

fprintf('\n \n');









%---
% 6. Plots.
%---

h=figure(1);
f1=ishandle(h)&&strcmp(get(h,'type'),'figure'); if f1,clf(1);
```

```
   end
axis equal;
axis tight;
plotNURBSPL(geometry_NURBS_EXT);
hold on;
plotNURBSPL(geometry_NURBS_INT);
plot(xx,yy,'mo','MarkerEdgeColor','k',...
                        'MarkerFaceColor','m',...
                        'MarkerSize',4)
hold off;
```

Bibliografia

[1] M. Abate, F. Tovena, Curve e Superfici, Springer (2006), pp.4-7.

[2] N. Fusco, P.Marcellini, C.Sbordone, Analisi Matematica due, Liguori Editore (1996), pp.311-314

[3] P.J. Davis, A construction of nonnegative approximate quadratures, Math. Comp. 21 (1967), pp.578–582.

[4] N. Hale, A. Townsend, Fast and Accurate Computation of Gauss-Legendre and Gauss-Jacobi Quadrature Nodes and Weights. SIAM J. Sci. Comput. 35(2), A652–A674 (2013).

[5] L. Piegl, The NURBS Book, 2nd Edition, Springer-Verlag, Berlin-Heidelberg. (1997)

[6] A. Sommariva, M. Vianello, Computing Tchakaloff-like cubature rules on spline curvilinear polygons, Dolomites Res. Notes Approx. DRNA 14 (2021), pp.1–11.

[7] A. Sommariva, M. Vianello, Low cardinality Positive Interior cubature on NURBS-shaped domains, BIT Numer. Math., 63 (22) (2023).

[8] V. Tchakaloff, Formules de cubatures mecaniques à coefficients non negatifs (French), Bull. Sci. Math. 81 (1957), pp.123–134.

[9] Wilhelmsen, D.R., A Nearest Point Algorithm for Convex Polyhedral Cones and Applications to Positive Linear approximation, Math. Comp. 30 (1976), pp.48–57.

[10] C. Zara, Codici Matlab per calcolo di formule di cubatura a bassa cardinalità di tipo Positive Interior su domini molteplicemente connessi con bordo di tipo NURBS, `https://github.com/chiarazaraa/codici_tesi`.