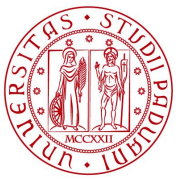


UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"

Corso di Laurea Triennale in Matematica

Implementazione Numerica delle Formule di Cubatura di Glaubitz

Relatore:
Prof. Alvise Sommariva

Laureando: Sara Pangrazio
Matricola: 2003583

Correlatore:
Dott. Giacomo Elefante

Anno Accademico 2024/2025

13 dicembre 2024

Indice

Introduzione

Un classico problema dell'analisi numerica consiste nell'approssimare numericamente l'integrale definito di una funzione continua f relativamente a una funzione peso w, su un dominio multivariato Ω.

A tal proposito vengono determinate delle formule di cubatura che ne approssimano il valore. In particolare, dato un dominio $\Omega \subset \mathbb{R}^d$, una integranda continua $f : \Omega \to \mathbb{R}$ e una funzione peso w, una *formula di cubatura* con nodi $\{x_n\}_{n=1}^N \subseteq \Omega \times \mathbb{R}$ e pesi $\{w_n\}_{n=1}^N$ approssima l'integrale $\int_\Omega f(x)\omega(x)\,dx$ mediante la somma pesata $\sum_{n=1}^N w_n f(x_n)$.

L'intenzione di questo lavoro è approfondire alcuni temi affrontati da *J. Glaubitz* nell' articolo *Stable high-order cubature formulas for experimental data*, il cui scopo è quello di determinare delle formule di cubatura ottimali (stabili ed esatte) per un insieme di punti (*scattered*).

Nelle applicazioni, questi punti sono dati ottenuti da qualche strumento (ad esempio sensori o rilevazioni da satelliti) oppure da campionamenti di natura (quasi-)random.

Riportiamo sinteticamente le tecniche che sono state sviluppate da J. Glaubitz e successivamente procediamo a una rielaborazione dei concetti esposti allo scopo di applicare tale idee su domini di integrazione non considerati dall'autore, ottenendo formule di quadratura esatte e stabili.

Nel Capitolo 1 richiamiamo le formule proposte da J. Glaubitz.

Nel Capitolo 2 descriviamo come ottenere formule di cubatura di tipo prodotto tensoriale su una famiglia di domini planari curvi noti come *blending of elliptic arcs*.

Nel Capitolo 3 mostriamo esperimenti numerici su questa famiglia di domini, relativamente alla cubatura su dati scattered, utilizzando la tecnica proposta da J. Glaubitz.

Nell'appendice inseriamo infine i codici utilizzati.

Capitolo 1

Le formule di quadratura di Glaubitz

In questo capitolo introduciamo le recenti formule di cubatura descritte da J. Glaubitz in [4].

Dati

- un dominio $\Omega \subset \mathbb{R}^d$,
- le coppie di punti $\{(x_n, f_n)\}_{n=1}^N \subseteq \Omega \times \mathbb{R}$,
- la funzione continua $f : \Omega \to \mathbb{R}$,

una *formula di cubatura* $S_N(f)$ intende approssimare l'integrale $I(f)$

$$I(f) := \int_\Omega f(x)\omega(x)\,dx \approx S_N(f) = \sum_{n=1}^N w_n f(x_n) \tag{1.1}$$

I punti $\{x_n\}_{n=1}^N$ sono detti *nodi* mentre i valori $\{w_n\}_{n=1}^N$ sono noti come *pesi*.

A seconda degli autori si parla di *quadratura* in ambito univariato e di *cubatura* in ambito multivariato, oppure col termine *quadratura* si include quello di *cubatura*. Nel nostro caso utilizzeremo questo secondo significato.

Lo scopo di [4] è definire formule di cubatura $S_N(f)$ partendo dalla conoscenza del valore dell'integranda su dato un insieme di punti *scattered* che

- abbiano un elevato grado algebrico di esattezza,
- siano *stabili*, ovvero forniscano un risultato accurato anche in assenza delle valutazioni esatte $\{f(x_n)\}_{n=1}^N$, disponendo solo di loro approssimazioni $\{f_n^\epsilon\}_{n=1}^N$.

Il fatto che i nodi abbiano questa natura non prefissata a piacere dall'utente, é molto comune nelle applicazioni. Tipicamente possono essere sensori piazzati dove possibile da tecnici, dati derivanti da satelliti o punti di tipo *random* o `quasi-random`.

Quale notazione, utilizzeremo in seguito $\mathbb{P}_\delta(\Omega)$ o piú semplicemente $\mathbb{P}_\delta$, per indicare lo spazio dei polinomi di grado totale al piú δ sul dominio di integrazione Ω. Ricordiamo che un polinomio p in $\Omega \subseteq \mathbb{R}^d$ ha *grado totale* δ se e solo se

$$p(x) = \sum_{(i_1,\dots i_d)\in\mathbb{N}^d} a_{i_1,\dots,i_d} x_1^{i_1} \cdot \ldots \cdot x_d^{i_d}$$

dove gli esponenti $i_1, \dots, i_d$ sono tali che $\sum_{k=1}^d i_k \leq \delta$.

1.1 Esattezza della formula di quadratura

Inoltre, come nel caso univariato, una formula di quadratura S_N ha grado di esattezza polinomiale *almeno* δ se

$$S_N(p_\delta) = I(p_\delta) \quad \text{per ogni } p_\delta \text{ in } \mathbb{P}_\delta(\Omega)$$

ed ha grado di esattezza precisamente δ se ha grado almeno δ ed esiste un polinomio $p_{\delta+1} \in \mathbb{P}_{\delta+1}$ per cui $S_N(p_{\delta+1}) \neq I(p_{\delta+1})$.

Definizione 1.1 (Insieme Unisolvente)**.** *Un insieme di punti $X = \{x_n\}_{n=1}^N \subset \mathbb{R}^d$ si dice $\mathbb{P}_\delta(\Omega)$ − unisolvente se per ogni $p \in \mathbb{P}_\delta(\Omega)$ si ha che*

$$p(x_n) = 0, \quad \text{per } n = 1, \dots, N \implies p \equiv 0 \tag{1.2}$$

La condizione di esattezza può essere riscritta mediante il sistema lineare (detto *dei momenti*). Data una base $\{p_k\}_{k=1}^K$ di $\mathbb{P}_\delta(\Omega)$, sia $\eta := \dim(\mathbb{P}_\delta(\Omega))$ e i momenti $m_k = I(p_k)$, $k = 1, \dots, \eta$.

Se la formula ha grado di esattezza δ allora

$$\underbrace{\begin{pmatrix} p_1(x_1) & \cdots & p_1(x_N) \\ \vdots & \ddots & \vdots \\ p_\eta(x_1) & \cdots & p_\eta(x_N) \end{pmatrix}}_{=:\mathbf{P}_\delta} \underbrace{\begin{pmatrix} w_1 \\ \vdots \\ w_N \end{pmatrix}}_{=:\mathbf{w}} = \underbrace{\begin{pmatrix} m_1 \\ \vdots \\ m_\eta \end{pmatrix}}_{=:\mathbf{m}} \tag{1.3}$$

Se in particolare $\eta < N$, il sistema dei momenti diventa indeterminato. Per stabilire se non ha soluzioni o se ne ha infinite, nel caso $X = \{x_n\}_{n=1}^N$ sia unisolvente per $\mathbb{P}_\delta$, introduciamo il seguente lemma (cf. Lemma 5 in [4]).

Lemma 1.1. *Sia $\eta := dim(\mathbb{P}_\delta(\Omega)) < N$ e sia $X = \{x_n\}_{n=1}^N$ un insieme $\mathbb{P}_\delta(\Omega)$ − unisolvente. Allora il sistema lineare (1.3) è indeterminato e genera uno spazio affine di soluzioni*

$$W = \left\{ w \in \mathbb{R}^N \mid \mathbf{P}_\delta \mathbf{w} = \mathbf{m} \right\} \tag{1.4}$$

di dimensione $N - \eta$.

Dunque, se l'insieme di punti *scattered* $\{x_n\}_{n=1}^N$ è $\mathbb{P}_\delta(\Omega)$ − unisolvente, sappiamo che il sistema (1.4) ha infinite soluzioni e ciascuna di esse rappresenta i pesi di una formula di cubatura di grado δ.

1.2 Stabilità della formula di quadratura

Per proseguire con la nostra analisi preliminare, utile per introdurre una certa famiglia di formule di cubatura, descriviamo il concetto di stabilità.

Si considerino

- un dominio $\Omega \subset \mathbb{R}$,
- una funzione continua $f : \Omega \to \mathbb{R}$,
- i punti $\{x_i\}_{i=1,\dots,N} \subset \Omega$,
- siano noti dei valori f_k^ϵ che approssimano $f(x_k)$, $k = 1, \dots, N$, e supponiamo che valga $\max_{k=1,\dots,N} |f(x_k) - f_k^\epsilon| \leq \epsilon$.

Dalla disuguaglianza triangolare

$$\begin{aligned} |I(f) - S_N(f^\epsilon)| &= |I(f) - S_N(f) + S_N(f) - S_N(f^\epsilon)| \\ &\leq |I(f) - S_N(f)| + |S_N(f) - S_N(f^\epsilon)|. \end{aligned}$$

Inoltre

$$\begin{aligned} |S_N(f) - S_N(f^\epsilon)| &= |\sum_{i=0}^{n} w_n (f_n - f_n^\epsilon)| \\ &\leq \sum_{i=0}^{n} |w_n||(f_n - f_n^\epsilon)| \\ &\leq \max_{j=1,\dots,N} |f_j - f_j^\epsilon| \left(\sum_{i=0}^{n} |w_n| \right) \end{aligned} \tag{1.5}$$

Si conclude quindi

$$|S_N(f) - S_N(f^\epsilon)| \leq \epsilon \, k(\mathbf{w}), \quad k(\mathbf{w}) = \sum_{i=0}^{n} |w_n|. \tag{1.6}$$

Il termine $k(\mathbf{w})$ viene chiamato *fattore di amplificazione dell'errore* ed è connesso alla stabilità della formula. Infatti, piú alto risulta $k(\mathbf{w})$ piú $S_N(f^\epsilon)$ puó discostarsi da $S_N(f)$.

Se la formula di cubatura considerata ha grado di esattezza maggiore o uguale a zero, essa è esatta per le costanti, detta $\mu(\Omega)$ la misura del dominio, abbiamo

$$0 \leq \mu(\Omega) = I(1) = \sum_{n=1}^{N} w_n \leq \sum_{n=1}^{N} |w_n| = k(\mathbf{w}) \tag{1.7}$$

con l'uguaglianza che vale nella precedente stima se e solo se i pesi della formula di cubatura sono non negativi. Quindi, una formula di cubatura con pesi positivi ha un fattore di amplificazione dell'errore $k(\mathbf{w})$ minimo.

1.3 Metodi Proposti

Tenendo a mente le osservazioni appena fatte, Glaubitz introduce le due formule di cubatura assicurandone prima la stabilità, e successivamente massimizzando il grado di esattezza.

1.2.1 Formule di cubatura ℓ^1

L'obiettivo principale di questo primo approccio è assicurare la stabilità e dunque minimizzare $k(\mathbf{w}) = \sum_{i=0}^{n} |w_n|$, quindi si determina un elemento $\mathbf{w}^* \in W$ tale che

$$k(\mathbf{w}^*) \leq k(\mathbf{w}), \quad \text{per ogni } \mathbf{w} \in W \tag{1.8}$$

Dato che $k(\mathbf{w}) = \|\mathbf{w}\|_1$, questo può essere definito come un problema di ℓ^1-minimizzazione, la cui soluzione è

$$\mathbf{w}^* = \mathbf{w}^{\ell^1} = \arg\min_{\mathbf{w}\in W} \|\mathbf{w}\|_1 \tag{1.9}$$

Se i pesi sono positivi allora la formula di quadratura è stabile per costruzione, ha grado di esattezza δ e viene definita in questo modo

$$S_N^{\ell^1}(f^\epsilon) := \sum_{n=1}^{N} w_n^{\ell^1} f_n^\epsilon \tag{1.10}$$

Se è presente almeno un peso negativo, il metodo diminuisce il grado di esattezza passando da δ a $\delta - 1$, riformulando il sistema dei momenti relativamente al nuovo grado di esattezza, ricavando lo spazio di soluzioni W, per quindi determinare $\mathbf{w}^{\ell^1} = \arg\min_{\mathbf{w}\in W} \|\mathbf{w}\|_1$. Si può ripetere il procedimento finchè non si ottiene una formula di cubatura non negativa.

1.2.2 Formula di cubatura ai minimi quadrati

Il secondo approccio che considera Glaubitz è minimizzare la norma pesata ℓ^2 invece della norma ℓ^1.

Quindi, il vettore dei pesi sarà dato da

$$\mathbf{w}^{LS} = \arg\min_{w\in W} \|\mathbf{R}^{-\frac{1}{2}}\mathbf{w}\| \tag{1.11}$$

dove la matrice $R^{-1/2}$ è definita come

$$\mathbf{R}^{-\frac{1}{2}} = \operatorname{diag}\left(\frac{1}{\sqrt{r_1}}, \ldots, \frac{1}{\sqrt{r_n}}\right), \quad r_n = \frac{\omega(x_n)\cdot\mu(\Omega)}{N} > 0 \tag{1.12}$$

Per quanto riguarda la stabilità, possiamo dire che il primo metodo offre una formula più stabile di quest'ultimo, essendo $k(\mathbf{w}^{\ell^1}) \leq k(\mathbf{w}^{LS})$, che peró ha il vantaggio di disporre di una forma esplicita di $\mathbf{w}^{LS}$ in quanto

$$\mathbf{w}^{LS} = \mathbf{R}\mathbf{P}^T(\mathbf{P}\mathbf{R}\mathbf{P}^T)^{-1}\mathbf{m}. \tag{1.13}$$

Dove $\mathbf{P} \equiv \mathbf{P}_\delta$ dell'equazione (1.3). Quindi, una volta calcolato il vettore dei pesi, si ripete il procedimento adoperato con il primo metodo, diminuendo il grado di esattezza δ finchè non otteniamo una formula di cubatura non negativa. A quel punto si può concludere che

$$S_N^{LS}(f^\epsilon) := \sum_{n=1}^{N} w_n^{LS} f_n^\epsilon \tag{1.14}$$

1.4 Risultati Teorici

Il risultato principale dello studio di Glaubitz è il teorema che segue, che afferma che fissato un grado di esattezza δ, i pesi della *formula di cubatura LS* sono non negativi solo se un numero sufficiente dei punti $\{x_n\}_{n=1}^N$ sono $\mathbb{P}_\delta(\Omega)$- unisolventi.

Teorema 1.1. *Sia $N_0 \in \mathbb{N}_0 = \mathbb{N} \setminus \{0\}$ tale che $X_{N_0} = \{x_n\}_{n=1,\dots,N_0} \subset \Omega$ sia $\mathbb{P}_\delta(\Omega)$-unisolvente e sia $\omega(x_j) > 0$ per $n = 1, \dots, N_0$ con ω funzione peso.*

Inoltre, per $N > N_0$ sia $X_N = X_{N_0} \cup \{x_n\}_{n=N_0+1}^N$ e $r_n = \frac{\omega(x_n)\mu(\Omega)}{N}$ i coefficienti che definiscono la matrice $\mathbf{R}^{-1/2}$ del problema ai minimi quadrati (1.11).

Assumiamo che

$$\lim_{N\to\infty} \frac{\mu(\Omega)}{N} \sum_{n=1}^{N} u(x_n)v(x_n)\omega(x_n) = \int_\Omega u(x)v(x)\omega(x)\,dx, \text{ per ogni } u, v \in \mathbb{P}_\delta(\Omega). \quad (1.15)$$

Allora esiste $N_1 \geq N_0$ tale che per ogni $N \geq N_1$ anche i pesi della formula di cubatura LS sono tutti non negativi.

Il teorema ha il seguente significato. Se la sequenza di punti $\{x_n\}_{n\in\mathbb{N}}$ ha una certa proprietá relativa all'integrazione, e si mostra per un certo N_0 l'insieme X_{N_0} é $\mathbb{P}_\delta(\Omega)$-unisolvente, allora esiste un suo sovrainsieme X_{N_1} dato dai primi N_1 punti della sequenza, determinante una formula a pesi positivi.

1.4.1 Polinomi Ortogonali e Caratterizzazione del problema ai Minimi Quadrati

Glaubitz osserva che nonostante la *formula di cubatura LS* offra una scrittura esplicita dei pesi in quanto $\mathbf{w}^{LS} = \mathbf{R}\mathbf{P}^T(\mathbf{P}\mathbf{R}\mathbf{P}^T)^{-1}\mathbf{m}$, da un punto di vista pratico non è conveniente cercare di calcolarli in quanto la matrice $\mathbf{P}\mathbf{R}\mathbf{P}^T$ può risultare molto complicata.

Tuttavia, introducendo i polinomi ortogonali discreti, la loro scrittura si riduce a

$$\mathbf{w}^{LS} = \mathbf{R}\mathbf{P}^T\mathbf{m} \quad (1.16)$$

In quest'ottica si definisce un classico prodotto scalare continuo definito dalla funzione peso ω

$$\langle u, v\rangle = \int_\Omega u(x)v(x)\omega(x)dx \quad (1.17)$$

la cui norma corrispondente è $\|\cdot\| = \sqrt{\langle\cdot,\cdot\rangle}$.

Relativamente a tale prodotto scalare, é possibile determinare una base ortogonale $\{\pi_k\}_{k=1}^{\eta_\delta}$ in $\mathbb{P}_\delta(\Omega)$ rispetto a w.

Per certi domini come il disco unitario, il quadrato, il simplesso, la sfera, nel caso $w \equiv 1$ tali basi sono note esplicitamente.

Si può ripetere lo stesso procedimento per il caso discreto. Noto un vettore di pesi positivi $\mathbf{r} = (r_1, \ldots, r_N)$ é possibile introdurre il prodotto scalare discreto

$$\langle u, v \rangle_N = \sum_{n=1}^{N} r_n u(x_n) v(x_n) \tag{1.18}$$

e la norma corrispondente $\|\cdot\|_N = \sqrt{\langle \cdot, \cdot \rangle_N}$. Supponiamo inoltre che il set $X = \{x_n\}_{n=1,\ldots,N}$ sia $\mathbb{P}_\delta(\Omega)$ unisolvente.

A partire da questo prodotto scalare discreto, si puó determinare una famiglia di polinomi ortonormali $\{\phi_k\}_{k=1,\ldots,\eta_\delta}$ (questa volta discreti), ovvero tali che

$$\langle \phi_k, \phi_l \rangle_N = \delta_{k,l} \tag{1.19}$$

A questo punto si nota che la matrice (di Gram) $\mathbf{P}\mathbf{R}\mathbf{P}^T$

$$\begin{pmatrix} \langle \phi_1, \phi_1 \rangle_N & \ldots & \langle \phi_1, \phi_k \rangle_N \\ \vdots & & \vdots \\ \langle \phi_{\eta_\delta}, \phi_1 \rangle_N & \ldots & \langle \phi_{\eta_\delta}, \phi_{\eta_\delta} \rangle_N \end{pmatrix}$$

corrisponde alla matrice identica e quindi

$$\mathbf{w}^{LS} = \mathbf{R}\mathbf{P}^T\mathbf{m}. \tag{1.20}$$

Dalla struttura diagonale della matrice $\mathbf{R}$, ricordando la definizione di $\mathbf{P}$ e $\mathbf{m}$, ricaviamo che i pesi della *formula di cubatura LS* possono essere scritti esplicitamente come

$$\begin{aligned} w_n^{LS} &= (\mathbf{R}\mathbf{P}^T\mathbf{m})_n = r_n \sum_{k=1}^{\eta_\delta} \mathbf{P}_{n,k}^T m_k \\ &= r_n \sum_{k=1}^{\eta_\delta} \phi_k(x_n)\, I(\phi_k), \quad n = 1, \ldots, N. \end{aligned} \tag{1.21}$$

1.5 Algoritmi di costruzione delle formule di cubatura di Glaubitz

In questa sezione, riportiamo il modo in cui Glaubitz espone l'aspetto computazionale delle formule proposte.

Nella sua implementazione, Glaubitz calcola la matrice $\mathbf{P}$ a partire da una base di monomi. Invece, per quanto riguarda i momenti, in molti domini possono essere noti, altrimenti possono essere determinati usando una formula di cubatura.

Glaubitz riassume il processo di costruzione delle due formule nei seguenti Algoritmi 1 e 2. Essi si basano su due passi fondamentali:

- Partire con il grado di esattezza δ più alto possibile per cui i punti dati sono $\mathbb{P}_\delta(\mathbb{R}^d)-$ unisolventi,

- Diminuire δ fino a trovare formule di cubatura non negative.

Algorithm 1 Costruzione della formula di cubatura ℓ^1

1: Determinare il massimo $\delta \in \mathbb{N}$ tale che X sia $\mathbb{P}_\delta(\mathbb{R}^d) -$ unisolvente
2: Formulare il sistema lineare $P\mathbf{w} = m$ per il grado di esattezza δ
3: Calcolare la soluzione ℓ^1, $\mathbf{w}^* = \arg\min_{w \in W} \|w\|_1$
4: **while** $\mathbf{w}^{\ell^1} \not\geq 0$ **do**
5: Diminuire il grado di esattezza: $\delta = \delta - 1$
6: Formulare il sistema lineare $P\mathbf{w} = m$ per il grado di esattezza diminuito δ
7: Calcolare la soluzione ℓ^1, $\mathbf{w}^* = \arg\min_{w \in W} \|w\|_1$
8: **end while**
9: Approssimare $I[f]$ con $C_N^{\ell^1}[f^\epsilon]$

Algorithm 2 Costruzione della formula di cubatura LS

1: Determinare il massimo $\delta \in \mathbb{N}$ tale che X sia $\mathbb{P}_\delta(\mathbb{R}^d) -$ unisolvente
2: Formulare il sistema lineare $P\mathbf{w} = m$ per il grado di esattezza δ
3: Calcolare la soluzione LS, $\mathbf{w}^{LS} = \arg\min_{w \in W} \|R^{-1/2} w\|_2$
4: **while** $\mathbf{w}^{LS} \not\geq 0$ **do**
5: Diminuire il grado di esattezza: $\delta = \delta - 1$
6: Formulare il sistema lineare $P\mathbf{w} = m$ per il grado di esattezza diminuito δ
7: Calcolare la soluzione LS, $\mathbf{w}^{LS} = \arg\min_{w \in W} \|R^{-1/2} w\|_1$
8: **end while**
9: Approssimare $I[f]$ con $C_N^{LS}[f^\epsilon]$

Tuttavia, dalle implementazioni numeriche viene appurato che utilizzando questi algoritmi si raggiunge un grado di esattezza finale decisamente inferiore al massimo grado possibile per cui l'insieme di punti X sia $\mathbb{P}_\delta(\mathbb{R}^d) -$ unisolvente. Quindi nella sua implementazione Glaubitz decide di utilizzare l'Algoritmo 3, dimostratosi più efficiente.

Algorithm 3 Costruzione efficiente delle formule di cubatura ℓ^1 e LS

$\delta, r, K, w_{\min} = 0$
while $r = K$ **and** $w_{\min} \geq 0$ **do**
    Aumentare il grado di esattezza: $\delta = \delta + 1$
    $K = \binom{\delta+d}{d}$
    Formulare il sistema lineare $Pw = m$ per grado di esattezza δ
    Calcolare il rango di P: $r = \text{rango}(P)$
    Calcolare la soluzione ℓ^1/LS detta w^*
    Determinare il peso minimo: $w_{\min} = \min(w^*)$
end while
Diminuire il grado di esattezza: $\delta = \delta - 1$
Formulare il sistema lineare $Pw = m$ per grado di esattezza δ
Calcolare la soluzione ℓ^1/LS detta w^*
Approssimare $I[f]$ con la formula di cubatura ℓ^1/LS

È importante notare che la condizione

$$rango(P) = K \quad \text{con} \quad K = \binom{\delta + d}{d} \tag{1.22}$$

Equivale a porre X $\mathbb{P}_\delta(\mathbb{R}^d)$ − unisolvente.

Invece, per quanto riguarda l'implementazione dei singoli metodi, si può notare che la soluzione ℓ^1 è definita come la soluzione un problema di programmazione lineare:

$$\min_{w \in \mathbb{R}^{\mathbb{N}}} 1^T w \quad \text{s.a.} \quad P\mathbf{w} = m, \quad w \geq 0 \tag{1.23}$$

Dunque Glaubitz sceglie di implementarla con la funzione *linprog* di MATLAB.

Invece, per quanto riguarda la soluzione LS, viene utilizzata la funzione di MATLAB é `lsqminnorm`, che usa una scomposizione QR della matrice $\mathbf{PR}^{1/2} = A$.

Capitolo 2

Blending lineare di archi ellittici

In questo capitolo, avendo in mente esperimenti numerici su tali domini introduciamo brevemente i cosiddetti domini che sono *blending lineare di archi ellittici*, come descritto in [7].

Nel dettaglio,

- daremo la definizione di tali domini;
- citeremo un teorema che descrive come ottenere delle formule di quadratura su tali domini;
- mostreremo alcuni speciali domini che hanno questa struttura, come settori circolari e segmenti circolari.

2.1 Domini dati da blending lineare di archi ellittici

Prima di cominciare la descrizione del blending lineare di archi ellittici, ricordiamo che dati due punti $x, y \in \mathbb{R}^n$ e uno scalare $\lambda \in [0, 1]$, la combinazione convessa di x e y è data dal segmento che unisce x con y, ovvero $\lambda x + (1 - \lambda)y$.

I domini dati da blending lineare di archi ellittici sono delle regioni planari generate dalla combinazione convessa della parametrizzazione polare dei due archi corrispondenti aventi uguali intervalli angolari.

Tra alcuni esempi possiamo vedere la lente simmetrica, la mezzaluna, il segmento circolare oppure archi circolari come il settore circolare anulare.

Andiamo a vedere più nel dettaglio cosa sono i domini planari generati da blending lineare. Siano

$$P(\theta) = A_1 \cos(\theta) + B_1 \sin(\theta) + C_1, \quad Q(\theta) = A_2 \cos(\theta) + B_2 \sin(\theta) + C_2 \tag{2.1}$$

con $\theta \in [\alpha, \beta]$, due curve trigonometriche planari di primo grado con i vettori non nulli

$$A_i = (a_{i1}, a_{i2}), \quad B_i = (b_{i1}, b_{i2}), \quad C_i = (c_{i1}, c_{i2}), \quad i = 1, 2 \tag{2.2}$$

Se le due curve, che sono una rappresentazione degli archi circolari o ellittici in questione, sono parametrizzate sullo stesso intervallo angolare $[\alpha, \beta]$, si può dimostrare che tramite

una riparametrizzazione, se A_i e B_i non sono ortogonali, le curve iniziali sono archi di ellissi centrate rispettivamente in C_1 e C_2.

Osserviamo che scegliendo opportunamente A_i e B_i, le curve possono ridursi a segmenti o perfino a punti.

Consideriamo adesso il dominio

$$\Omega = \{(x,y) = U(t,\theta) \coloneqq tP(\theta) + (1-t)Q(\theta)\}, \quad (t,\theta) \in [0,1] \times [\alpha,\beta], \tag{2.3}$$

che corrisponde a una trasformazione del rettangolo $[0,1] \times [\alpha,\beta]$ tramite combinazione convessa (*linear blending*) delle curve $P(\theta)$ e $Q(\theta)$.

La mappa U è analitica in (t,θ), che significa che in un intorno di (t,θ) può essere espressa da una serie di potenze convergente. Assumiamo ora che sia anche iniettiva in $[0,1] \times [\alpha,\beta]$, questo implica che il segno del suo determinante Jacobiano $JU(t,\theta)$ rimane costante in tutto il rettangolo.

Scriviamo ora $P(\theta) = (p_1(\theta), p_2(\theta))$ e $Q(\theta) = (q_1(\theta), q_2(\theta))$, otteniamo che

$$JU(t,\theta) = \begin{pmatrix} p_1(\theta) - q_1(\theta) & tp_1'(\theta) + (1-t)q_1'(\theta) \\ p_2(\theta) - q_2(\theta) & tp_2'(\theta) + (1-t)q_2'(\theta) \end{pmatrix} \tag{2.4}$$

Calcoliamo adesso il determinante,

$$\begin{aligned} \det(JU(t,\theta)) &= (p_1(\theta) - q_1(\theta))\big(tp_2'(\theta) + (1-t)q_2'(\theta)\big) \\ &\quad - \big(tp_1'(\theta) + (1-t)q_1'(\theta)\big)(p_2(\theta) - q_2(\theta)) \\ &= t\big[p_2'(\theta)(p_1(\theta) - q_1(\theta)) - p_1'(\theta)(p_2(\theta) - q_2(\theta))\big] \\ &\quad + (1-t)\big[q_2'(\theta)(p_1(\theta) - q_1(\theta)) - q_1'(\theta)(p_2(\theta) - q_2(\theta))\big]. \end{aligned}$$

Se ora poniamo

$$u = (p_2' - q_2')(p_1 - q_1) - (p_1' - q_1')(p_2 - q_2) \quad \text{e} \quad v = q_2'(p_1 - q_1) - q_1'(p_2 - q_2)$$

otteniamo

$$|\det(JU(t,\theta))| = |tu(\theta) + v(\theta)| = \pm(tu(\theta) + v(\theta)) \tag{2.5}$$

Si può quindi vedere che il determinante Jacobiano rimane a segno costante e $u \in \mathbb{T}_1[\alpha,\beta]$ e $v \in \mathbb{T}_2[\alpha,\beta]$, dove

$$\mathbb{T}_n([\alpha,\beta]) = \operatorname{span}\{\{1, \cos(k\theta), \sin(k\theta)\} \mid \quad 1 \le k \le n, \quad \theta \in [\alpha,\beta]\} \tag{2.6}$$

è lo spazio vettoriale dei polinomi trigonometrici di grado n.

Dunque si può dire che

$$|\det(JU(t,\theta))| \equiv \pm(tu(\theta) + v(\theta)) \in \mathbb{P}_1[0,1] \bigotimes \mathbb{T}_2[\alpha,\beta]$$

.

Andando a sostituire le espressioni esplicite di

$$\begin{aligned} p_1 &= a_{11}\cos(\theta) + b_{11}\sin(\theta) + c_{11}, \\ p_2 &= a_{12}\cos(\theta) + b_{12}\sin(\theta) + c_{12}, \\ q_1 &= a_{21}\cos(\theta) + b_{21}\sin(\theta) + c_{21}, \\ q_2 &= a_{22}\cos(\theta) + b_{22}\sin(\theta) + c_{22} \end{aligned} \tag{2.7}$$

otteniamo che

$$u(\theta) = u_0 + u_1 \cos(\theta) + u_2 \sin(\theta) \tag{2.8}$$

con

$$\begin{aligned} u_0 &= (a_{11} - a_{21})(b_{12} - b_{22}) + (a_{12} - a_{22})(b_{21} - b_{11}), \\ u_1 &= (b_{12} - b_{22})(c_{11} - c_{21}) + (b_{21} - b_{11})(c_{12} - c_{22}), \\ u_2 &= (a_{11} - a_{21})(c_{12} - c_{22}) + (a_{12} - a_{22})(c_{21} - c_{11}). \end{aligned}$$

e

$$v(\theta) = v_0 + v_1 \cos(\theta) + v_2 \sin(\theta) + v_3 \cos(\theta)\sin(\theta) + v_4 \sin^2(\theta) \tag{2.9}$$

con

$$\begin{aligned} v_0 &= b_{21}(a_{22} - a_{12}) + b_{22}(a_{11} - a_{21}), \\ v_1 &= b_{21}(c_{22} - c_{12}) + b_{22}(c_{11} - c_{21}), \\ v_2 &= a_{21}(c_{12} - c_{22}) + a_{22}(c_{21} - c_{11}), \\ v_3 &= a_{12}a_{21} - a_{11}a_{22} + b_{11}b_{22} - b_{12}b_{21}, \\ v_4 &= a_{12}b_{21} - a_{11}b_{22} + a_{21}b_{12} - a_{22}b_{11}. \end{aligned}$$

A questo punto, si può osservare che esiste una proprietà geometrica che, se valida, conferma l'iniettività della mappa U. Infatti, dato che la funzione è iniettiva se non mappa due punti distinti nello stesso punto, abbiamo che la stessa proprietà vale se

- gli archi P e Q si possono intersecare solo nei loro estremi,

- dati $\theta_1, \theta_2 \in [\alpha, \beta]$ i segmenti $[P(\theta_1), Q(\theta_1)], [P(\theta_2), Q(\theta_2)]$ si possono intersecare solo nei loro estremi.

Consideriamo il caso in cui abbiamo un arco di ellisse, quindi dove $P(\theta) \equiv C_1$. Secondo quello che è stato detto sopra, C_1 deve appartenere a una regione ammissibile, limitata tra le due tangenti alla curva $Q(\theta)$ nei punti $Q(\alpha)$, $Q(\beta)$, come si può vedere in Figura 2.1.

Abbiamo quindi due casi, che possiamo vedere in Figura 2.2.

1. Il segmento che unisce C_2 con V, il punto di intersezione delle tangenti in $Q(\alpha)$ e $Q(\beta)$ non interseca la curva $Q(\theta)$. Allora, la regione ammissibile è limitata e coincide con la parte del doppio angolo generato dalle tangenti che contiene l'arco,

2. Il segmento che unisce C_2 e V interseca la curva $Q(\theta)$. Allora, la regione ammissibile è illimitata e coincide con la parte superiore del doppio angolo, limitata dalla curva Q unita alla parte inferiore del doppio angolo.

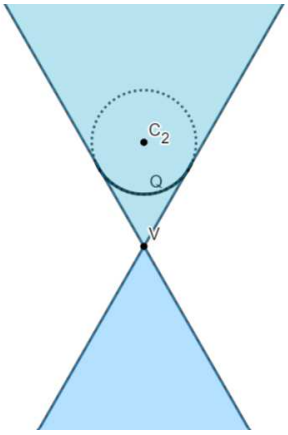


Figura 2.1: Regione di piano ammissibile per C_1

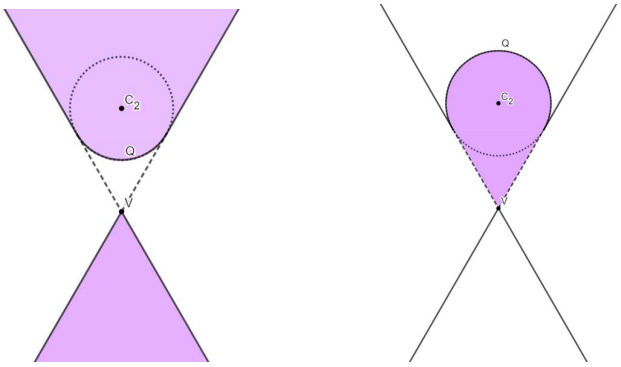


Figura 2.2: Rispettivamente, caso illimitato e illimitato di regioni ammissibili per C_1

2.2 Sulle formule di quadratura su tali domini

Per descrivere al meglio come ottenere formule di quadratura su domini dati da blending lineare di archi ellittici, bisogna prima dare la definizione di *formula di quadratura Gaussiana* e in seguito citare uno dei risultati principali di [2] che è stato uno dei primi passi per trovare delle formule di quadratura Gaussiane per domini in questione.

Le formule di quadratura Gaussiane, come visto in [3] nascono dalla necessità di trovare formule di quadratura valide anche su intervalli illimitati, valide per certe funzioni peso e, a parità di nodi rispetto con formule, con un grado di precisione più alto.

Viene dimostrato che ciò e possibile da Jacobi nel 1826, nel *Teorema di esistenza e unicità delle formule gaussiane*. Infatti, Jacobi prova che data una funzione f e una funzione peso $w(x)$, per qualsiasi $n \geq 1$ esistono e sono unici i nodi $x_1, \ldots, x_n$ e i pesi $w_1, \ldots, w_n$ che garantiscono una formula di quadratura con grado di esattezza $\delta = 2n-1$.

I nodi $x_1, \ldots, x_n$ sono gli zeri del polinomio ortogonale di grado n,

$$\phi_n(x) = A_n \cdot (x - x_1)\cdot, \ldots, \cdot(x - x_n)$$

e i pesi sono dati da

$$w_i = \int_a^b L_i(x)w(x)dx = \int_a^b L_i^2(x)w(x)dx, \quad i = 1, \ldots, n$$

Dove $L_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x-x_j}{x_i-x_j}$ è l'i-esimo polinomio di Lagrange.

A questo punto introduciamo la seguente proposizione.

Proposizione. 2.2.1. *Siano $\{\xi_j, \lambda_j\}_{1\leq j\leq \delta+1}$ i nodi e i pesi positivi della formula di quadratura Gaussiana per la funzione peso*

$$w(x) = \frac{2\sin(\omega/2)}{\sqrt{1-x^2\sin^2(\omega/2)}}, \quad x \in (-1,1), \quad \omega \in (0,\pi]. \tag{2.10}$$

Allora per $0 < \beta-\alpha \leq 2\pi$ vale la seguente formula di quadratura Gaussiana trigonometrica

$$\int_\alpha^\beta f(\theta)d\theta = \sum_{j=1}^{\delta+1} \lambda_j f(\theta_j + \mu), \quad \text{per ogni } f \in \mathbb{T}_\delta([\alpha,\beta]), \quad \mu = \frac{\alpha+\beta}{2}, \tag{2.11}$$

dove

$$\theta_j = 2\arcsin\left(\xi_j \sin(\omega/2)\right) \in (-\omega,\omega), \quad j = 1,2,\ldots,\delta+1, \quad \omega = \frac{\beta-\alpha}{2}. \tag{2.12}$$

Servirà in particolare la formula (2.11), che permetterà di trovare formule di cubatura algebrica per i domini dati da blending lineare nel seguente teorema.

Teorema 2.1. *Si consideri il dominio planare generato dal blending lineare di due archi parametrici definiti come in (2.1)*

$$\Omega = \{(x,y) = U(t,\theta), \quad (t,\theta) \in [0,1]\times[\alpha,\beta], \quad 0 < \beta-\alpha \leq 2\pi\}, \tag{2.13}$$

dove la trasformazione $U(t,\theta)$ è iniettiva per ogni $(t,\theta) \in (0,1)\times(\alpha,\beta)$. Allora vale la seguente formula Gaussiana con $\delta^2/2 + \mathcal{O}(\delta)$ nodi

$$\int\int_\Omega f(x,y)dxdy = I_n(f) = \sum_{j=1}^{\delta+k+1} \sum_{j=1}^{\lceil \frac{\delta+h+1}{2} \rceil} W_{ij} f(x_{ij}, y_{ij}), \quad \text{per ogni } f \in \mathbb{P}^2_\delta, \tag{2.14}$$

dove $\mathbb{P}^2_\delta$ rappresenta lo spazio dei polinomi in due variabili di grado minore o uguale a δ. Inoltre vale che, considerando i parametri u_i, v_j con $i = 0,1,2, \quad j = 0,\ldots,4$ visti in (2.8) e (2.9)

$$h = \begin{cases} 0 & \text{se } u_i = 0,\ i = 0,1,2 \\ 1 & \text{altrimenti} \end{cases}$$

$$k = \begin{cases} 0 & \text{se } u_1 = u_2 = 0 \text{ e } v_i = 0,\ i = 1,\ldots,4 \\ 1 & \text{se } v_3 = v_4 = 0 \text{ e almeno uno tra } u_1, u_2, v_1, v_2 \text{ è diverso da } 0 \\ 2 & \text{se } v_3 \neq 0 \text{ o } v_4 \neq 0 \end{cases}$$

Inoltre in (2.14) abbiamo

$$(x_{ij}, y_{ij}) = U(t_i^{GL}, \theta_j + \mu), \quad 0 < W_{ij} = |\det(JU(t_i^{GL}, \theta_j + \mu))| w_i^{GL} \lambda_j, \tag{2.15}$$

Dove $\{(\theta_j + \mu, \lambda_j)\}$ *sono i nodi angolari e i pesi della formula Gaussiana trigonometrica vista in (2.11) con grado di esattezza* $\delta + k$ *su* $[\alpha, \beta]$, *e* $\left\{(t_i^{GL}, w_i^{GL})\right\}$ *i nodi e i pesi della formule di Gauss-Legendre di grado di esattezza* $\delta + h$ *su* $[0, 1]$.

Dal punto di vista implementativo, la routine Matlab che definisce i nodi e pesi di quadratura di tale formula di tipo prodotto tensoriale, e' disponibile nel pacchetto subp. Queste formule di quadratura verranno implementate per trovare la matrice $\mathbf{P}$ e i momenti m nell'algoritmo di Glaubitz, come vedremo nel **Capitolo 3**. Invece Glaubitz, nei suoi esempi, per trovare la matrice $\mathbf{P}$ e i momenti lavora con domini standard per cui utilizza formule di natura algebrica già note e con il grado di esattezza necessario.

2.3 Domini speciali: settori e segmenti circolari

È importante osservare che grazie alla formula (2.14) possiamo ottenere formule adatte a sezioni del disco, come settori circolari, settori circolari anulari, segmenti circolari, lenti simmetriche ed altri. Infatti, nonostante fossero già presenti delle formule di cubatura per il disco [1] o per sezioni particolari come l'annulo completo [5], mancavano le formule per il caso generale.

Per esempio, si può rappresentare il segmento circolare in almeno tre modi usando il blending lineare di archi ellittici. Consideriamo un disco di raggio R e centro nell'origine, di intervallo angolare $[-\beta, \beta]$, dove il punto medio della corda è $M = (R\cos\beta, 0)$. Allora le possibili rappresentazioni, che si possono vedere in Figura 2.3, sono:

1. $P(\theta) = (R\cos\theta, R\sin\theta), \quad Q(\theta) = (R\cos\theta, -R\sin\theta), \quad \theta \in [0, \beta]$ sono due mezzi archi simmetrici. Quindi, in relazione alla formula (2.1), vale che $A_1 = (R, 0),\ B_1 = (0, R),\ C_1 = (0, 0)$ e $A_2 = (R, 0),\ B_2 = (0, -R), \quad C_2 = (0, 0)$. Di conseguenza, il determinanite Jacobiano corrispondente non dipende da t in quanto

$$\left|\det(JU)\right| = \pm(tu(\theta) + v(\theta)),$$

   e per le formule (2.8)-(2.9), $u_0 = u_1 = u_2 = 0$ e $v_4 = 2R^2 \neq 0$. Allora, secondo il **Teorema 2.1** abbiamo che $h = 0$ e $k = 2$ e, dato che il numero di nodi e pesi equivale a

$$\frac{(\delta + h + 1)(\delta + h + 1)}{2} = \frac{\delta^2}{2} + \mathcal{O}(\delta),$$

   i nodi sono $(\delta+1)(\delta+3)/2$ e, in questo caso, sono disposti come nella parte sinistra di Figura 2.4.

2. Nel secondo metodo, valido per $0 \leq \beta \leq \pi/2$, P e Q sono la corda e l'arco $P(\theta) = (R\cos\beta, R\sin\theta),\ Q(\theta) = (R\cos\theta, R\sin\theta),\ \theta \in [-\beta, \beta]$. Allora, $A_1 = (0, 0),\ B_1 = (0, R),\ C_1 = (R\cos\beta, 0)$ e $A_2 = (R, 0),\ B_2 = (0, R), \quad C_2 = (0, 0)$. Come nel caso precedente, il determinante Jacobiano non dipende da t in quanto $u_0 = u_1 = u_2 = 0$.

Inoltre $v_4 = R^2 \neq 0$, quindi si ha $h = 0$, $k = 2$ e il numero totale di nodi è $(n+1)(n+3)/2$. In questo caso essi sono disposti lungo segmenti orizzontali, come si può vedere nella parte centrale della Figura 2.4 .

3. L'ultimo metodo riguarda i settori ellittici e circolari generalizzati, con il vertice nel punto medio della corda. Abbiamo quindi $P(\theta) \equiv (R\cos\beta, 0)$ e $Q(\theta) = (R\cos\theta, R\sin\theta)$, $\theta \in [-\beta, \beta]$. Di conseguenza, $A_1 = B_1 = (0,0)$, $C_1 = (R\cos\beta, 0)$ e $A_2 = (R,0)$, $B_2 = (0,R)$, $C_2 = (0,0)$. Dato che $v_3 = v_4 = 0$ ma $u_1 = -R^2\cos\beta \neq 0$, allora $h = 1$, $k = 1$. I nodi sono $(n+2)^2/2$ e sono disposti lungo dei raggi che partono dal vertice, come si può vedere nel lato destro della Figura 2.4 .

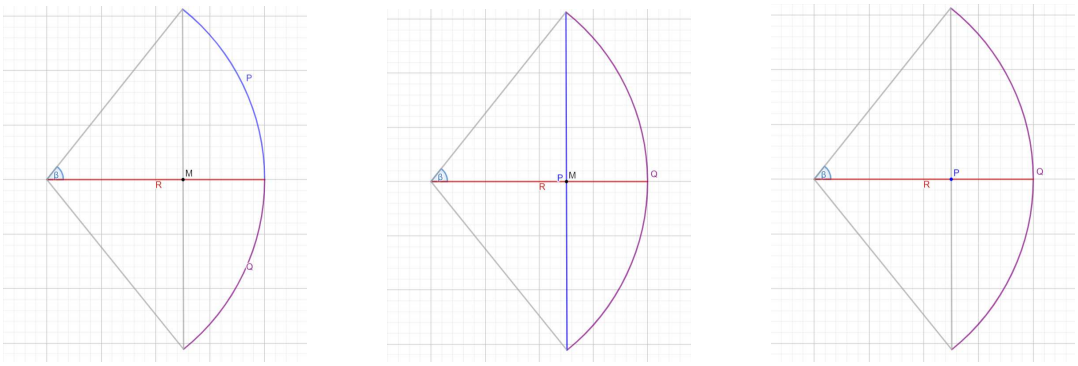


Figura 2.3: Rispettivamente, le tre rappresentazioni del segmento circolare tramite linear blending

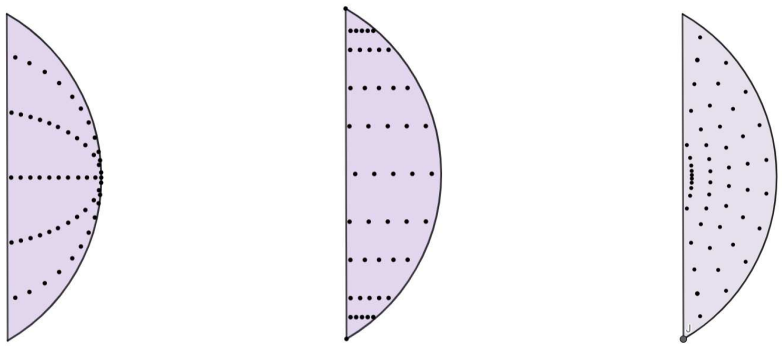


Figura 2.4: distribuzione dei nodi di cubatura per $N = 8$ in un segmento circolare

Infine, nella Figura 2.5 si può vedere la distribuzione dei nodi di cubatura in tre regioni standard dell'ellisse: il segmento ellittico, il settore anulare e la zona ellittica.

In questi esempi abbiamo utilizzato formule di grado 8. Sottolineiamo come domini di questo tipo risultano importanti per la costruzione di formule di cubatura su triangoli sferici e piú in generale poligoni sferici [6]. L'ingrediente fondamentale risulta la cubatura su settori ellittici che sono particolari domini di tipo linear blending di archi ellittici in cui uno di questi degenera a un punto.

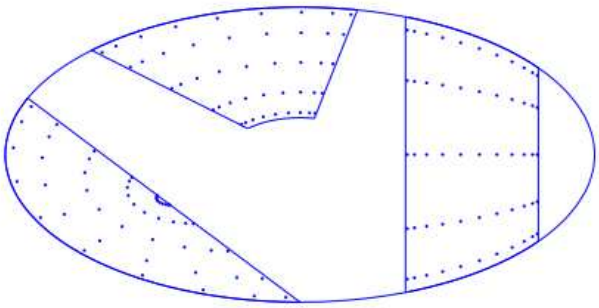


Figura 2.5: Nodi di cubatura, per il grado di esattezza polinomiale $\delta = 8$, di segmento ellittico, settore anulare, zona ellittica.

Capitolo 3

Esperimenti numerici

In questo capitolo ci interessiamo agli esperimenti numerici, svolti ambiente in Matlab.

Nell'articolo originale, J. Glaubitz ha considerato i seguenti esempi

- l'ipercubo $C^d = [-1, 1]^d$, con $d = 2, 3$,
- la palla $B_d = \left\{x \in \mathbb{R}^d | \|x\|_2 \leq 1\right\}$, con $d = 2, 3$,
- il dominio $\Omega = B_2 \cup [1, 2]^2$.

L'apporto di questo lavoro consiste nel testare il metodo su una classe di domini non compresa nell'articolo di Glaubitz, che include quelli di tipo *blending lineare di archi ellittici*, permettendo di considerare settori circolari, segmenti circolari.

3.1 Aspetti comuni degli esperimenti numerici

Nella nostra analisi, per ottenere i risultati richiesti, sará fondamentale

1. la determinazione di punti scattered sul dominio di integrazione, ad esempio mediante una funzione `indomain` che determini se un punto stia o meno nel dominio di integrazione;

2. avere a disposizione una formula di quadratura algebrica di grado n sul dominio di integrazione avente nodi $\{\mathbf{x}_i\}_{1,\dots,N_\delta}$ e rispettivi pesi $\{w_i\}_{1,\dots,N_\delta}$;

3. la determinazione, mediante la formula di cubatura calcolata, di una base ortogonale $\{\phi_l\}_{l=1,\dots,\eta}$ dello spazio dei polinomi $\mathbb{P}_\delta(\Omega)$ di grado δ (come in precedenza η é la dimensione dello spazio dei polinomi di grado totale δ su Ω);

4. infine, ancora mediante la formula di cubatura, il calcolo dei momenti

$$\gamma_k = \int_\Omega \phi_k(P)\, d\Omega = \sum_{i=1}^{N_\delta} w_i \phi_k(\mathbf{x}_i).$$

3.1.1 Determinazione dei punti scattered

Come detto precedentemente, per testare i metodi di Glaubitz su i domini scelti, per prima cosa è importante definire i punti scattered. A tale scopo ci comportiamo come segue.

1. Determiniamo la *bounding box* $\mathcal{R}$ del dominio, ossia il piú piccolo rettangolo con lati paralleli agli cartesiani contenente Ω, tramite la funzione `domain_boundingbox`.

2. Consideriamo il numero N di nodi che vogliamo avere nel nostro dominio. A questo proposito abbiamo considerato l'insieme di punti di Halton, che sono utilizzati in molte applicazioni visto che sono a *bassa discrepanza*, seppure altre scelte possono essere facilmente considerate, come ad esempio i punti di Sobol.

   Sia `ratio`$=\mu(\mathcal{R})/\mu(\Omega)$.

   Se intendiamo generare almeno N punti di Halton Ω, sembra opportuno determinarne almeno $N \cdot$ `ratio` in $\mathcal{R}$. Quindi, se per essere conservativi ne calcoliamo $4 \cdot N \cdot$ `ratio` in $\mathcal{R}$ risulterá molto probabile che ne avremo almeno N in Ω. Questo procedimento viene svolto dalla funzione `define_scattered_pointset`.

3. Mediante una opportuna routine `indomain`, variabile col dominio di integrazione, individiamo i punti di Halton appartenenti a Ω.

3.1.2 Calcolo di una base ortonormale

Dopo aver calcolato mediante le routines introdotte una formula di quadratura algebrica di grado δ valida sul dominio di integrazione avente nodi $\{x_i\}_{1,\dots,N_\delta}$ e rispettivi pesi $\{w_i\}_{1,\dots,N_\delta}$, risulta fondamentale calcolare una base ortonormale $\{\phi_l\}_{l=1,\dots,\eta}$ dello spazio dei polinomi $\mathbb{P}_\delta$ di grado δ, avente dimensione η. Questa servirà per calcolare la matrice $\mathbf{P}$ precedentemente introdotta.

In teoria, per trovare una base ortogonale di polinomi di $\mathbb{P}_\delta$ basterebbe partire da una base qualunque di $\mathbb{P}_\delta$ e ortogonalizzarla con il metodo di Gram-Schmidt. Tuttavia, soprattutto a seconda della scelta della base iniziale, questo metodo può portare a delle matrici di Vandermonde e Gram malcondizionate. Di seguito poniamo η la dimensione di $\mathbb{P}_\delta$.

Ricordiamo brevemente la definizione di queste ultime. La matrice di Gram, considerando gli spazi di polinomi, serve a verificare l'ortogonalità di essi. Se consideriamo quindi una base di polinomi $(p_1, \dots, p_\eta)$ e un prodotto scalare $[\cdot,\cdot]_\eta$, la matrice di Gram associata è

$$\begin{pmatrix} [p_1,p_1]_\eta & \dots & [p_1,p_k]_\eta \\ \vdots & & \vdots \\ [p_k,p_1]_\eta & \dots & [p_k,p_k]_\eta \end{pmatrix}$$

Invece, una matrice di Vandermonde, data una base di polinomi $(p_1, \dots, p_\eta)$ di $\mathbb{P}^d_\delta(\Omega)$, e i nodi $(x_1, \dots, x_M)$ è data da

$$\begin{pmatrix} p_1(x_1) & p_2(x_1) & \dots & p_\eta(x_1) \\ p_1(x_2) & p_2(x_2) & \dots & p_\eta(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ p_1(x_M) & p_2(x_M) & \dots & p_\eta(x_M) \end{pmatrix}$$

Quindi, per quanto detto sopra, non useremo l'ortogonalizzazione di Gram-Schmidt per ricavare la base ortogonale ma il metodo che si può vedere in [8].

Consideriamo la formula di cubatura positiva con grado di esattezza almeno 2δ e cardinalità $M = M_{2\delta}$.

$$\int_\Omega f(x)d\mu = \sum_{i=1}^{M} w_i f(x_i), \qquad \text{per ogni } f \in \mathbb{P}^d_{2\delta}(\Omega) \tag{3.1}$$

Nel nostro caso, quindi, la formula (2.14) è adatta. Data una base di polinomi $\langle p_1, \dots, p_\eta \rangle = \mathbb{P}^d_\delta(\Omega)$, consideriamo la matrice di Vandermonde

$$V_p = (v_{ij}) = (p_j(x_i)) \in \mathbb{R}^{M \times \eta} \tag{3.2}$$

e la matrice diagonale

$$\sqrt{W} = \begin{pmatrix} \sqrt{w_1} & 0 & \dots & 0 \\ 0 & \sqrt{w_2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sqrt{w_M} \end{pmatrix} \tag{3.3}$$

Ora, si può osservare che la matrice $G_p = (g_{hk}) = (\sqrt{W}V_p)^t(\sqrt{W}V_p)$ è una matrice di Gram definita positiva per il prodotto generato dalla misura $d\mu$,

$$g_{hk} = \sum_{i=1}^{M} \sqrt{w_i}p_h(x_i)\sqrt{w_i}p_k(x_i) = \int_\Omega p_h(x)p_h(x)d\mu, \quad h \geq 1, k \leq \eta. \tag{3.4}$$

Allora, le matrici $\sqrt{W}V_p$ e V_p sono a rango pieno e si può applicare una scomposizione QR per cui

$$\sqrt{W}V_p = QR, \quad Q \in \mathbb{R}^{M \times \eta}, \quad R \in \mathbb{R}^{\eta \times \eta}, \tag{3.5}$$

dove Q è una matrice ortogonale e R una matrice triangolare superiore non singolare. Se a questo punto consideriamo la matrice $V_\phi = V_p R^{-1}$, essa soddisfa la proprietà

$$G_\phi = (\sqrt{W}V_\phi)^t(\sqrt{W}V_\phi) = Q^tQ = I, \tag{3.6}$$

il che vuole dire che la base di polinomi $(\phi_1, \dots, \phi_\eta) = (p_1, \dots, p_\eta)R^{-1}$ è ortogonale.

Tuttavia, lavorando con macchine con una precisione finita, il risultato può essere diverso. Infatti, se la matrice di Vandermonde V_p è estremamente malcondizionata, la matrice Q non risulterà ortogonale. In questo caso, si è visto come una seconda scomposizione Q_1R_1 della matrice Q possa essere la soluzione. Infatti, si può scrivere che

$$\sqrt{W}V_\phi = \sqrt{W}V_pR^{-1} = Q_1R_1, \tag{3.7}$$

dove $V_\phi = V_p R{-}1$ è ben condizionata, così che la matrice di Vandermonde

$$V_q = V_p R^{-1} R_1^{-1} \tag{3.8}$$

sia numericamente ortogonale e la base $(q_1, \ldots, q_\eta) = (p_1, \ldots, p_\eta) R^{-1} R_1^{-1}$ sia la base ortogonale cercata. Questo processo è stato implementato in MATLAB tramite la funzione `multivop`.

Infine, grazie alla formula di cubatura e la base di polinomi appena calcolata, procederemo col calcolare numericamente i momenti

$$\gamma_k = \int_\Omega \phi_k(P)\, d\Omega = \sum_{i=1}^{M_\delta} w_i \phi_k(x_i), \quad k = 1, \ldots, \eta, \tag{3.9}$$

mediante una formula avente grado di precisione almeno δ.

3.2 Esperimenti su un settore circolare

In questa sezione cominciamo la nostra analisi prendendo in considerazione un settore circolare. Piú esattamente trattiamo il caso in cui esso abbia raggio $R = 1$, $\alpha = \pi/4$, $\beta = \pi/2$, $C = [1,\ 1.5]$, ma d'altra parte piú in generale, senza difficoltá si puó prendere in considerazione il caso in cui R sia un generico numero reale positivo e l'intervallo angolare $[\alpha, \beta]$ sia tale che $-\pi \leq \alpha < \beta \leq \pi]$.

In Figura 3.1, mostriamo il settore considerato.

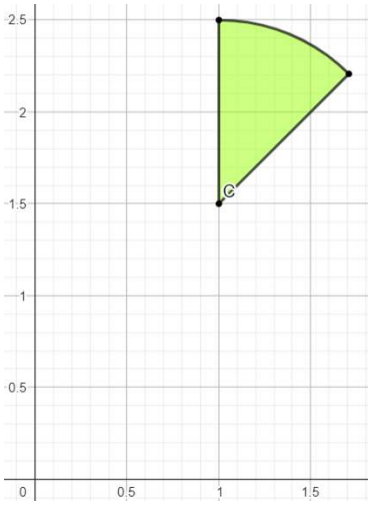


Figura 3.1: Settore circolare di raggio 1 e intervallo angolare $\alpha = \pi/4$, $\beta = \pi/2$.

Vediamo, per prima cosa, come implementare la funzione `indomain` per il settore circolare.

Un ingrediente richiesto per la definizione dei punti scattered, é la determinazione della *bounding box* del settore. Nel nostro esempio risulta evidentemente molto facile, essendo

definita una volta nota i vertici. Il caso generale, potenzialmente piú complesso, é implementato nella routine `domain_boundingbox.m`, che va a considerare tutte le coordinate significative dei punti del dominio, raccolte in `boxVx`, e ne isola i valori minimi e massimi sia per le coordinate x che y. Quindi, la *bounding box* del settore circolare è il più piccolo rettangolo che lo contiene, ossia il rettangolo $[x_{min}, x_{max}] \times [y_{min}, y_{max}]$.

A questo punto, la funzione `define_scattered` genera il numero sufficiente di punti di Halton all'interno di $\mathcal{R}$ con il procedimento esposto precedentemente. Per stabilire se appartengono al dominio o no, si utilizza infine la funzione `indomain`. Sia $P = r\cos(\theta)$ un punto che è stato generato nella bounding box del settore circolare, se $r \in [0, R]$ e $\theta \in [\theta_1, \theta_2]$, il punto verrà tenuto, altrimenti verrà scartato.

Utilizzando la notazione vista nel **Capitolo 2**, possiamo scrivere il settore circolare come combinazione convessa delle curve

$$P(\theta) = A_1\cos(\theta) + B_1\sin(\theta) + C_1, \quad Q(\theta) = A_2\cos(\theta) + B_2\sin(\theta) + C_2 \tag{3.10}$$

Nel nostro caso, $P(\theta) = (0,0)$ e $Q(\theta) = (R\cos(\theta), R\sin(\theta))$, quindi vale che $A_1 = B_1 = C_1 = (0,0)$ e $A_2 = (R,0)$, $B_2 = (0,R)$ e $C_2 = (0,0)$. Questo, come abbiamo visto, è fondamentale per poi applicare la formula di cubatura (2.14) su questo dominio.

3.2.1 Test Numerici

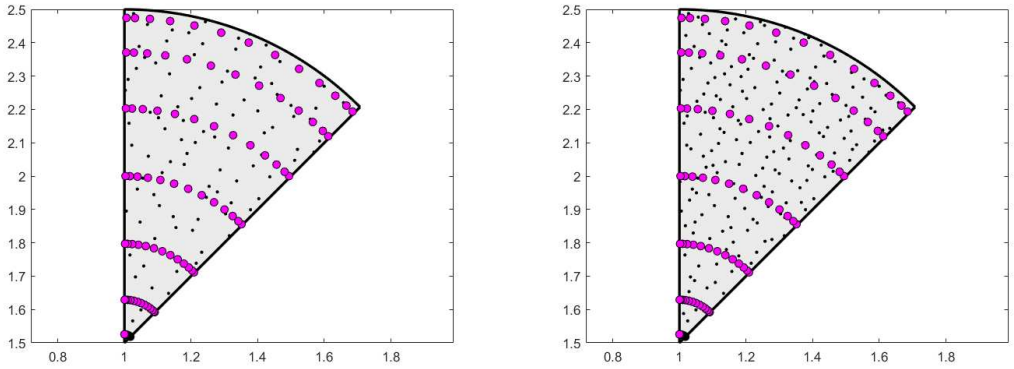
Per prima cosa, vediamo come avviene la distribuzione dei punti di Halton generati, al variare della cardinalità N della formula. Nella Figura 3.2, abbiamo rappresentato

- i nodi della formula di Glaubitz con dei punti in nero,
- i nodi della formula di riferimento implementata con `gqellblend` di grado 12 (cf. [7]) con punti piú grandi in violetto.

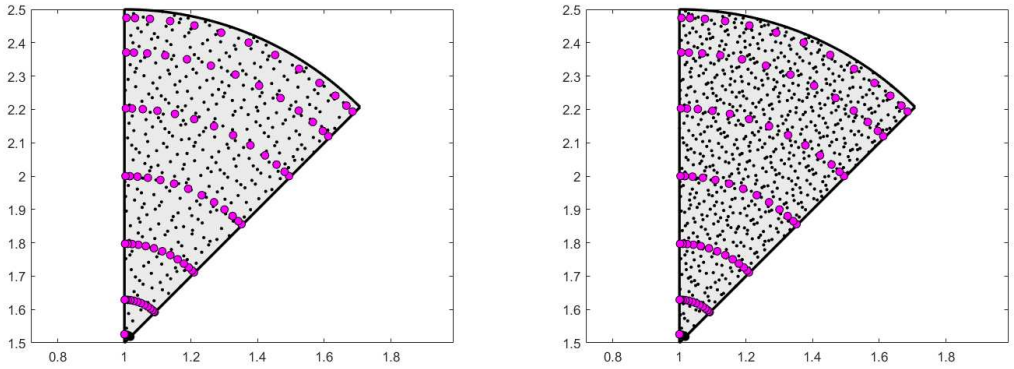
La formula di riferimento genera il valore dell'integrale che useremo come base per calcolare gli errori assoluti e relativi.

Successivamente, nella Tabella 3.1, vedremo come cambiano i seguenti parametri al variare di N, tramite la routine `demo_glaubitz_2D`, ovvero

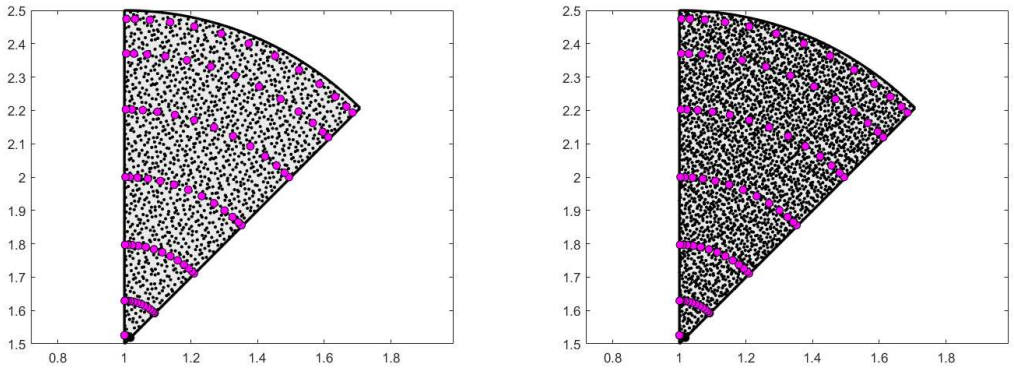
- grado di precisione δ della formula di Glaubitz,
- cardinalità `Card Ref` della formula di riferimento,
- cardinalità `Card GL` della formula di Glaubitz,
- dimensione η dello spazio di polinomi $\mathbb{P}^d_\delta(\Omega)$, dove $d = 2$ in quanto le integrande sono funzioni in due variabili,
- errore assoluto $\epsilon_a = \left|I_r - I\right|$, dove I_r è l'integrale considerato esatto calcolato tramite la formula di riferimento con grado 40, e I l'integrale approssimato,
- errore relativo $\epsilon_r = \dfrac{\left|I_r - I\right|}{\left|I\right|}$,



(a) $N = 100$ (b) $N = 200$



(c) $N = 400$ (d) $N = 800$



(e) $N = 1600$ (f) $N = 3200$

Figura 3.2: In nero: distribuzione nodi di Glaubitz al variare di N. In rosa: distribuzione nodi della formula di riferimento con grado 12.

- tempo di calcolo t, misurato in secondi.

Divideremo gli esperimenti in base alla formula di cubatura utilizzata, LS o ℓ^1, e la funzione integranda presa in considerazione, tra le funzioni $f(x,y) = e^{-x^2-y^2}$, $f(x,y) = \frac{1}{x^2+y^2}$.

Tabella 3.1

(a) Formula LS, $f(x,y) = e^{-x^2-y^2}$ sul settore circolare

N	δ	Card REF	Card GL	η	ϵ_a	ϵ_r	t
100	6	861	100	28	$1.8 \cdot 10^{-9}$	$1.0 \cdot 10^{-6}$	$1.39s$
200	7	861	200	36	$1.9 \cdot 10^{-10}$	$1.1 \cdot 10^{-7}$	$1.31s$
400	7	861	400	36	$4.8 \cdot 10^{-10}$	$2.8 \cdot 10^{-7}$	$1.29s$
800	8	861	800	45	$7.5 \cdot 10^{-12}$	$4.3 \cdot 10^{-9}$	$1.31s$
1600	8	861	1600	45	$1.8 \cdot 10^{-12}$	$1.0 \cdot 10^{-9}$	$1.30s$
3200	8	861	3200	45	$3.3 \cdot 10^{-12}$	$1.9 \cdot 10^{-9}$	$1.56s$

(b) Formula ℓ^1, $f(x,y) = e^{-x^2-y^2}$ sul settore circolare

N	δ	Card REF	Card GL	η	ϵ_a	ϵ_r	t
100	8	861	27	45	$1.1 \cdot 10^{-8}$	$6.4 \cdot 10^{-6}$	$1.47s$
200	7	861	25	36	$3.2 \cdot 10^{-7}$	$1.9 \cdot 10^{-4}$	$1.34s$
400	8	861	28	45	$2.1 \cdot 10^{-8}$	$1.2 \cdot 10^{-5}$	$1.56s$
800	8	861	27	45	$2.7 \cdot 10^{-7}$	$1.5 \cdot 10^{-4}$	$1.27s$
1600	8	861	27	45	$1.3 \cdot 10^{-8}$	$7.6 \cdot 10^{-6}$	$1.22s$
3200	8	861	27	45	$4.9 \cdot 10^{-8}$	$2.8 \cdot 10^{-5}$	$1.91s$

(c) Formula LS, $f(x,y) = \frac{1}{(1+x^2)(1+y^2)}$ sul settore circolare

N	δ	Card REF	Card GL	η	ϵ_a	ϵ_r	t
100	6	861	100	28	$1.5 \cdot 10^{-9}$	$5.0 \cdot 10^{-8}$	$1.22s$
200	7	861	200	36	$1.6 \cdot 10^{-10}$	$5.4 \cdot 10^{-9}$	$1.04s$
400	7	861	400	36	$4.9 \cdot 10^{-11}$	$1.6 \cdot 10^{-9}$	$1.15s$
800	8	861	800	45	$1.1 \cdot 10^{-12}$	$3.7 \cdot 10^{-11}$	$1.41s$
1600	8	861	1600	45	$2.2 \cdot 10^{-13}$	$7.3 \cdot 10^{-12}$	$1.26s$
3200	8	861	3200	45	$1.3 \cdot 10^{-12}$	$4.4 \cdot 10^{-11}$	$1.43s$

(d) Formula ℓ^1, $f(x,y) = \frac{1}{(1+x^2)(1+y^2)}$ sul settore circolare

N	δ	Card REF	Card GL	η	ϵ_a	ϵ_r	t
100	8	861	27	45	$2.3 \cdot 10^{-9}$	$7.7 \cdot 10^{-8}$	$1.30s$
200	7	861	25	36	$7.2 \cdot 10^{-8}$	$2.4 \cdot 10^{-6}$	$1.27s$
400	8	861	28	45	$2.6 \cdot 10^{-8}$	$8.9 \cdot 10^{-7}$	$1.37s$
800	8	861	27	45	$6.6 \cdot 10^{-8}$	$2.2 \cdot 10^{-6}$	$1.46s$
1600	8	861	27	45	$1.7 \cdot 10^{-9}$	$5.6 \cdot 10^{-8}$	$1.43s$
3200	8	861	27	45	$2.7 \cdot 10^{-8}$	$9.2 \cdot 10^{-7}$	$1.73s$

3.2.2 Analisi dei risultati

In questa sezione esponiamo i commenti che si possono fare circa questi risultati.

Per prima cosa, possiamo osservare che in tutti i casi gli errori assoluti e relativi, che variano tra gli ordini 10^{-5} e 10^{-13}, tendono a diminuire all'aumentare di N. Questa era il comportamento atteso in quanto una formula con maggiore cardinalità e grado di precisione, dovrebbe in linea di massima offrire una rappresentazione più accurata della funzione integranda.

Per quanto riguarda le formule LS e ℓ^1, si può osservare che per la prima la cardinalità della formula di Glaubitz, ossia il numero di punti effettivamente usati dalla formula, aumenta con N, e dunque utilizza tutti i punti a disposizione. Invece, nella formula ℓ^1 la cardinalità rimane quasi costante e varia solo tra le cifre 25, 27, 28. Inoltre, nonostante la formula ℓ^1 produca comunque errori accettabili, essi sono in generale maggiori rispetto a quelli prodotti dalla formula LS a parità di N.

Si può quindi concludere sperimentalmente che la formula LS risulti più precisa, nonostante richieda una cardinalità più alta

Entrambe le formule raggiungono lo stesso grado di precisione, tra 6 e 8. Però si può osservare che il grado della formula LS aumenta a seconda della cardinalità, mentre quello della formula ℓ^1 rimane quasi costante.

Come avevamo visto quando avevamo descritto l'algoritmo di Glaubitz, la dimensione η dello spazio di polinomi $\mathbb{P}^2_\delta(\Omega)$ per le integrande che abbiamo considerato, è pari a

$$\eta = \dim(\mathbb{P}^2_\delta(\Omega)) = \binom{\delta+2}{2}$$

Per quanto riguarda le due integrande, $f(x,y) = e^{-x^2-y^2}$ e $f(x.y) = \frac{1}{x^2+y^2}$, possiamo dire che l'uso della prima, in generale, produce errori minori rispetto alla seconda.

3.3 Esperimenti sulle lenti simmetriche

Prendiamo ora in considerazione l'esempio di una lente simmetrica. Questo dominio coincide con l'intersezione di due circonferenze con centri simmetrici rispetto a un'asse comune e raggi uguali. In questo caso specifico, tali circonferenze hanno centri rispettivamente $[1,0]$, $[-1,0]$ e raggio 2.5, si veda la Figura 3.3.

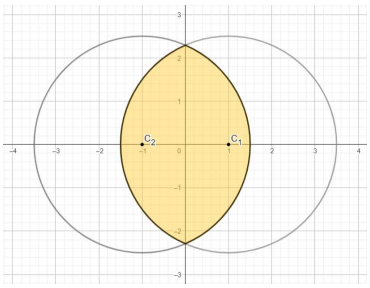


Figura 3.3: Lenti simmetriche con centri rispettivamente [1,0],[-1,0] e raggi uguali a 2.5.

Vediamo come implementare la funzione `indomain` per le lenti simmetriche. La *bounding box* è data dal rettangolo $[x_1, x_2] \times [y_1, y_2]$ dove $[x_1, x_2] = [a - R, -a + R]$ e $[y_1, y_2] = [-\sqrt{R^2 - a^2}, \sqrt{R^2 - a^2}]$. Quindi, nel nostro esempio

$$\mathcal{R} = \Big[-\frac{3}{2}, \frac{3}{2} \Big] \times \Big[-\frac{\sqrt{21}}{2}, \frac{\sqrt{21}}{2} \Big].$$

A questo punto la funzione `define_scattered` genera i punti di Halton all'interno di $\mathcal{R}$. La funzione `indomain`, dato il punto $P = (x, y)$ da analizzare, calcola le distanze

$$\texttt{dist1} = \sqrt{(x - (-a))^2 + (y - 0)^2} = \sqrt{(x + a)^2 + y^2}$$

,

$$\texttt{dist2} = \sqrt{(x - a^2) + (y - 0)^2} = \sqrt{(x - a)^2 + y^2}.$$

Se $\max($`dist1`, `dist2`$) \leq R$, allora stabilisce che il punto é interno al dominio, al piú sulla frontiera, altrimenti risulta esterno.

Infine, come abbiamo già visto, le lenti simmetriche possono essere viste come una combinazione convessa delle curve

$$P(\theta) = A_1 \cos(\theta) + B_1 \sin(\theta) + C_1, \quad Q(\theta) = A_2 \cos(\theta) + B_2 \sin(\theta) + C_2, \tag{3.11}$$

In questo caso si può scrivere che $P(\theta) = (R\cos(\theta) - a, R\sin(\theta))$ e $Q(\theta) = (-R\cos(\theta) + a, R\sin(\theta))$, quindi con $A_1 = (R, 0)$, $B_1 = (0, R)$, $C_1 = (-a, 0)$ e $A_2 = (-R, 0)$, $B_2 = (0, R)$, $C_2 = (a, 0)$.

3.3.1 Test Numerici

Nella Figura 3.4 mostriamo

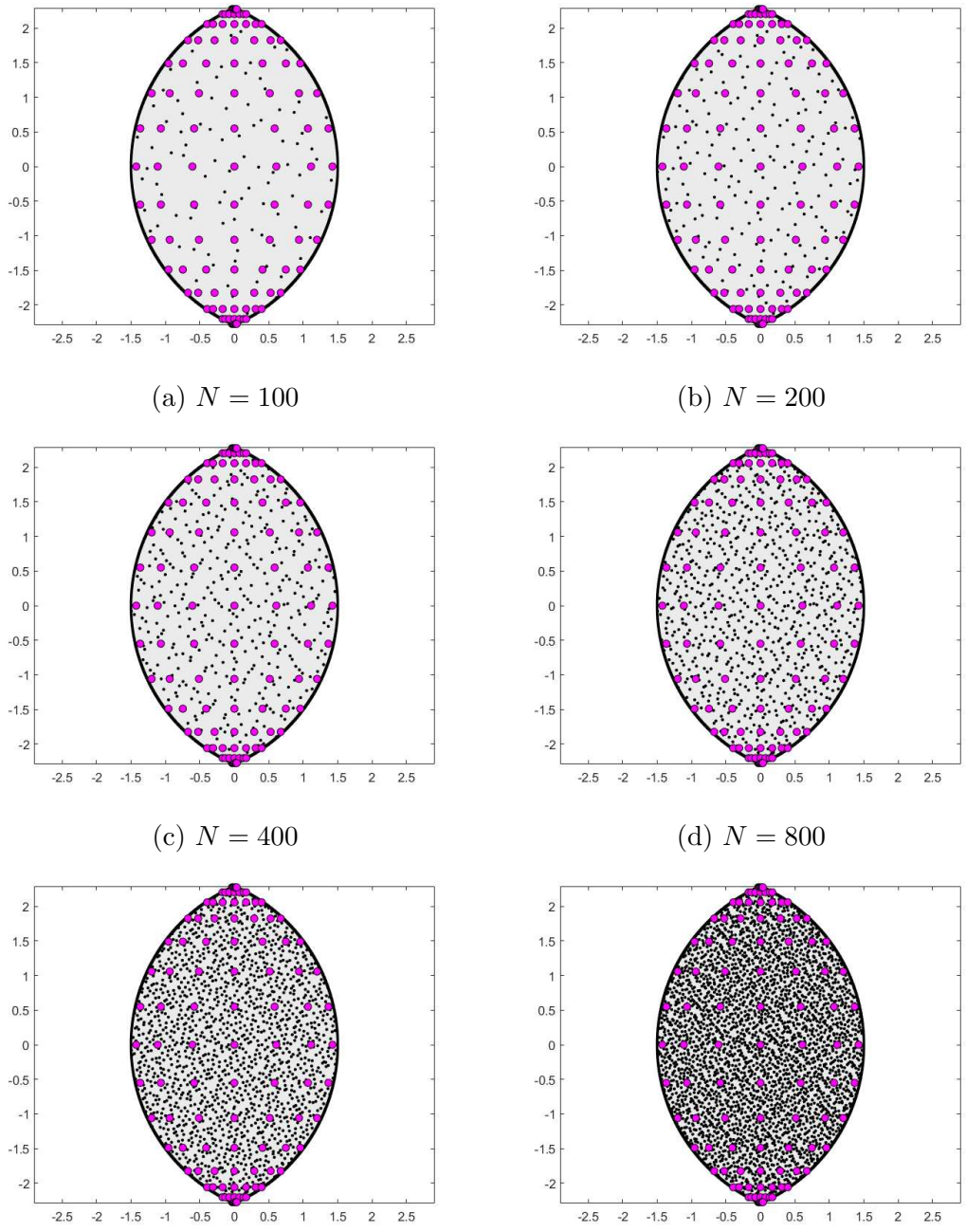
- i nodi della formula di Glaubitz con dei punti in nero,
- i nodi della formula di riferimento implementata con `gqellblend` di grado 12 (cf. [7]) con punti piú grandi in violetto.

Inoltre nella Tabella 3.2 vediamo come cambiano i seguenti parametri al variare di N

- grado di precisione δ della formula di Glaubitz,
- cardinalità `Card Ref` della formula di riferimento,
- cardinalità `Card GL` della formula di Glaubitz,
- dimensione dello spazio di polinomi $\mathbb{P}^2_\delta(\Omega)$,
- errore assoluto $\epsilon_a = \big|I_r - I\big|$, dove I_r è l'integrale esatto calcolato tramite la formula di riferimento con grado 40, e I l'integrale approssimato,
- errore relativo $\epsilon_r = \frac{\big|I_r - I\big|}{\big|I\big|}$,

- tempo di calcolo t, misurato in secondi.

Divideremo nuovamente gli esperimenti in base a formula e funzione integranda utilizzate.



(a) $N = 100$ (b) $N = 200$

(c) $N = 400$ (d) $N = 800$

(e) $N = 1600$ (f) $N = 3200$

Figura 3.4: In nero: distribuzione nodi di Halton per le formule di Glaubitz al variare di N. In rosa: distribuzione nodi della formula di riferimento a grado 12.

Tabella 3.2

(a) Formula LS, $f(x,y) = e^{-x^2-y^2}$ sulle lenti simmetriche

N	δ	Card REF	Card GL	η	ϵ_a	ϵ_r	t
100	5	903	100	21	$6.4 \cdot 10^{-2}$	$2.1 \cdot 10^{-2}$	$0.85s$
200	7	903	200	36	$5.9 \cdot 10^{-3}$	$2.0 \cdot 10^{-3}$	$0.58s$
400	8	903	400	45	$5.1 \cdot 10^{-5}$	$1.7 \cdot 10^{-5}$	$0.79s$
800	11	903	800	78	$1.2 \cdot 10^{-4}$	$4.2 \cdot 10^{-5}$	$0.85s$
1600	16	903	1600	153	$2.2 \cdot 10^{-7}$	$7.5 \cdot 10^{-8}$	$0.96s$
3200	20	903	3200	231	$1.5 \cdot 10^{-9}$	$4.9 \cdot 10^{-10}$	$1.78s$

(b) Formula ℓ^1, $f(x,y) = e^{-x^2-y^2}$ sulle lenti simmetriche

N	δ	Card REF	Card GL	η	ϵ_a	ϵ_r	t
100	6	903	28	28	$3.5 \cdot 10^{-2}$	$1.2 \cdot 10^{-2}$	$0.72s$
200	9	903	55	55	$1.8 \cdot 10^{-3}$	$6.2 \cdot 10^{-4}$	$0.78s$
400	11	903	78	78	$2.2 \cdot 10^{-3}$	$7.5 \cdot 10^{-4}$	$1.14s$
800	15	903	136	136	$8.8 \cdot 10^{-6}$	$3.0 \cdot 10^{-6}$	$5.72s$
1600	20	903	265	231	$3.4 \cdot 10^{-8}$	$1.1 \cdot 10^{-8}$	$45.59s$
3200	20	903	342	231	$4.4 \cdot 10^{-8}$	$1.5 \cdot 10^{-8}$	$159.68s$

(c) Formula LS, $f(x,y) = \frac{1}{(1+x^2)(1+y^2)}$ sulle lenti simmetriche

N	δ	Card REF	Card GL	η	ϵ_a	ϵ_r	t
100	5	903	100	21	$5.1 \cdot 10^{-2}$	$1.2 \cdot 10^{-2}$	$0.83s$
200	7	903	200	36	$6.6 \cdot 10^{-3}$	$1.6 \cdot 10^{-3}$	$1.59s$
400	8	903	400	45	$8.5 \cdot 10^{-5}$	$2.1 \cdot 10^{-5}$	$1.23s$
800	11	903	800	78	$6.3 \cdot 10^{-4}$	$1.5 \cdot 10^{-4}$	$1.2657s$
1600	16	903	1600	153	$1.7 \cdot 10^{-5}$	$4.1 \cdot 10^{-6}$	$1.75s$
3200	20	903	3200	231	$2.2 \cdot 10^{-7}$	$5.4 \cdot 10^{-8}$	$1.76s$

(d) Formula ℓ^1, $f(x,y) = \frac{1}{(1+x^2)(1+y^2)}$ sulle lenti simmetriche

N	δ	Card REF	Card GL	η	ϵ_a	ϵ_r	t
100	6	903	28	28	$4.9 \cdot 10^{-2}$	$1.2 \cdot 10^{-2}$	$0.79s$
200	9	903	55	55	$1.0 \cdot 10^{-2}$	$2.5 \cdot 10^{-3}$	$0.73s$
400	11	903	78	78	$1.4 \cdot 10^{-2}$	$3.5 \cdot 10^{-3}$	$1.27s$
800	15	903	136	136	$3.5 \cdot 10^{-4}$	$8.6 \cdot 10^{-5}$	$5.49s$
1600	20	903	265	231	$3.2 \cdot 10^{-5}$	$7.7 \cdot 10^{-6}$	$45.99s$
3200	20	903	342	231	$7.6 \cdot 10^{-5}$	$1.8 \cdot 10^{-5}$	$168.99s$

3.3.2 Analisi dei Risultati

Anche in questo caso possiamo notare che, come ci si aspettava, all'aumentare di N, gli errori assoluti e relativi diminuiscono. Essi, tuttavia, sono maggiori rispetto agli errori dell'esempio precedente. Infatti, per le lenti simmetriche, aventi una geometria meno regolare, possono arrivare all'ordine di 10^{-2}, mentre nel settore circolare avevamo visto errori di ordine massimo 10^{-5}.

Di nuovo, possiamo osservare che la cardinalità della formula LS è sempre pari a N, dunque utilizza tutti i punti a disposizione, a differenza della formula ℓ^1, che ne usa un numero minore. Avevamo riscontrato questa caratteristica anche nel primo esempio, tuttavia nel settore circolare la cardinalità della formula ℓ^1 risultava decisamente minore e rimaneva quasi costante, variando solo tra 25, 27 e 28. Qui, invece, cresce all'aumentare di N e raggiunge cifre molto più alte. Come nell'esempio precedente, gli errori assoluti e relativi sono in generale minori nella formula LS, mostrando maggiore precisione.

Il grado di esattezza cresce in modo simile, all'aumentare di N, per entrambe le formule. Tuttavia, a differenza di quello che avevamo visto nel settore circolare, subisce un aumento importante, partendo da un valore iniziale da 5 e raggiungendo progressivamente 20.

Per quanto riguarda le due integrande, $f(x, y) = e^{-x^2-y^2}$ e $f(x, y) = \frac{1}{(1+x^2)(1+y^2)}$, la prima presenta, a parità di N, errori più piccoli rispetto alla seconda.

Bibliografia

[1] R. Cools e K.J. Kim. *A survey of known and new cubature formulas for the unit disk*, Korean Journal of Computational and Applied Mathematics 7 (2000), pp. 477–485.

[2] G. Da Fies e M. Vianello. *Trigonometric Gaussian quadrature on subintervals of the period*, Electronic Transactions on Numerical Analysis 39 (2012), pp. 102–112.

[3] W. Gautschi. *Orthogonal polynomials: applications and computation*, Acta Numerica 5 (1996), pp. 45–119.

[4] J. Glaubitz. *Stable high-order cubature formulas for experimental data*, Journal of Computational Physics 447 (2021).

[5] W.H. Pierce. *Numerical integration over the planar annulus*, Journal of the Society for Industrial and Applied Mathematics 5.2 (1957), pp. 66–73.

[6] A. Sommariva. *Numerical cubature and hyperinterpolation over Spherical Polygons*, sottomesso per la pubblicazione a Appl. Math. Comput (2024).

[7] A. Sommariva e M. Vianello. *Algebraic cubature by linear blending of elliptical arcs*, Appl. Numer. Math. 74 (2013), pp. 49–61.

[8] A. Sommariva e M. Vianello. *Numerical hyperinterpolation over nonstandard planar regions*, Mathematics and Computers in Simulation 141 (2017), pp. 110–120.

Appendice A

Codici

In questa sezione vengono riportate in calce le principali routines utili nella realizzazione di questa tesi, con i relativi commenti ed eventuale documentazione. Gli stessi codici sono disponibili open-source al link:

https://github.com/pangraziosara/Routines

```
function domain_structure=define_domain(domain_str,varargin)

domain_structure.domain=domain_str;

args = varargin;
L=length(args);
k=1;

switch domain_str

    case 'polygon'

        % ..... setting defaults .....
        %
        % domain_structure.vertices:
        % 0. it is a matrix N x 2;
        % 1. each row consists of the coordinates of a vertex of the domain
        %    in cartesian coordinates;

        % setting defaults
        example_type=6;
        domain_structure=gallery_domains_2D(domain_str,example_type);

        while k <= L
            strL=args{k};
            switch strL
                case 'vertices'
                    domain_structure.vertices=args{k+1};
            end
            k=k+2;
        end
```

```
    vertices=domain_structure.vertices;

    if norm(vertices(1,:)-vertices(end,:)) == 0
        vertices=vertices(1:end-1,:);
    end

    XV=vertices(:,1); YV=vertices(:,2);

    domain_structure.polyshape=polyshape(XV,YV);



case 'disk'

    % ..................... setting defaults ......................
    %
    % domain_structure.center:
    % 0. it is a vector 1 x 2; it consists of the coordinates of the
    % center of the circular annular sector;
    %
    % domain_structure.radius:
    % 0. it is a vector of length 1;
    % 1. first value consists of the radius of the first disk;
    % 2. second value consists of the radius of the second disk;


    % setting defaults
    domain_structure.center=[0 0];
    domain_structure.radius=1;

    while k <= L
        strL=args{k};
        switch strL
            case 'center'
                % setting center as a row vector
                value=args{k+1}; value=(value(:))';
                domain_structure.center=value;
            case 'radius'
                domain_structure.radius=args{k+1};
        end
        k=k+2;
    end


    % ellblend structure (as sector)

    R2=0;
    R1=domain_structure.radius;
    C=domain_structure.center;

    domain_structure.A=[R1 0; R2 0];
    domain_structure.B=[0 R1; 0 R2];
    domain_structure.C=[C; C];
    domain_structure.alpha=0;
    domain_structure.beta=2*pi;
```

```
case 'lune'

    % ..... setting defaults .....
    %
    % domain_structure.centers:
    % 0. it is a matrix 2 x 2;
    % 1. first row consists of the coordinates of the center of the
    %    first lune;
    % 2. second row consists of the coordinates of the center of the
    %    second lune;
    %
    % domain_structure.radii:
    % 0. it is a vector of length 2;
    % 1. first value consists of the radius of the first lune;
    % 2. second value consists of the radius of the second lune;

    domain_structure.centers=[0.5 0.5; 1.1 0.5];
    domain_structure.radii=[0.5; 0.5];

    while k <= L
        strL=args{k};
        switch strL
            case 'centers'
                domain_structure.center=args{k+1};
            case 'radii'
                domain_structure.radius=args{k+1};
        end
        k=k+2;
    end


    domain_structure.NURBS=ellblend2NURBS(domain_structure);



case 'circular-annular-sector'

    % ..... setting defaults .....
    %
    % domain_structure.center:
    % 0. it is a vector 1 x 2; it consists of the coordinates of the
    % center of the circular annular sector;
    %
    % domain_structure.radii:
    % 0. it is a vector of length 2;
    % 1. first value consists of the radius of the first disk;
    % 2. second value consists of the radius of the second disk;
    %
    % domain_structure.angles:
    % 0. it is a vector of length 2;
    % 1. the first component is the parameter "alpha" and must be in
    %    [-pi,pi]
    % 2. the second component is the parameter "beta" and must be in
    %    [-pi,pi]


    % default example:
```

```
    domain_structure.center=[0 0];
    domain_structure.radii=[0.5 2];
    domain_structure.angles=[pi/4; pi/2];

    while k <= L
        strL=args{k};
        switch strL
            case 'center'
                % setting center as a row vector
                value=args{k+1}; value=(value(:))';
                domain_structure.center=value;
            case 'radii'
                domain_structure.radius=args{k+1};
            case 'angles'
                domain_structure.angles=args{k+1};
        end
        k=k+2;
    end

    domain_structure.NURBS=ellblend2NURBS(domain_structure);


    % ellblend structure

    R=domain_structure.radii;
    R1=R(1);
    R2=R(2);
    C=domain_structure.center;
    thetaV=domain_structure.angles;

    domain_structure.A=[R1 0; R2 0];
    domain_structure.B=[0 R1; 0 R2];
    domain_structure.C=[C; C];
    domain_structure.alpha=thetaV(1);
    domain_structure.beta=thetaV(2);



case 'sector'

    % ......................... setting defaults ........................
    %
    % domain_structure.center:
    % 0. it is a vector 1 x 2; it consists of the coordinates of the
    % center of the sector;
    %
    % domain_structure.radius:
    % 0. it is a vector of length 1;
    % 1. it consists of the radius of the disk;
    %
    % domain_structure.angles:
    % 0. it is a vector of length 2;
    % 1. the first component is the parameter "alpha" and must be in
    %    [-pi,pi]
    % 2. the second component is the parameter "beta" and must be in
    %    [-pi,pi]
```

```
    % default example:
    domain_structure.center=[1 1.5];
    domain_structure.radius=1;
    domain_structure.angles=[pi/4; pi/2];

    while k <= L
        strL=args{k};
        switch strL
            case 'center'
                % setting center as a row vector
                value=args{k+1}; value=(value(:))';
                domain_structure.center=value;
            case 'radius'
                domain_structure.radius=args{k+1};
            case 'angles'
                domain_structure.angles=args{k+1};
        end
        k=k+2;
    end

    domain_structure.NURBS=ellblend2NURBS(domain_structure);


    % ellblend structure (as symmetric annular sector)

    R1=0;
    R2=domain_structure.radius;
    C=domain_structure.center;
    thetaV=domain_structure.angles;

    domain_structure.A=[R1 0; R2 0];
    domain_structure.B=[0 R1; 0 R2];
    domain_structure.C=[C; C];
    domain_structure.alpha=thetaV(1);
    domain_structure.beta=thetaV(2);



case 'asymmetric-circular-sector'

    % ......................... setting defaults .......................
    %
    % domain_structure.centers:
    % 0. it is a matrix 2 x 2;
    % 1. first row consists of the coordinates of the first center of
    %    the 'asymmetric-circular-sector';
    % 2. second row consists of the coordinates of the second center of
    %    the 'asymmetric-circular-sector';
    %
    % domain_structure.radius:
    % 0. it is a vector of length 1;
    % 1. it consists of the radius of the disk;
    %
    % domain_structure.angles:
    % 0. it is a vector of length 2;
    % 1. the first component is the parameter "alpha" and must be in
    %    [-pi,pi];
```

```
    % 2. the second component is the parameter "beta" and must be in
    %    [-pi,pi];

    % default example:
    domain_structure.centers=[1 1; 2 2];
    domain_structure.radius=3;
    domain_structure.angles=[pi/4; pi/2];

    while k <= L
        strL=args{k};
        switch strL
            case 'centers'
                domain_structure.centers=args{k+1};
            case 'radius'
                domain_structure.radius=args{k+1};
            case 'angles'
                domain_structure.angles=args{k+1};
        end
        k=k+2;
    end

    domain_structure.NURBS=ellblend2NURBS(domain_structure);


    % ellblend structure

    R=domain_structure.radius;
    C=domain_structure.centers;
    thetaV=domain_structure.angles;

    domain_structure.A=[R 0; 0 0];
    domain_structure.B=[0 R; 0 0];
    domain_structure.C=C;
    domain_structure.alpha=thetaV(1);
    domain_structure.beta=thetaV(2);



case 'asymmetric-annulus'

    % ....................... setting defaults ......................
    %
    % IMPORTANT: the 'asymmetric-annulus' is determined by a first disk
    % "D1" and a second disk "D2". It is required that "D2" is included
    % in "D1".
    %
    % domain_structure.centers:
    % 0. it is a matrix 2 x 2;
    % 1. first row consists of the coordinates of the first center of
    %    the 'asymmetric-circular-sector';
    % 2. second row consists of the coordinates of the second center of
    %    the 'asymmetric-circular-sector';
    %
    % domain_structure.radii:
    % 0. it is a vector of length 2;
    % 1. first value consists of the radius of the first disk;
    % 2. second value consists of the radius of the second disk;
    %
```

```
    % default example:
    domain_structure.centers=[0.5 0.5; 0.7 0.7];
    domain_structure.radii=[0.5 0.15];

    while k <= L
        strL=args{k};
        switch strL
            case 'centers'
                domain_structure.centers=args{k+1};
            case 'radii'
                domain_structure.radii=args{k+1};
        end
        k=k+2;
    end


    % ellblend structure

    R=domain_structure.radii;
    R1=R(1); R2=R(2);
    C=domain_structure.centers;

    domain_structure.A=[R1 0; R2 0];
    domain_structure.B=[0 R1; 0 R2];
    domain_structure.C=C;
    domain_structure.alpha=-pi;
    domain_structure.beta=pi;



case 'vertical-circular-zone'

    % ..................... setting defaults ......................
    %
    % domain_structure.center:
    % 0. it is a matrix 1 x 2;
    % 1. it consists of the coordinates of center of the
    %    'vertical-circular-zone';

    % domain_structure.radius:
    % 0. it is a vector of length 1;
    % 1. it consists of the radius of the disk;

    % domain_structure.angles:
    % 0. it is a vector of length 2;
    % 1. the first component is the parameter "alpha" and must be in
    %    [0,pi];
    % 2. the second component is the parameter "beta" and must be in
    %    [0,pi]; it is required that "alpha < beta";

    domain_structure.center=[1 1];
    domain_structure.radius=1;
    domain_structure.angles=[pi/6 pi/2+pi/6];

    while k <= L
        strL=args{k};
```

```
        switch strL
            case 'center'
                domain_structure.center=args{k+1};
            case 'radius'
                domain_structure.radius=args{k+1};
            case 'angles'
                domain_structure.angles=args{k+1};
        end
        k=k+2;
    end

    domain_structure.NURBS=ellblend2NURBS(domain_structure);

    % ellblend structure

    R=domain_structure.radius;
    C=domain_structure.center;
    thetaV=domain_structure.angles;

    domain_structure.A=[R 0; R 0];
    domain_structure.B=[0 R; 0 -R];
    domain_structure.C=[C; C];
    domain_structure.alpha=thetaV(1);
    domain_structure.beta=thetaV(2);



case 'horizontal-circular-zone'
    % ..................... setting defaults ......................
    %
    % domain_structure.center:
    % 0. it is a matrix 1 x 2;
    % 1. it consists of the coordinates of center of the
    %    'vertical-circular-zone';
    %
    % domain_structure.radius:
    % 0. it is a vector of length 1;
    % 1. it consists of the radius of the disk;
    %
    % domain_structure.angles:
    % 0. it is a vector of length 2;
    % 1. the first component is the parameter "alpha" and must be in
    %    [-pi/2,pi/2];
    % 2. the second component is the parameter "beta" and must be in
    %    [-pi/2,pi/2]; it is required that "alpha < beta";

    domain_structure.center=[1 1];
    domain_structure.radius=1;
    domain_structure.angles=[-pi/4 pi/6];

    while k <= L
        strL=args{k};
        switch strL
            case 'center'
                domain_structure.center=args{k+1};
            case 'radius'
                domain_structure.radius=args{k+1};
            case 'angles'
```

```
                domain_structure.angles=args{k+1};
            end
            k=k+2;
        end

        domain_structure.NURBS=ellblend2NURBS(domain_structure);

        % ellblend structure

        R=domain_structure.radius;
        C=domain_structure.center;
        thetaV=domain_structure.angles;

        domain_structure.A=[R 0; -R 0];
        domain_structure.B=[0 R; 0 R];
        domain_structure.C=[C; C];
        domain_structure.alpha=thetaV(1);
        domain_structure.beta=thetaV(2);



    case 'circular-segment'

        % ....................... setting defaults ......................
        %
        % domain_structure.center:
        % 0. it is a matrix 1 x 2;
        % 1. it consists of the coordinates of center of the
        %    'circular-segment';
        %
        % domain_structure.radius:
        % 0. it is a vector of length 1;
        % 1. it consists of the radius of the disk;
        %
        % domain_structure.angles:
        % 0. it is a vector of length 2;
        % 1. the first component is the parameter "alpha" and must be in
        %    [-pi,pi];
        % 2. the second component is the parameter "beta" and must be in
        %    [-pi,pi];

        domain_structure.center=[1 1];
        domain_structure.radius=2;
        domain_structure.angles=[-pi/4 pi/6];

        while k <= L
            strL=args{k};
            switch strL
                case 'center'
                    domain_structure.center=args{k+1};
                case 'radius'
                    domain_structure.radius=args{k+1};
                case 'angles'
                    domain_structure.angles=args{k+1};
            end
            k=k+2;
        end
```

```
    % NURBS structure

    domain_structure.NURBS=ellblend2NURBS(domain_structure);

    % ellblend structure

    R=domain_structure.radius;
    C=domain_structure.center;
    thetaV=domain_structure.angles;
    alpha=thetaV(1); beta=thetaV(2); gamma=(thetaV(1)+thetaV(2))/2;

    domain_structure.A=[R*cos(alpha) R*sin(alpha); R*cos(beta) R*sin(beta)];
    domain_structure.B=[-R*sin(alpha) R*cos(alpha); R*sin(beta) -R*cos(beta)
        ];

    domain_structure.C=[C; C];
    domain_structure.alpha=0;
    domain_structure.beta=abs(beta-alpha)/2;



case 'symmetric-lens'

    % ..................... setting defaults ......................
    %
    % domain_structure.center:
    % 0. it is a matrix 1 x 1;
    % 1. it consists of the x0 coordinates of center of the
    %    'symmetric-lens', i.e. the lens will be defined by the disks
    %    having centers [+/- domain_structure.center,0];
    %
    % domain_structure.radius:
    % 0. it is a vector of length 1;
    % 1. it consists of the radius of each disk of the lens;
    %

    % setting defaults
    domain_structure.center=1;
    domain_structure.radius=2.5;

    while k <= L
        strL=args{k};
        switch strL
            case 'center'
                domain_structure.center=args{k+1};
            case 'radius'
                domain_structure.radius=args{k+1};
        end
        k=k+2;
    end


    % ellblend structure

    R=domain_structure.radius;
    a=domain_structure.center;

    domain_structure.A=[R 0; -R 0];
```

```
    domain_structure.B=[0 R; 0 R];
    domain_structure.C=[-a 0; a 0];

    domain_structure.alpha=-acos(a/R);
    domain_structure.beta=acos(a/R);


case 'butterfly'

    % ..... setting defaults .....
    %
    % domain_structure.center:
    % 0. it is a matrix 1 x 2;
    % 1. it consists of the coordinates of center of the
    %    'butterfly';
    %
    % domain_structure.radius:
    % 0. it is a vector of length 1;
    % 1. it consists of the radius of the disk of the
    %    butterfly;
    %
    % domain_structure.angles:
    % 0. it is a vector of length 2;
    % 1. the first component is the parameter "alpha" and must be in
    %    [-pi,pi];
    % 2. the second component is the parameter "beta" and must be in
    %    [-pi,pi];

    % setting defaults
    domain_structure.center=[1 2];
    domain_structure.radius=1;
    domain_structure.angles=[-pi/4 pi/6];


    while k <= L
        strL=args{k};
        switch strL
            case 'center'
                domain_structure.center=args{k+1};
            case 'radius'
                domain_structure.radius=args{k+1};
            case 'angles'
                domain_structure.angles=args{k+1};
        end
        k=k+2;
    end

    domain_structure.NURBS=ellblend2NURBS(domain_structure);

    % ellblend structure

    R=domain_structure.radius;
    C=domain_structure.center;
    thetaV=domain_structure.angles;

    domain_structure.A=[R 0; -R 0];
    domain_structure.B=[0 R; 0 -R];
```

```
    domain_structure.C=[C; C];
    domain_structure.alpha=thetaV(1);
    domain_structure.beta=thetaV(2);


case 'candy'

    % ..... setting defaults .....
    %
    % domain_structure.center:
    % 0. it is a matrix 1 x 1;
    % 1. it determines the value of the parameter "a"  defining
    %    the candy;
    %
    % domain_structure.radius:
    % 0. it is a vector of length 1;
    % 1. it consists of the radius "r" defining the candy;
    %
    % domain_structure.angle:
    % 0. it is a vector of length 1;
    % 1. it is the parameter "alpha" defining the candy and must be in
    %    [-pi,pi];
    %
    % IMPORTANT: it must be "-alpha > acos(a/r)"


    % setting defaults
    domain_structure.center=0.5;
    domain_structure.radius=1;
    domain_structure.angle=-pi/2;


    while k <= L
        strL=args{k};
        switch strL
            case 'center'
                domain_structure.center=args{k+1};
            case 'radius'
                domain_structure.radius=args{k+1};
            case 'angle'
                domain_structure.angle=args{k+1};
        end
        k=k+2;
    end


    % ellblend structure

    r=domain_structure.radius;
    a=domain_structure.center;
    alpha=domain_structure.angle;

    domain_structure.A=[r 0; -r 0];
    domain_structure.B=[0 r;0 r];
    domain_structure.C=[-a 0; a 0];
    domain_structure.alpha=alpha;
    domain_structure.beta=-alpha;
```

```
case 'NURBS'
    % ..... setting defaults .....

    %
    % domain_structure.geometry:
    % 0. it is a "Matlab-structure" defining the geometry of
    %    the NURBS;
    % 1. for examples on how to define this "structure"
    %    see the gallery "gallery_NURBS".

    % setting default from gallery of domains

    example_type=42;
    domain_structure=gallery_domains_2D(domain_str,example_type);

    while k <= L
        strL=args{k};
        switch strL
            case 'geometry'
                domain_structure.NURBS=args{k+1};
        end
        k=k+2;
    end



case 'union-disks'

    % ..... setting defaults .....
    %
    % domain_structure.centers:
    % 0. it is a matrix M x 2;
    % 1. the k-th row consist of the coordinates of the center
    %    of the k-th disk defining the domain;
    %
    % domain_structure.radii:
    % 0. it is a vector of length M;
    % 1. the k-th component consist of the radius of the k-th
    %     disk defining the domain;


    % setting defaults
    example_type=1;
    domain_structure=gallery_domains_2D(domain_str,example_type);

    while k <= L
        strL=args{k};
        switch strL
            case 'centers'
                domain_structure.centers=args{k+1};
            case 'radii'
                domain_structure.radii=args{k+1};
        end
        k=k+2;
    end
```

```
case 'square'

    % reference square: [-1,1] x [-1,1]

    vertices=[-1 -1; 1 -1; 1 1; -1 1; -1 -1];
    domain_structure.vertices=vertices;

    XV=domain_structure.vertices(:,1);
    YV=domain_structure.vertices(:,2);

    domain_structure.polyshape=polyshape(XV,YV);



case 'unit-square[0,1]x[0,1]'

    % reference square: [0,1] x [0,1]

    vertices=[0 0; 1 0; 1 1; 0 1; 0 0];
    domain_structure.vertices=vertices;

    XV=domain_structure.vertices(:,1);
    YV=domain_structure.vertices(:,2);

    domain_structure.polyshape=polyshape(XV,YV);



case 'rectangle'

    % 0. defining the domain as dbox=[xLimits; yLimits], where
    %    xLimits=[xmin, xmax], yLimits=[ymin, ymax]
    %
    % 1. the routine will also assign the vertices of the domain.

    % setting defaults
    domain_structure.xLimits=[0.5 1];
    domain_structure.yLimits=[2 3];

    while k <= L
        strL=args{k};
        switch strL
            case 'xLimits'
                domain_structure.xLimits=args{k+1};
            case 'yLimits'
                domain_structure.yLimits=args{k+1};
        end
        k=k+2;
    end

    xmin=min(domain_structure.xLimits);
    xmax=max(domain_structure.xLimits);
    ymin=min(domain_structure.yLimits);
    ymax=max(domain_structure.yLimits);

    vertices=[xmin ymin; xmax ymin; xmax ymax; xmin ymax; ...
```

```
        xmin ymin];

    domain_structure.vertices=vertices;

    XV=domain_structure.vertices(:,1);
    YV=domain_structure.vertices(:,2);

    domain_structure.polyshape=polyshape(XV,YV);


case 'triangle'

    % ..... setting defaults .....
    %
    % domain_structure.vertices:
    % 0. it is a matrix 3 x 2 (no requirement that the first and last
    %     vertex coincide;
    % 1. each row consists of the coordinates of a vertex of the domain
    %    in cartesian coordinates;

    % setting defaults
    vertices=[0 0; 1 0; 1 1; 0 0];

    while k <= L
        strL=args{k};
        switch strL
            case 'vertices'
                vertices=args{k+1};
        end
        k=k+2;
    end

    if norm(vertices(1,:)-vertices(end,:)) > 0
        vertices(end+1,:)=vertices(1,:);
    end

    domain_structure.vertices=vertices;

    XV=domain_structure.vertices(:,1);
    YV=domain_structure.vertices(:,2);

    domain_structure.polyshape=polyshape(XV,YV);


case 'polygcirc'

    % ..... setting defaults .....
    %
    % domain_structure.extrema:
    % 0. it is a matrix 2 x 2;
    % 1. each row consists of the coordinates of a vertex of the domain
    %    in cartesian coordinates;
    % 2. the first row is the first extremum (point from which it
    %    starts the arc of the disk);
    % 3. the second row is the second extremum (point from which it
    %    end the arc of the disk);
```

```
    %
    % domain_structure.vertices:
    % 0: it is a matrix N x 2 of polygon vertices (cart. coordinates);
    % 1: arc extrema are added to "domain_structure.vertices", by
    %    default in counterclockwise order;
    %
    % domain_structure.center:
    % 0. it is a matrix 1 x 2;
    % 1. it determines the center of the disk defining "polygcirc"
    %    domain;
    %
    % domain_structure.radius:
    % 0. it is a vector of length 1;
    % 1. it consists of the radius "r" of the disk defining "polygcirc"
    %    domain;
    %
    % domain_structure.convex:
    % 0: it is a parameter that if "0" will define a concave arc,
    %     otherwise a convex arc.

    % default example
    example_type=2;
    domain_structure=gallery_domains_2D(domain_str,example_type);

    while k <= L
        strL=args{k};
        switch strL
            case 'extrema'
                domain_structure.extrema=args{k+1};
            case 'vertices'
                domain_structure.vertices=args{k+1};
            case 'center'
                domain_structure.center=args{k+1};
            case 'radius'
                domain_structure.radius=args{k+1};
            case 'convex'
                domain_structure.convex=args{k+1};
        end
        k=k+2;
    end


case 'unit-simplex'

    domain_structure.vertices=[0 0; 1 0; 0 1; 0 0];

    XV=domain_structure.vertices(:,1);
    YV=domain_structure.vertices(:,2);

    domain_structure.polyshape=polyshape(XV,YV);

case 'sphere'

    domain_structure=gallery_domains_2D(domain_str);
    %XW=cub_sphere(deg);

 case 'spherical-polygon'
```

```
        example_type=2;
        domain_structure=gallery_domains_S2(domain_str,example_type);

        sphpgon_vertices=domain_structure.vertices;
        %XW=cub_sphpgon(deg,sphpgon_vertices);



end

domain_structure.dbox=domain_boundingbox(domain_structure);


function domain_structure_dbox=domain_boundingbox(domain_structure)

% domain_structure is a structure containing at least:
% 1. domain_structure.domain: string with domain name
% 2. domain_structure.parms : domain structure (variable from domain).

domain_structure_dbox=[];
domain_string=domain_structure.domain;


switch domain_string

    case {'polygon','square','rectangle','triangle',...
            'unit-square[0,1]x[0,1]'}

        [xlimit,ylimit]=boundingbox(domain_structure.polyshape);
        domain_structure_dbox=[xlimit; ylimit];


    case {'lune','circular-annular-sector','sector',...
            'vertical-circular-zone',...
            'horizontal-circular-zone','circular-segment','butterfly'}

        domain_structure_dbox=compute_bbox_NURBS(domain_structure);

    case 'disk'

        % ..................... decode variables .......................
        center=domain_structure.center;
        r=domain_structure.radius;

        % ................... determine bounding box ....................
        xlimit=[center(1)-r center(1)+ r];
        ylimit=[center(2)-r center(2)+ r];

        domain_structure_dbox=[xlimit; ylimit];

    case 'asymmetric-circular-sector'

        % ..................... decode variables .......................

        structure_RS=domain_structure.NURBS;
        [xyw, ~, ~, ~,bbox] = cub_NURBS(0,structure_RS);
```

```
    % ................... determine bounding box ......................

    xLimit=[bbox(1), bbox(2)]; yLimit=[bbox(3), bbox(4)];
    domain_structure_dbox=[xLimit; yLimit];

case 'asymmetric-annulus'

    % ..................... decode variables .........................

    centerV=domain_structure.centers;
    rV=domain_structure.radii;

    a=centerV(1,1); b=centerV(1,2); c=centerV(2,1); d=centerV(2,2);
    r1=rV(1); r2=rV(2);

    % ................... determine bounding box ......................

    xLimit=[min(a-r1,c-r2) max(a+r1,c+r2)];
    yLimit=[min(b-r1,d-r2) max(b+r1,d+r2)];
    domain_structure_dbox=[xLimit; yLimit];


case 'symmetric-lens'

    % ....................... decode variables .......................
    a=domain_structure.center; a=abs(a);
    r=domain_structure.radius;

    center1=[-a 0]; center2=[a 0];

    % ................... determine bounding box ......................
    xlimit=[a-r -a+ r];
    c=sqrt(r^2-a^2); ylimit=[-c c];

    domain_structure_dbox=[xlimit; ylimit];



case 'candy'

    % ....................... decode variables .......................

    a=domain_structure.center;
    r=domain_structure.radius;
    alpha=domain_structure.angle;

    % ................... determine bounding box ......................

    domain_structure_dbox=[];

case 'NURBS'

    % ..................... decode variables .........................

    structure_RS=domain_structure.NURBS;
    [xyw, ~, ~, ~,bbox] = cub_NURBS(0,structure_RS);
```

```
    % ................... determine bounding box .....................

    xLimit=[bbox(1), bbox(2)]; yLimit=[bbox(3), bbox(4)];
    domain_structure_dbox=[xLimit; yLimit];


case 'union-disks'

    % ..................... decode variables ........................

    centers=domain_structure.centers;
    rs=domain_structure.radii;

    % ................... determine bounding box .....................
    Xc=centers(:,1); Yc=centers(:,2);

    xLimits=[min(Xc-rs) max(Xc+rs)];
    yLimits=[min(Yc-rs) max(Yc+rs)];
    domain_structure_dbox=[xLimits; yLimits];

case 'polygcirc'

    % ..................... decode variables ........................

    a=domain_structure.extrema(1,:);
    b=domain_structure.extrema(2,:);
    v=domain_structure.vertices;
    cc=domain_structure.center;
    r=domain_structure.radius;
    conv=domain_structure.convex;

    % ................... determine bounding box .....................

    vertices=[a; v; b];
    X=[a(1); v(:,1); b(1)]; Y=[a(2); v(:,2); b(2)];

    if conv
        X=[X; cc(1)+r; cc(1)-r]; Y=[Y; cc(2)+r; cc(2)-r];
    end

    xLimits=[min(X) max(X)]; yLimits=[min(Y) max(Y)];
    domain_structure_dbox=[xLimits; yLimits];

case 'unit-simplex'

    % ................... determine bounding box .....................

    xLimits=[0 1]; yLimits=[0 1];
    domain_structure_dbox=[xLimits; yLimits];

case 'sphere'

    % ................... determine bounding box .....................

    xLimits=[-1 1]; yLimits=[-1 1]; zLimits=[-1 1];
    domain_structure_dbox=[xLimits; yLimits; zLimits];

   case 'spherical-polygon'
```

```
        % ..................  determine bounding box ......................
        XYZV=domain_struct.vertices;
        xLimits=[min(XYZV(:,1)) max(XYZV(:,1))];
        yLimits=[min(XYZV(:,2)) max(XYZV(:,2))];
        zLimits=[min(XYZV(:,3)) max(XYZV(:,3))];
        domain_structure_dbox=[xLimits; yLimits; zLimits];

end
```

```
function domain_structure_dbox=compute_bbox_NURBS(domain_structure)

[~,~,~,~,~,~,boxVx]=indomain_NURBS([],domain_structure.NURBS);

% rectangle is [bbox(1), bbox(2)] x [bbox(3), bbox(4)];
xLimit=[min(boxVx(:,1)), max(boxVx(:,2))];
yLimit=[min(boxVx(:,3)), max(boxVx(:,4))];

domain_structure_dbox=[xLimit; yLimit];


function [t,dbox,area_domain]=define_scattered_pointset(card,...
    domain_struct,scat_type,area_domain)

if nargin < 1, card=800; end
if nargin < 2,  domain_struct=define_domain('polygon'); end
if nargin < 3, scat_type='halton'; end
if nargin < 4
    xyw=define_cub_rule(domain_struct,1,0); area_domain=sum(xyw(:,3));
end


switch scat_type
    case 'halton'
        P = haltonset(2);
    otherwise
        P = sobolset(2);
end

% bounding box
dbox=domain_struct.dbox;
xLimit=dbox(1,:);
yLimit=dbox(2,:);
area_boundingbox=diff(xLimit)*diff(yLimit);

% ratio: area(bounding box)/area(domain)
ratio=area_boundingbox/area_domain;

% computing sufficiently large pointset on [0,1] x [0,1]
card_bbox=ceil(4*ratio*card);
t0_ref = net(P,card_bbox);
```

```
% mapping sufficiently large pointset on domain bounding box
t0=[xLimit(1)+(t0_ref(:,1))*diff(xLimit) ...
    yLimit(1)+(t0_ref(:,2))*diff(yLimit)];

% extracting points on domain
in0=indomain_routine(domain_struct,t0);
in=find(in0 == 1);

% problem with cardinality: warning of failure
if length(in) < card
    fprintf('\n \t Number of points provided: %6.0f, asked for: %6.0f',...
       length(in),card )
end

t=t0(in(1:card),:);


function demo_glaubitz_2D

%-------------------------------------------------------------------------
% OBJECT
%-------------------------------------------------------------------------
% Demo used to compute integrals from samples involving scattered data.
%
% 1. Determines domain and functions to study from function defined by the
%    user.
% 2. computes integral using only evaluations at scattered data:
%
% (a) For methods based on evaluating in scattered data, using the values
%    to evaluate at points of an algebraic rule:
%    Makes experiments, evaluating the integrand at cubature nodes of an
%    algebraic rule, when only samples at scattered data are available.
%
%    Important: algebraic degree of exactness of the cubature rule is fixed.
%
% (b) By Glaubitz algorithm computes the value of the integral, with a
%    formula based on scattered data.
%
% 3. Makes plot of the domain, scattered samples, cubature nodes.
%-------------------------------------------------------------------------
% Modified: November 6, 2024.
%-------------------------------------------------------------------------
% COPYRIGHT
%-------------------------------------------------------------------------
% Copyright (C) 2024-
%
% Authors:
% Alvise Sommariva.
%
% 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 3 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, see <https://www.gnu.org/licenses/>.
%--------------------------------------------------------------------------



% ..... Problem settings  .....................

%--------------------------------------------------------------------------
% 1:'polygon'; 2:'disk'; 3:'lune'; 4:'circular-annular-sector';
% 5:'sector'; 6:'asymmetric-circular-sector'; 7:'asymmetric-annulus';
% 8:'vertical-circular-zone'; 9:'horizontal-circular-zone';
% 10:'circular-segment'; 11:'symmetric-lens'; 12:'butterfly';
% 13:'candy'; 14:'NURBS'; 15:'union-disks';16:'asymmetric-circular-sector';
% 17:'square'; 18:'unit-square[0,1]x[0,1]'; 19:'triangle'; 20:'polygcirc';
% 21, 'unit-simplex';
%--------------------------------------------------------------------------
% Note: use deg_rule <=15 for NURBS
%--------------------------------------------------------------------------
tic
domain_type=11;

deg_rule=40; %mettere 20/30 % degree of precision of the rule; choose deg=10 for
     NURBS.
% grado formula di riferimento (basata su gqellblend)
card=100;    % scattered data cardinality
scat_type='halton'; % scattered data type.

f=@(x,y) 1./( (1+x.^2).*(1+y.^2));
%f =@(x,y) exp(-x.^2-y.^2);

method='Glaubitz-algorithm';


%nella figura punti rossi ref, neri glaubitz

% ..... Preliminary settings  ..........................

% some additional routines are required
addpath(genpath('../EXTERNAL_ROUTINES/'));

% flag_compression: if "1" compression will be attempted, if necessary,
% otherwise the code will propose an alternative rule.
flag_compression=1;



% ..... Making numerical tests  ........................


% 1. Define domain
%    Note: the domain routine is in "../EXTERNAL_ROUTINES/DCUB" folder

fprintf('\n \t * defining integration domain');
domain_example=domain_str(domain_type);
domain_struct=define_domain(domain_example);
fprintf('\n \n \t \t The domain is: '); disp(domain_example);
```

```
% 1A. Define scattered data
fprintf('\n \t * defining scattered data');
[t,dbox,area_domain]=define_scattered_pointset(card,domain_struct,...
    scat_type);
ft=feval(f,t(:,1),t(:,2));

% 1B. Define cubature rule
%    Note: the domain routine is in "../EXTERNAL_ROUTINES/DCUB" folder
XYW=define_cub_rule(domain_struct,deg_rule,flag_compression);

% 1C: Compute integral from evaluation at scattered data


area_dbox=diff(dbox(1,:))*diff(dbox(2,:));
wQMC=(area_domain/area_dbox)/size(t,1);
[w,deg_rule]=glaubitz_algorithm(t,domain_struct,wQMC,'l1'); % chage approach to
    change formula
I=w'*ft;


%--------------------------------------------------------------------------
% 2. Display statistics.
%--------------------------------------------------------------------------
fprintf('\n \t ------------------------------------------------------');
fprintf('\n \t Domain (integration)          : '); disp(domain_struct.domain);
fprintf('\n \t Scattered data type           : '); disp(scat_type);
fprintf(   '\t Evaluation method             : '); disp(method);
fprintf('\n \t Scatt. points card. required  : %5.0f',card);
fprintf('\n \t Scatt. points card. provided  : %5.0f \n',size(t,1));

fprintf('\n \t Degree of precision GL          : %5.0f',deg_rule);
% grado di precisione formula di Glaubitz
% caso bivariato sarebbero grado 6 --> 28 punti
card_rule=size(XYW,1); % cardinality della formula di riferimento
wp = find(w>0);  %cardinality della formula di Glaubitz
fprintf('\n \t Cardinality of the formula Ref   : %5.0f',card_rule);
fprintf('\n \t Cardinality of the formula GL    : %5.0f',length(wp));
dim_poly=(deg_rule+1)*(deg_rule+2)/2;
fprintf('\n \t Dimension of the poly. space  : %5.0f',dim_poly);


fprintf('\n \t Integral approximation        : %1.15e',I);

switch domain_type
    case 17
        atol=10^(-12); rtol=10^(-12);
        IR = integral2(f,-1,1,-1,1,'AbsTol',atol,'RelTol',rtol);
    case 18
        atol=10^(-12); rtol=10^(-12);
        IR = integral2(f,0,1,0,1,'AbsTol',atol,'RelTol',rtol);
    otherwise
        ft=feval(f,XYW(:,1),XYW(:,2));
        IR=(XYW(:,3))'*ft;
end

fprintf('\n \t Integral computed by alg.rule : %1.15e',IR);
fprintf('\n \t Absolute error                : %1.1e',abs(IR-I));
fprintf('\n \t Relative error                : %1.1e',abs(IR-I)/abs(I));
```

```
fprintf('\n \n');

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

fstr = strrep(char(f),'@(x)','');
fprintf('\n \n \t The integrand is: '); disp(fstr);
fprintf('\n \t The domain is   : '); disp(domain_struct.domain);
fprintf('\n \n \t ..................................................... ');
fprintf('\n \n');



% 3. plot domain and scattered data
figure(1)
plot_2D(domain_struct,t,XYW(:,1:2));
saveas(gcf,domain_example,'epsc');



toc



function domain_example=domain_str(example)

%--------------------------------------------------------------------------
% OBJECT
%--------------------------------------------------------------------------
% Given a number "example", it provides the name of the corrsponding domain
% in the gallery,
%--------------------------------------------------------------------------

switch example

    case 1, domain_example='polygon';
    case 2, domain_example='disk';
    case 3, domain_example='lune';
    case 4, domain_example='circular-annular-sector';
    case 5, domain_example='sector';
    case 6, domain_example='asymmetric-circular-sector';
    case 7, domain_example='asymmetric-annulus';
    case 8, domain_example='vertical-circular-zone';
    case 9, domain_example='horizontal-circular-zone';
    case 10, domain_example='circular-segment';
    case 11, domain_example='symmetric-lens';
    case 12, domain_example='butterfly';
    case 13, domain_example='candy';
    case 14, domain_example='NURBS';
    case 15, domain_example='union-disks';
    case 16, domain_example='asymmetric-circular-sector';
    case 17, domain_example='square';
    case 18, domain_example='unit-square[0,1]x[0,1]';
    case 19, domain_example='triangle';
    case 20, domain_example='polygcirc';
    case 21, domain_example='unit-simplex';
```

```
end


function in=indomain_routine(domain_structure,pts)

% 1: 'polygon';
% 2: 'disk';
% 3: 'lune';
% 4: 'circular-annular-sector';
% 5: 'sector';
% 6: 'asymmetric-circular-sector';
% 7: 'asymmetric-annulus';
% 8: 'vertical-circular-zone';
% 9: 'horizontal-circular-zone';
% 10: 'circular-segment';
% 11: 'symmetric-lens';
% 12: 'butterfly';              % NOT PASSED!
% 13: 'candy';                  % NOT PASSED!
% 14: 'NURBS';
% 15: 'union-disks';            % 10^(-11)
% 16: 'asymmetric-circular-sector';

domain_str=domain_structure.domain;

switch domain_str

    case {'polygon','unit-simplex','square','rectangle','triangle', ...
            'unit-square[0,1]x[0,1]'}

        % ..................... indomain code ............................

        P=domain_structure.polyshape;
        X=pts(:,1); Y=pts(:,2);
        in=isinterior(P,X,Y);

    case 'disk'

        % ..................... decode variables .........................

        center=domain_structure.center;
        r=domain_structure.radius;

        % ..................... indomain code ............................

        dist_C_pts=sqrt((pts(:,1)-center(1)).^2+...
            (pts(:,2)-center(2)).^2);
        in=(dist_C_pts <= r);

    case 'lune'

        % ..................... decode variables .........................

        center1=domain_structure.centers(1,:);
        center2=domain_structure.centers(2,:);
        r1=domain_structure.radii(1);
        r2=domain_structure.radii(2);

        dist1=sqrt((pts(:,1)-center1(1)).^2+...
            (pts(:,2)-center1(2)).^2);
```

```
    dist2=sqrt((pts(:,1)-center2(1)).^2+...
        (pts(:,2)-center2(2)).^2);

    in=(dist1 <= r1) & (dist2 > r2);

case 'circular-annular-sector'

    % ..................... decode variables .......................

    center=domain_structure.center;
    r1=domain_structure.radii(1);
    r2=domain_structure.radii(2);
    alpha=domain_structure.angles(1);
    beta=domain_structure.angles(2);

    % ..................... indomain code ..........................

    Xref=pts(:,1)-center(1); Yref=pts(:,2)-center(2);
    [TH,R] = cart2pol(Xref,Yref);

    inR=(R <= max(r1,r2)) & (R >= min(r1,r2));

    if not((abs(alpha) <= pi)  & (abs(beta) <= pi))
        error('alpha >= beta or at least one not in [-pi,pi]');
    end

    if (alpha < beta)
        % regular situation
        in_TH=(TH >= alpha) & (TH <= beta);
    else
        in_TH=not((TH >= beta) & (TH <= alpha));
    end

    in=inR & in_TH;



case 'sector'
    %cooprdinate polari, guardo raggio tra 0 e 1, guardo angolo
    %settore, se sono tutti nell'intervallo giusto allora il punto e'
    %dentro.
    % prendo bounding box(quadratino che contiene il dominio), metto punti
       nella bounding box, voglio 200
    % punti random. Quindi nel dominio grande calcolo quanto il bb e'
    % piu grande del settore(audio)
    % ..................... decode variables .......................

    center=domain_structure.center;
    r=domain_structure.radius;
    alpha=domain_structure.angles(1);
    beta=domain_structure.angles(2);

    % ..................... indomain code ..........................

    Xref=pts(:,1)-center(1); Yref=pts(:,2)-center(2);
    [TH,R] = cart2pol(Xref,Yref);
```

```
    inR=(R <=r);

    if not((abs(alpha) <= pi)  & (abs(beta) <= pi))
        error('alpha >= beta or at least one not in [-pi,pi]');
    end

    if (alpha < beta)
        % regular situation
        in_TH=(TH >= alpha) & (TH <= beta);
    else
        in_TH=not((TH >= beta) & (TH <= alpha));
    end

    in=(R <=r) & in_TH;



case  'asymmetric-annulus'

    % ..................... decode variables .......................

    centers=domain_structure.centers;
    rV=domain_structure.radii;

    % ..................... indomain code ..........................

    C=centers(1,:);
    V=centers(2,:);
    r1=rV(1); r2=rV(2);

    X=pts(:,1); Y=pts(:,2);
    inCr1=(vecnorm([X-C(:,1) Y-C(:,2)],2,2) <= r1);
    inCr2=(vecnorm([X-V(:,1) Y-V(:,2)],2,2) <= r2);

    in=inCr1 & not(inCr2);

case 'vertical-circular-zone'

    % ..................... decode variables .......................

    center=domain_structure.center;
    r=domain_structure.radius;
    alpha=domain_structure.angles(1);
    beta=domain_structure.angles(2);

    % ..................... indomain code ..........................

    xmin=r*cos(beta); xmax=r*cos(alpha);

    Xref=pts(:,1)-center(1); Yref=pts(:,2)-center(2);
    [TH,R] = cart2pol(Xref,Yref);

    in=(R <= r) & ( Xref >= xmin ) & ( Xref <= xmax );

case 'horizontal-circular-zone'

    % ..................... decode variables .......................
```

```
    center=domain_structure.center;
    r=domain_structure.radius;
    alpha=domain_structure.angles(1);
    beta=domain_structure.angles(2);

    % ..................... indomain code ..........................

    ymin=r*sin(alpha); ymax=r*sin(beta);

    Xref=pts(:,1)-center(1); Yref=pts(:,2)-center(2);
    [TH,R] = cart2pol(Xref,Yref);

    in=(R <= r) & ( Yref >= ymin ) & ( Yref <= ymax );



case 'symmetric-lens'

    % ..................... decode variables .......................

    a=domain_structure.center; a=abs(a);
    r=domain_structure.radius;

    center1=[-a 0];
    center2=[a 0];

    % ..................... indomain code ..........................

    dist1=sqrt((pts(:,1)-center1(1)).^2+(pts(:,2)-center1(2)).^2);
    dist2=sqrt((pts(:,1)-center2(1)).^2+(pts(:,2)-center2(2)).^2);

    in=(dist1 <= r) & (dist2 <= r);


case 'candy'

    % ..................... decode variables .......................

    a=domain_structure.center;
    r=domain_structure.radius;
    alpha=domain_structure.angle;

    % ..................... indomain code ..........................

    error('Indomain routine for candy is not implemented');

case {'NURBS','butterfly','asymmetric-circular-sector',...
        'circular-segment'}

    % ..................... indomain code ..........................

    structure_RS=domain_structure.NURBS;
    [in,on]=indomain_NURBS(pts,structure_RS);

case 'union-disks'

    % ..................... decode variables .......................
```

```
    centers=domain_structure.centers;
    rs=domain_structure.radii;

    % ..................... indomain code ..........................

    Xc=centers(:,1); Yc=centers(:,2);
    X=pts(:,1); Y=pts(:,2);
    in=zeros(length(X),1);

    for k=1:length(rs)

        XLc=Xc(k); YLc=Yc(k); rL=rs(k);

        iout=find(in == 0);
        XL=X(iout); YL=Y(iout);
        r = vecnorm([XL-XLc YL-YLc],2,2);

        in(iout)=(r <= rL);

    end


case 'polygcirc'

    % ..................... decode variables .......................

    ab=domain_structure.extrema;
    a=ab(1,:); b=ab(2,:);
    v=domain_structure.vertices;
    cc=domain_structure.center;
    r=domain_structure.radius;
    conv=domain_structure.convex;

    % ..................... indomain code ..........................

    vertices=[a; v; b];

    X=pts(:,1); Y=pts(:,2);
    inP=inpolygon(X,Y,vertices(:,1),vertices(:,2));

    inD=( (X-cc(1)).^2+(Y-cc(2)).^2 <= r^2);

    if conv
        [THa,Ra] = cart2pol(a(1)-cc(1),a(2)-cc(2));
        [THb,Rb] = cart2pol(b(1)-cc(1),b(2)-cc(2));

        [TH,R] = cart2pol(X-cc(1),Y-cc(2));

        if THa <= THb
            inD=(R <= r) & not(TH >= THa & TH <= THb);
        else
            inD=(R <= r) & (TH >= THb & TH <= THa);
        end

        in= (inD | inP);

    else
        in= (not(inD) & inP);
```

```
        end


    otherwise
        error('indomain_routine not implemented for this domain');

end


function [w,dmax]=glaubitz_algorithm(xS,domain_structure,wQMC,approach,...
    d_start)

%-------------------------------------------------------------------------
% INPUT
%-------------------------------------------------------------------------
% xS: scattered data
% domain_structure: structure of the domain; it must include:
%       domain.ratio=area_domain/area_bounding_box;
% wQMC: weights of a QMC cubature rule, having "t" as nodes;
% approach: 'LS' or 'l1';
% d_start : starting degree of analysis;
%
% The variables "approach" and "d_start" are not mandatory.
%-------------------------------------------------------------------------
% OUTPUT
%-------------------------------------------------------------------------
% w : weights of the cubature rule, having "t" as nodes;
% d : highest possible degree of exactness (ADE of (t,w))
%-------------------------------------------------------------------------
% SUBROUTINE USED
%-------------------------------------------------------------------------
% 1. compute_weights (attached)
% 2. define monomials (attached)
% 3. define_cub_rule (external, compute an alg. rule, with prescribed
%    degree of exactness, in several domains)
%-------------------------------------------------------------------------

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

if nargin < 4, approach='LS'; end
if isempty(approach), approach='LS'; end
if not(strcmp(approach,'LS') | strcmp(approach,'l1')), approach='LS'; end

if nargin < 5, d_start=0; end
if isempty(d_start), d_start=0; end


% ................. assign some variables .................

xS_N=size(xS,1);
xS_dim=size(xS,2);
R = diag(wQMC); % discrete weights matrix
w = zeros(xS_N,1); % cubature weights


% ................. set up initial values ................

d = d_start; % degree of exactness (DoE)
```

```
K = nchoosek(xS_dim + d, xS_dim); % number of basis elements
check_rank = K; % rank
w_min = min(w); % smallest cubature weight

w_history={};

% ....... loop in which d is increased until the CF becomes unstable ......

e = -1e-14; % tollerance to allow small rounding errors
while check_rank == K && w_min >= e

    d = d+1; % increase the DoE
    K = nchoosek(xS_dim + d, xS_dim); % number of basis elements

    basis=define_monomials(d,xS_dim);
    P = basis(xS); % Vandermonde matrix

    check_rank = rank(P); % rank of P

    % compute moments
    [XW,~,domain_structure]=define_cub_rule(domain_structure,d);

    PXW=basis(XW(:,1:2));
    m=PXW*XW(:,3);

    % check wheter the data points are d-unisolvent
    if check_rank == K

        % different approaches
        switch approach
            case 'LS'
                A = P*sqrt(R);
                v = lsqminnorm(A,m); % indirect computation by optim. tools
                w = sqrt(R)*v;
            case 'l1'
                options = optimoptions('linprog','Display','none');
                w = linprog(ones(xS_N,1), [], [], P, m, zeros(xS_N,1),...
                    [], options ); % l1 weights
        end
        %options = optimoptions('linprog','Display','none');
        %w = linprog(ones(xS_N,1), [], [], P, m, zeros(xS_N,1),...
                    %[], options ); % l1 weights
    end

    if sum(w)==sum(w) && sum(abs(w))~=Inf && ...
            length(w) > 0 % w does not contain NaNs or Infs and is nonempty
        w_min = min(w);
    else
        w_min = -1;
    end

    w_history{end+1}=w;

end

w=w_history{end-1};
dmax=d-1;
```

```
function basis=define_monomials(d,dim)

K = nchoosek(dim + d, dim); % binomial coefficient/ dimension
alpha1 = zeros(K,1); % vector of exponents for x
alpha2 = zeros(K,1); % vector of exponents for y
alpha3 = zeros(K,1); % vector of exponents for z
m = zeros(K,1); % moments

%% exponents and basis
if dim == 1
    alpha1 = (0:d)';
    basis = @(x) x'.^alpha1; % basis
elseif dim == 2
    k = 1;
    for k1=0:1:d
        for k2=0:1:d
            if k1+k2<=d
                alpha1(k) = k1;
                alpha2(k) = k2;
                k = k+1;
            end
        end
    end
    basis = @(x) x(:,1)'.^alpha1 .* x(:,2)'.^alpha2; % basis
elseif dim == 3
    k = 1;
    for k1=0:d
        for k2=0:d
            for k3=0:d
                if k1+k2+k3<=d
                        alpha1(k) = k1;
                    alpha2(k) = k2;
                    alpha3(k) = k3;
                    k = k+1;
                end
            end
        end
    end
    basis = @(x) x(:,1)'.^alpha1 .* x(:,2)'.^alpha2 .* x(:,3)'.^alpha3; % basis
else
    error('Desired dimension not yet implemented!')
end
```