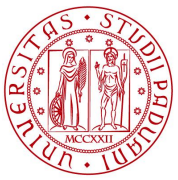


UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"

Corso di Laurea Triennale in Matematica

Tesi di Laurea

Una suite MATLAB per la compressione QMC su domini bivariati

Relatore:
Prof. Alvise Sommariva

Laureando: Mattia Santoro
Matricola: 1220508

Correlatore:
Dott. Giacomo Elefante

Anno Accademico 2023/2024

23/02/2024

Alla mia famiglia
che mi ha sostenuto e creduto in me,
più di quanto io avessi fatto.

A tutti coloro
che mi hanno accompagnato in questo mio percorso
regalandomi un pezzettino della loro anima.

Alle mie fragilità e alla mia insicurezza
che mi hanno fatto barcollare nel buio,
crollare
e subito dopo rialzare.

Indice

Introduzione

Il proposito di questa tesi è di approssimare integrali su domini dalla geometria particolarmente complessa, ma per cui non sia difficile stabilire se un punto appartenga o meno al dominio.

Il desiderio è quello di poter effettuare tale compito mediante poche e semplici linee di codice Matlab.

In regioni di questo tipo è pratica comune utilizzare metodi di tipo Quasi-Monte Carlo (che abbrevieremo con QMC), che sono di facile elaborazione ma che richiedono in generale la valutazione della integranda in una fitta nuvola di punti a bassa discrepanza.

In questi casi, è d'altra parte frequente che risulti complicata la determinazione della frontiera del dominio, rendendo di fatto problematica il calcolo di formule di cubatura algebriche.

Nella recente pubblicazione *CQMC: an improved code for low-dimensional Compressed Quasi-Monte Carlo cubature* gli autori hanno proposto un metodo per comprimere formule di cubatura di tipo Quasi-Monte Carlo, cosí da richiedere la valutazione della integranda in un sottoinsieme dei punti QMC, la cui cardinalitá è una piccola frazione di quella del metodo originario.

L'idea è di valutare i momenti di una opportuna base dello spazio dei polinomi di grado totale n e quindi risolvere il cosidetto *sistema sottodeterminato dei momenti*, fornendo una quale soluzione un vettore w avente componenti nonnegative ed in cui al più solo $(n+1)(n+2)/2$ sono nonnulle. Un ben noto teorema dovuto a Tchakaloff garantisce l'esistenza di almeno una soluzione di questo tipo.

Nella implementazione di CQMC si è cercato di fornire tale soluzione mediante l'applicazione anche di varianti efficienti del metodo di Lawson-Hanson.

Come risultato si ottiene una formula a bassa cardinalità e a pesi positivi su domini multivariati di geometrie complesse, che fornisce su polinomi di grado n lo stesso risultato di quella QMC avente però cardinalità molto maggiore.

L'intenzione, come anticipato, è quella di proporre i primi mattoni per costruire una shell semplice per utilizzare in Matlab il complesso algoritmo CQMC.

Si sono usate a tal proposito strutture di tipo *struct* e *classi*, per definire facilmente i domini di integrazione bivariati qualora questi siano ottenuti da operazioni insiemistiche su regioni più semplici quali dischi, ellissi, poligoni, per cui la valutazione della funzione indicatrice risulti avere un costo computazionale trascurabile.

Accanto a loro, le prime routines che determinano le *bounding-boxes* fondamentali per il calcolo di nodi a bassa discrepanza sul dominio di integrazione una volta che questi siano noti sul quadrato unitario $[0,1]^2$.

Nella sezione numerica, mostriamo come utilizzare tali routines tanto per la compressione QMC, quanto per calcolare numericamente integrali di funzioni bivariate.

Capitolo 1

Le formule Monte Carlo e quasi-Monte Carlo

1.1 Richiami

In questa sezione, verranno richiamate alcune definizioni di base e concetti fondamentali di Probabilità, essenziali nella comprensione e nella trattazione delle formule Monte Carlo.

Si ricordino le seguenti definizioni:

Definizione 1.1. *(σ-algebra) Sia Ω un insieme non vuoto e denotiamo con $\mathcal{P}(\Omega)$ l'insieme delle parti di Ω. Una famiglia $\mathcal{F} \subseteq \mathcal{P}(\Omega)$ si dice σ-algebra se:*

- $\emptyset \in \mathcal{F}$;
- *è chiusa rispetto alla complementazione, ossia $A \in \mathcal{F} \implies A^c \in \mathcal{F}$;*
- *è chiusa rispetto alle unioni numerabili, ossia $\{A_n\}_{n\in\mathbb{N}} \subset \mathcal{F} \implies \bigcup_{n\in\mathbb{N}} A_n \in \mathcal{F}$.*

Gli elementi di $\mathcal{F}$ si chiamano insiemi misurabili di Ω.

Definizione 1.2. *(Spazio misurabile) Uno spazio misurabile è una coppia $(\Omega, \mathcal{F})$, ove Ω è un insieme non vuoto e $\mathcal{F}$ è una σ-algebra su Ω.*

Definizione 1.3. *(Spazio di probabilità) Sia $(\Omega, \mathcal{F})$ uno spazio misurabile. Una funzione $\mathbf{P} : \mathcal{F} \to [0, 1]$ si chiama misura di probabilità se:*

1. $\mathbf{P}(\Omega) = 1$;
2. *data $\{A_n\}_{n\in\mathbb{N}} \subset \mathcal{F}$ successione numerabile di elementi disgiunti, si ha:*

$$\mathbf{P}\left(\bigcup_{n\in\mathbb{N}} A_n\right) = \sum_{n\in\mathbb{N}} \mathbf{P}(A_n).$$

In tal caso la terna $(\Omega, \mathcal{F}, \mathbf{P})$ è detta spazio di probabilità.

Definizione 1.4. *(Variabile aleatoria) Sia* $(\Omega, \mathcal{F}, \mathbf{P})$ *uno spazio di probabilità e sia* (E, ξ) *uno spazio misurabile. Una funzione* $X : \Omega \to E$ *si dice variabile aleatoria (v.a.) se per ogni* $c \in \xi$ *si ha* $X^{-1}(c) = \{\omega \in \Omega : X(\omega) = c\} \in \mathcal{F}$.

Ora, si consideri una successione di v.a. $(X_i)_{i\in\mathbb{N}}$ definite sullo stesso spazio di probabilità $(\Omega, \mathcal{F}, \mathbf{P})$. Definiamo la media campionaria (o media empirica) la seguente v.a.

$$\bar{X}_n = \frac{1}{n}\sum_{i=0}^{n} X_i \,, \quad n \geq 1.$$

Vale la seguente:

Definizione 1.5. *(Legge dei grandi numeri - LLN) Sia* $(X_i)_{i\in\mathbb{N}}$ *una successione di v.a. reali definite sullo stesso spazio di probabilità* $(\Omega, \mathcal{F}, \mathbf{P})$ *e supponiamo che abbiano lo stesso valor medio, cioè* $\mathbb{E}(X_i) = \mu$ *per ogni* $i \in \mathbb{N}$, $\mu \in \mathbb{R}$. *Diremo che* $(X_i)_{i\in\mathbb{N}}$ *soddisfa una legge dei grandi numeri (LLN) se*

$$\lim_{n\to+\infty} \mathbf{P}\Big(\big|\bar{X}_n - \mu\big| > \epsilon\Big) = 0 \quad \forall \epsilon > 0. \tag{1.1}$$

Presentiamo di seguito un risultato che fornisce condizioni sufficienti affinché una successione di v.a. reali soddisfi una LLN (per la dimostrazione, cfr Proposizione 7.4 [1])

Proposizione 1.1. *(Legge debole dei grandi numeri) Sia* $(X_i)_{i\in\mathbb{N}}$ *una successione di v.a. reali su* $(\Omega, \mathcal{F}, \mathbf{P})$. *Assumiamo che:*

1. *esiste* $M \in [0; +\infty)$ *tc* $\mathbb{E}(X_i^2) \leq M$ *per ogni* $i \in \mathbb{N}$, *ossia le v.a. hanno momento secondo finito;*

2. $\mathbb{E}(X_i) = \mu$ *per ogni* $i \in \mathbb{N}$, $\mu \in \mathbb{R}$;

3. $Var(X_i) = \sigma^2 \in [0; +\infty)$ *per ogni* $i \in \mathbb{N}$;

4. $Cov(X_i, X_j) = 0$ *per ogni* $i \neq j$.

Allora la successione $(X_i)_{i\in\mathbb{N}}$ *soddisfa alla LLN.*

Le ipotesi che le variabili X_i siano scorrelate, abbiano lo stesso valor medio e la stessa varianza sono automaticamente soddisfatte per una successione $(X_i)_{i\in\mathbb{N}}$ di variabili aleatorie indipendenti ed identicamente distribuite (i.i.d.). Pertanto possiamo enunciare il seguente corollario:

Corollario 1.1. *Sia* $(X_i)_{i\in\mathbb{N}}$ *una successione di v.a. reali i.i.d. su* $(\Omega, \mathcal{F}, \mathbf{P})$ *con momento secondo finito. Allora* $(X_i)_{i\in\mathbb{N}}$ *soddisfa alla LLN.*

1.2 Il metodo Monte Carlo

Consideriamo il seguente problema di integrazione: vogliamo valutare $L(f) = \int_\Omega f(x)dx$ dove $f : \Omega \to \mathbb{R}$ limitata e $\Omega \subset \mathbb{R}^d$ avente misura di Lebesgue d-dimensionale $0 < \lambda_d(\Omega) < +\infty$. Il metodo Monte Carlo consiste nell'interpretare l'integrale di cui sopra come quantità stocastica. A tal proposito, dotiamo il dominio di integrazione Ω della misura di probabilità $d\mu = dx/\lambda_d(\Omega)$ e riscriviamo:

$$L(f) = \int_\Omega f(x)dx = \lambda_d(\Omega) \int_\Omega f(x)d\mu(x) = \lambda_d(\Omega)\mathbb{E}[f(X)]. \tag{1.2}$$

La formula (1.2) ci rivela che il calcolo dell'integrale $L(f)$ può essere ricondotto al determinare il valore atteso di $f(X)$ con X v.a. uniforme su Ω.

Proviamo dunque a stimare tale quantità sfruttando la Legge dei grandi numeri: sia $(X_i)_{i\in\mathbb{N}}$ una successione di v.a. i.i.d. dove X_i ha legge uniforme su Ω per ogni $i \in \mathbb{N}$. Ricordiamo che:

$$\mathbb{E}[f(X_i)] = \frac{1}{\lambda_d(\Omega)} \int_\Omega f(x)dx \ , \ i \in \mathbb{N}.$$

Ora, consideriamo la successione $(f(X_i))_{i\in\mathbb{N}}$. Si noti che quest'ultima è una successione di v.a. i.i.d. e con momenti secondi finiti (in quanto f è limitata e $X_1 \dots X_N$ hanno momenti secondo finiti (cfr [1] p. 262) e quindi sono soddisfatte le ipotesi del Corollario (1.1). In particolare, fissato $N \in \mathbb{N}$ si ha:

$$\lim_{N\to+\infty} \mathbf{P}\left(\left| \frac{1}{N} \sum_{i=1}^{N} f(X_i) - \frac{1}{\lambda_d(\Omega)} \int_\Omega f(x)dx \right| > \epsilon \right) = 0, \quad \forall \epsilon > 0;$$

ossia con "grande probabilità" vale la seguente approssimazione:

$$\frac{1}{\lambda_d(\Omega)} \int_\Omega f(x)dx \approx \frac{1}{N} \sum_{i=1}^{N} f(X_i). \tag{1.3}$$

Quindi, alla luce delle relazioni (1.2) e (1.3), si deduce la seguente stima (detta *stima di Monte Carlo*):

$$L(f) = \int_\Omega f(x)dx \approx \frac{\lambda_d(\Omega)}{N} \sum_{i=1}^{N} f(X_i) =: L_N(f). \tag{1.4}$$

Nota 1. *Possiamo interpretare la N-upla di v.a. $(X_1, \dots, X_N)$ come punti random nel dominio d'integrazione Ω*

Nota 2. *Quanto abbiamo svolto ci suggerisce la natura probabilistica della stima fornita in (1.4). Per ovviare a ciò ed ottenere un risultato coerente con quanto stiamo approssimando, generalmente si cambiano gli N punti random, disponendo così di più stime, ed infine si considera la media dei valori $L_N(f)$ ottenuti.*

Nota 3. *A livello computazione, ci si può imbattere in domini Ω talmente complicati da non essere in grado di risalire al valore esatto di $\lambda_d(\Omega)$, il che renderebbe inutile la stima Monte Carlo fornita in (1.4). Possiamo tuttavia arginare tale problema operando un cambio di variabili in modo che Ω sia contenuto nell'ipercubo $[0,1]^d$, sicchè*

$$\int_\Omega f(x)dx = \int_{[0,1]^d} f(x)\chi_\Omega(x)dx,$$

ove χ_Ω è la funzione indicatrice di Ω. Usando (1.4) all'ultimo integrale, si ottiene la stima

$$\int_\Omega f(x)dx \approx \frac{1}{N}\sum_{\substack{i=1\\ x_i\in\Omega}}^{N} f(x_i),$$

dove $x_1, \dots, x_N$ sono punti casuali di $[0,1]^d$.

1.2.1 Analisi dell'errore

Relativamente all'errore probabilistico dell'integrazione Monte Carlo, definiamo la varianza di f come:

$$\sigma^2(f) = \int_\Omega (f(x) - \mathbb{E}[f(X)])^2 d\mu(x), \tag{1.5}$$

la quale risulta essere finita se $f \in L^2(\mu)$.

Si ricordi il seguente Teorema (per la dimostrazione, cfr Theorem 1.1 [2])

Teorema 1.1. *Sia $f \in L^2(\mu)$. Allora, per ogni $N \geq 1$, si ha:*

$$\int_\Omega \cdots \int_\Omega \left(\frac{1}{N}\sum_{i=1}^{N} f(X_i) - \mathbb{E}[f(X)] \right)^2 d\mu(X_1)\dots d\mu(X_N) = \frac{\sigma^2(f)}{N}. \tag{1.6}$$

Il Teorema (1.1) ci rivela che l'errore di approssimazione del valore atteso (1.3) è, in media, $\sigma(f)N^{-1/2}$, ove $\sigma(f) = (\sigma^2(f))^{1/2}$ è la deviazione standard di f. Ulteriori informazioni sull'errore si possono ottenere grazie al Teorema limite centrale [1] [2], di cui forniamo una sua riformulazione coerente al nostro caso.

Teorema 1.2. *(Teorema limite centrale) Se $0 < \sigma(f) < +\infty$, allora esiste una variabile aleatoria normale standard $\nu \in N(0,1)$ tale che:*

$$\lim_{N\to+\infty} \left| \frac{1}{N}\sum_{i=1}^{N} f(X_i) - \mathbb{E}[f(X)] \right| = \lambda_d(\Omega)\sigma(f)N^{-1/2}\nu. \tag{1.7}$$

Da quest'ultimo risultato si deduce che l'errore dell'integrazione Monte Carlo è dell'ordine $\mathcal{O}(N^{-1/2})$ a meno di fattori moltiplicativi. Inoltre, ricaviamo che tale errore è una variabile aleatoria avente distribuzione normale standard (il che conferma la peculiarità probabilistica dei metodi Monte Carlo). Un'ulteriore e rilevante osservazione è l'indipendenza della stima dalla dimensione d dello spazio, a differenza dei bound d'errore date dalle regole d'integrazione d-dimensionali classiche, le quali sono dell'ordine $\mathcal{O}(N^{-2/d})$. Quest'ultimo confronto, ci permette di concludere che, da un punto di vista computazionale, il metodo Monte Carlo si dimostra più conveniente rispetto a quelli classici per dimensioni $d \geq 5$.

1.3 I metodi quasi-Monte Carlo

Come accennato nella sezione precedente, il metodo Monte Carlo utilizza punti casuali nel dominio d'integrazione e genera un errore di approssimazione di natura aleatoria il cui ordine risulta $\mathcal{O}(N^{-1/2})$. Tale errore, inoltre, non dipende dalla dimensione d del dominio.

Tuttavia, possiamo migliorare la stima d'errore andando ad usare punti deterministici, in modo da avere una distribuzione il più uniforme possibile sullo spazio, indistintamente dalla sua dimensione. Una tale successione di punti prende il nome di *sequenza a bassa discrepanza*, la quale risulta di fondamentale importanza nella definizione dei cosiddetti *metodi quasi-Monte Carlo*.

1.3.1 Sequenze uniformemente distribuite

Prima di trattare le sequenze a bassa discrepanza, presentiamo un concetto che ci tornerà utile nella trattazione della nostra discussione: le *sequenze uniformemente distribuite*.

Supponiamo che il dominio di integrazione Ω sia l'ipercubo unitario d-dimensionale chiuso $[0,1]^d$ e sia $f : [0,1]^d \to \mathbb{R}$. Consideriamo il seguente problema di *approssimazione quasi-Monte Carlo*

$$\int_{[0,1]^d} f(x)dx \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i), \tag{1.8}$$

dove $x_1, \dots, x_N \in [0,1]^d$. Idealizzando tale modello, possiamo rimpiazzare la sequenza finita $x_1, \dots, x_N$ con una successione numerabile di punti $\{x_i\}_{i\in\mathbb{N}} \subset [0,1]^d$. Ragionevolmente, da (1.8) richiediamo che una tale sequenza produca un metodo convergente, ossia richiediamo che $\{x_i\}_{i\in\mathbb{N}}$ sia tale che

$$\lim_{N\to+\infty} \frac{1}{N}\sum_{i=1}^{N} f(x_i) = \int_{[0,1]^d} f(x)dx, \tag{1.9}$$

e che tale uguaglianza sussista per una determinata classe di funzioni (ad esempio lo spazio delle funzioni continue in $[0,1]^d$). La condizione (1.9) è equivalente al richiedere che la sequenza $\{x_i\}_{i\in\mathbb{N}}$ sia *uniformemente distribuita* su $[0,1]^d$.

Definizione 1.6. *Siano E uno spazio di Hausdorff compatto e μ una misura positiva regolare normata di Borel, ossia una misura soddisfacente alle seguenti condizioni*

(i) $\mu(E) = 1$,

(ii) $\mu(F) = \sup\{\mu(C) : C \subseteq F, C \text{ chiuso}\}$ *per ogni boreliano F in E,*

(iii) $\mu(F) = \inf\{\mu(D) : F \subseteq D, D \text{ aperto}\}$ *per ogni boreliano F in E.*

*Diremo che la sequenza $\{x_i\}_{i\in\mathbb{N}} \subset E$ è **μ-uniformemente distribuita** in E se per ogni $f \in \mathcal{C}(E)$ vale:*

$$\lim_{N\to+\infty} \frac{1}{N}\sum_{i=1}^{N} f(x_i) = \int_E f(x)d\mu(x), \tag{1.10}$$

dove denotiamo con $\mathcal{C}(E)$ lo spazio delle funzioni reali e continue in E.

Nota 4. *Per i nostri scopi, useremo $\mu = \lambda_d$, ovvero la misura di Lebesgue d-dimensionale.*

Quanto detto ci suggerisce che in (1.8) è preferibile usare un insieme di nodi la cui distribuzione sia simile a quella uniforme su $[0,1]^d$, ovvero che i punti siano distribuiti in modo omogeneo su $[0,1]^d$.

1.3.2 Discrepanza

La discrepanza è un indice di dispersione uniforme di un insieme di punti nel dominio. In particolare, più i punti sono dispersi uniformemente, minore sarà la discrepanza. Tale concetto ammette diverse definizioni a seconda del dominio considerato, ma hanno tutti un punto in comune: quello di misurare l'uniformità della distribuzione di un insieme di punti.

Siano J un insieme, $X = \{x_1, \dots, x_N\} \subset J$ una sequenza finita di punti di J e B un sottoinsieme arbitrario di J. Definiamo la *counting function* come il numero di indici j tali che $x_j \in B$, ossia:

$$\#(B, X) = \sum_{j=1}^{N} \chi_B(x_j), \tag{1.11}$$

dove χ_B è la funzione caratteristica di B.

Siamo, quindi, pronti a dare la definizione di discrepanza.

Definizione 1.7. *Sia $J = \prod_{i=1}^{d}[\alpha_i, \beta_i) = \{x \in \mathbb{R}^d : \alpha_i \le x_i < \beta_i\}$ e sia $X = \{x_1, \dots, x_N\}$ una sequenza finita di punti di J. La **discrepanza** di X è la quantità*

$$D(X) = \sup_{B \subseteq J} \left| \frac{\#(B, X)}{N} - \lambda_d(B) \right|, \tag{1.12}$$

dove i sottoinsiemi B di J sono della forma $B = \{y \in J : a_i \le y_i < b_i, i \in \{1, \dots, d\}\}$

Se l'insieme J della Definizione (1.7) ammette un vertice in 0, si introduce la nozione di *discrepanza stellata*.

Definizione 1.8. *Sia $J^* = \prod_{i=1}^{d}[0, \beta_i) = \{x \in \mathbb{R}^d : 0 \le x_i < \beta_i\}$ e sia $X = \{x_1, \dots, x_N\}$ una sequenza finita di punti di J^*. La **discrepanza stellata** di X è la quantità*

$$D^*(X) = \sup_{B \subseteq J^*} \left| \frac{\#(B, X)}{N} - \lambda_d(B) \right|, \tag{1.13}$$

dove i sottoinsiemi B di J^* sono della forma $B = \{y \in J^* : 0 \le y_i < b_i, i \in \{1, \dots, d\}\}$

Relativamente alla discrepanza e alla discrepanza stellata sussiste la seguente disuguaglianza (per la dimostrazione, cfr. Proposition 2.4 [2]):

$$D^*(X) \le D(X) \le 2^d D^*(X). \tag{1.14}$$

Possiamo inoltre considerare un insieme convesso nella Definizione (1.7) e in tal caso si parla di *discrepanza isotropica*.

Definizione 1.9. *Sia C un insieme convesso limitato e sia $X = \{x_1, \dots, x_N\}$ una sequenza finita di punti di C. La* ***discrepanza isotropica*** *di X è la quantità*

$$J(X) = \sup_{B \in \mathcal{C}} \left| \frac{\#(B, X)}{N} - \lambda_d(B) \right|, \tag{1.15}$$

dove $\mathcal{C}$ è la famiglia di tutti i sottoinsiemi convessi di C.

Vale la seguente relazione tra discrepanza e discrepanza isotropica [2]:

$$D(X) \leq J(X) \leq 4dD(X)^{1/d}. \tag{1.16}$$

Presentiamo ora un risultato che lega i vari concetti di discrepanza appena presentati con l'uniforme distribuzione di una sequenza.

Teorema 1.3. *Siano $J = \prod_{i=1}^{d} [\alpha_i, \beta_i) = \{x \in \mathbb{R}^d : \alpha_i \leq x_i < \beta_i\}$ e $X \subset J$ una sequenza. Allora la seguenti proprietà sono equivalenti:*

(i) *X è uniformemente distribuito in J;*

(ii) $\lim_{N \to +\infty} D(X) = 0$;

(iii) $\lim_{N \to +\infty} D^*(X) = 0$;

(iv) $\lim_{N \to +\infty} J(X) = 0$;

Abbiamo ora tutti gli strumenti necessari per definire una sequenza *a bassa discrepanza.*

Definizione 1.10. *Una sequenza $X = \{x_1, \dots, x_n\}$ si dice* ***a bassa discrepanza*** *se*

$$D(X) \leq c(\log N)^k N^{-1}, \tag{1.17}$$

ove c e k sono costanti indipendenti da N, che possono però dipendere dalla dimensione d.

Esempi di sequenze a bassa discrepanza sono quelle di Halton, di Sobol e di Faure (per la loro definizione e costruzione si veda [2]). Un altro parametro che misura il grado di distribuzione uniforme dei punti è la *fill distance* o *meshsize.*

Definizione 1.11. *Dato un insieme $X = \{x_1, \dots, x_N\}$ di punti su un dominio compatto Ω, definiamo* ***fill distance*** *la seguente quantità:*

$$h_{X,\Omega} := \sup_{x \in \Omega} \min_{x_i \in X} \|x - x_i\|_2.$$

Geometricamente, la fill distance rappresenta il raggio della più grande palla costruibile in Ω non contenente alcuno dei punti di X. Questa interpretazione ci suggerisce che la fill distance rappresenta l'omogeneità della distribuzione dei punti di X all'interno del dominio Ω.

1.3.3 Analisi dell'errore

Sia $\Omega = [0,1]^d$. Allora, coerentemente con le notazioni usate, l'errore assoluto di un metodo di integrazione quasi-Monte Carlo è dato da

$$\epsilon(f) := \left| \frac{1}{N} \sum_{i=1}^{N} f(x_i) - \int_{[0,1]^d} f(x)dx \right|. \tag{1.18}$$

Al fine di presentare un risultato sull'errore di integrazione quasi-Monte Carlo (che è il Teorema di Koksma-Hlawka), introduciamo un concetto essenziale per la sua comprensione: la classe delle *funzioni a variazione limitata* [2].

Definizione 1.12. *Sia $f : [0,1]^d \to \mathbb{R}$ una funzione e J un sottointervallo di $[0,1]^d$. Sia $\Delta(f;J)$ una somma alternata di valori di f nei vertici di J (ossia f assume valori opposti in vertici adiacenti). La* ***variazione di f su $[0,1]^d$ in senso di Vitali*** *è definita da*

$$V^{(d)}(f) = \sup_{\mathcal{P}} \sum_{J \in \mathcal{P}} |\Delta(f;J)|, \tag{1.19}$$

dove l'estremo superiore è esteso a tutte le partizioni $\mathcal{P}$ di $[0,1]^d$ in sottointervalli. Inoltre, se la derivata parziale $\frac{\partial^d f}{\partial t_1 \dots \partial t_d}$ sia continua in $[0,1]^d$, la (1.19) è equivalente a

$$V^{(d)}(f) = \int_0^1 \cdots \int_0^1 \left| \frac{\partial^d f}{\partial t_1 \dots \partial t_d} \right| dt_1 \dots dt_d.$$

Per $1 \le k \le d$ e $1 \le i_1 < i_2 < \cdots < i_k \le d$, sia $V^{(k)}(f; i_1, \dots, i_k)$ la variazione in senso di Vitali della restrizione di f alla faccia k-dimensionale $\{(u_1, \dots, u_d) \in [0,1]^d : u_j = 1 \text{ per } j \neq i_1, \dots, i_k\}$. Allora

$$V(f) = \sum_{k=1}^{d} \sum_{1 \le i_1 < i_2 < \cdots < i_k \le d} V^{(k)}(f; i_1, \dots, i_k)$$

è detta ***variazione di f su $[0,1]^d$ in senso di Hardy-Krause****. Se $V(f)$ è finita, f si dice a* ***variazione limitata in senso di Hardy-Krause****.*

Teorema 1.4. *(di Koksma-Hlawka) Sia $f : [0,1]^d \to \mathbb{R}$ una funzione a variazione limitata nel senso di Hardy-Krause e sia $X = \{x_1, \dots, x_N\} \subset [0,1]^d$ una generica sequenza. Allora*

$$\epsilon(f) \le V(f) D_N^*(X), \tag{1.20}$$

dove $D_N^*(X)$ è la discrepanza stellata di X.

Come immediata conseguenza del Teorema 1.4 si ricava il seguente

Corollario 1.2. *L'errore di integrazione per i metodi quasi-Monte Carlo è dell'ordine* $\mathcal{O}((\log N)^k N^{-1})$*, con* k *dipendente dalla dimensione* d

Dimostrazione. Si consideri $X = \{x_1, \dots, x_N\}$ sequenza a bassa discrepanza. Allora per (1.17) si ha $D_N(X) \leq c(\log N)^k N^{-1}$ con c, k indipendenti da N ma eventualmente dipendenti dalla dimensione d. Sfruttando (1.14) e il Teorema 1.4, si ricava che

$$\epsilon(f) \leq V(f) D_N^*(X) \leq V(f) D_N(X) \leq cV(f)(\log N)^k N^{-1},$$

il che prova la tesi. □

Capitolo 2

Compressione

Quanto esposto in questo capitolo è tratto dagli articoli *"Compressed QMC volume and surface integration on union of balls"* (cf. [3]) e *"Compressed QMC: an improved code for low-dimensional Compressed Quasi-Monte Carlo cubature"* (cf. [4]). Il proposito é di introdurre il concetto di compressione di una formula di quadratura per poi particolarizzarlo all'ambito dei metodi quasi-Monte Carlo.

2.1 Compressione di una formula di quadratura

Siano $\Omega \subset \mathbb{R}^d$ un insieme compatto e $f\colon \Omega \to \mathbb{R}$ una funzione continua. Consideriamo una formula di quadratura avente supporto $X = \{P_i\}_i \subset \Omega$ e pesi ad essi associati $\lambda = \{\lambda_i\}_i$, $1 \leq i \leq N$. Comprimere una formula di quadratura significa esprimere l'integrale su Ω di f mediante una somma finita, estraendo un sottoinsieme di punti di X e ripesandoli opportunamente:

$$L_{INT}(f) = \int_\Omega f(x)dx \approx S(f) = \sum_{i=1}^{N} \lambda_i f(P_i) \approx \sum_{k=1}^{M} w_k f(Q_k), \tag{2.1}$$

ove

$$\{Q_k\} \subset X\ ,\ \ M = \mathrm{card}(\{Q_k\}) \leq r = \dim(\mathbb{P}_n^d(X)) < N.$$

Denotiamo con $\mathbb{P}_n^d(X)$ lo spazio dei polinomi d-variati aventi grado al più n e ristretti ad X e con $\{\phi_1, \ldots, \phi_r\}$ una sua base.

Da un punto di vista computazionale, ci interessa trovare un algoritmo che estragga dei punti e determinando i pesi corrispondenti, in modo tale che:

$$S(f) = \sum_{i=1}^{N} \lambda_i p(P_i) = \sum_{k=1}^{M} w_k p(Q_k)\ ,\ \ \forall p \in \mathbb{P}_n^d(X).$$

Nel seguito, presentiamo i vari passi di tale algoritmo:

- definiamo la matrice

$$V(X) = V_N(X) = (\phi_j(P_i)) \in \mathbb{R}^{N \times r},\ N > r \tag{2.2}$$

  con $1 \leq i \leq N = \mathrm{card}(X)$, $1 \leq j \leq r = \dim(\mathbb{P}_n^d(X))$

- consideriamo il seguente sistema lineare sotto-determinato (chiamato *sistema dei momenti coniugati*):
$$V^t(X)u = m_X,\ u \geq 0, \tag{2.3}$$
dove $m_X = V^t(X)\lambda$, e risolviamo (1.3) cercando la soluzione del problema NNLS (NonNegative Least-Squares) corrispondente:
$$\min\{\|V^t(X)u - m_X\|_2, u \in \mathbb{R}^r, u \geq 0\}; \tag{2.4}$$

- posti $w = \{w_k = u_{i_k} : u_{i_k} > 0\}$ e $T = \{t_k = x_{i_k} : u_{i_k} > 0\}$, la misura iniziale verrà compressa in un'altra misura avente supporto $T \subseteq X$ tale che:
$$V^t(T)w = m_X,\ w \in \mathbb{R}^\nu,\ \nu = card(T) \leq r, \tag{2.5}$$
o equivalentemente
$$\sum_{k=1}^{\nu} w_k p(t_k) = \sum_{i=1}^{N} \lambda_i p(x_i), \quad \forall p \in \mathbb{P}_n^d(X). \tag{2.6}$$

La liceità di 2.6 è garantita dal seguente Teorema

Teorema 2.1. *(di Tchakaloff) Siano μ una misura discreta multivariata avente supporto finito $X = \{P_i\} \subset \mathbb{R}^d$ e $\lambda = \{\lambda_i\}$, $1 \leq i \leq M$, le masse corrispondenti. Sia inoltre $S = span\{\phi_1, \ldots, \phi_L\}$ uno spazio di dimensione finita di funzioni d-variate definite su $K \supset X$ con $N = dim(S\big|_X) \leq L$. Allora esiste una formula di quadratura con nodi $Q = \{Q_j\} \subseteq X$ e pesi positivi $w = \{w_j\}$, $1 \leq j \leq m \leq N$ tale che:*
$$\int_X f(x)d\mu = \sum_{i=1}^{M} \lambda_i f(P_i) = \sum_{j=1}^{m} w_j f(Q_j)\ ,\ \forall f \in S\big|_X.$$

Nota 5. *Di seguito assumeremo che l'insieme $X \subset \Omega$ determini in modo univoco una base di $\mathbb{P}_n^d$, ossia*
$$r = \dim(\mathbb{P}_n^d) = \binom{n+d}{d} \tag{2.7}$$
(o equivalentemente, se $p \in \mathbb{P}_n^d$ è tale che $p(P_i) = 0\ \forall P_i \in X$, allora $p \equiv 0$ su Ω)

2.2 Formule Quasi-Monte Carlo compresse

La compressione delle formule Quasi-Monte Carlo (QMC) è la compressione di una particolare misura discreta. Ricordando quanto visto nella Sezione 2.1, quest'ultima consiste nell'estrarre un sottoinsieme di punti dal supporto di una misura discreta e ripesare le corrispondenti masse, col vantaggio di preservare l'errore di approssimazione. Relativamente alla compressione nell'ambito della quadratura numerica, presentiamo il seguente risultato sulla rappresentazione discreta degli operatori lineari positivi.

Teorema 2.2. *(Teorema di Davis - Wilhelmsen)*
Siano $\Omega \subset \mathbb{R}^d$ un insieme compatto e $\Psi = \text{span}\{\phi_1, \ldots, \phi_N\}$, ove $\{\phi_j\}_{j=1}^N$ è un sistema finito di funzioni reali linearmente indipendenti definite su Ω. Supponiamo che Ψ soddisfi alla condizione di Krein, ossia esiste almeno una funzione $f \in \Psi$ non identicamente nulla su Ω. Sia inoltre $L\colon \Psi \to \mathbb{R}$ un operatore lineare positivo (ossia $L(f) > 0$ per ogni f non-negativa). Se $\{P_i\}_{i=1}^\infty$ è ovunque un sottoinsieme denso di Ω, allora per m sufficientemente grande l'insieme $X_m = \{P_i\}_{i=1}^m$ è un insieme di Tchakaloff, ovvero esistono dei nodi $\{\mathcal{T}_k\}_{k=1}^\nu \subset X_m \subset \Omega$ e dei pesi $w_k > 0 \; \forall\, k \in \{1, \ldots, \nu\}$ con $\nu = \text{card}(\{\mathcal{T}_k\}) \leq N$ tali che:

$$L(f) = \sum_{k=1}^{\nu} w_k f(\mathcal{T}_k), \;\; \forall f \in \Psi. \tag{2.8}$$

Come conseguenza immediata del Teorema 2.2, asseriamo il seguente

Corollario 2.1. *Sia λ una misura positiva su un insieme compatto $\Omega \subset \mathbb{R}^d$ e supponiamo che supp(λ) determini univocamente lo spazio dei polinomi d-variati di grado al più n $\mathbb{P}_n^d(\Omega)$(ossia se un polinomio in $\mathbb{P}_n^d(\Omega)$ si annulla su supp(λ), allora è identicamente nullo su Ω). Allora abbiamo la tesi del Teorema 1.1 definendo $L(f) = L_{INT}(f) = \int_\Omega f d\lambda$.*

Una interessante applicazione consiste nel fatto che sia possibile applicare il Corollario 2.1 all'operatore di *Quasi-Monte Carlo*, ovvero a

$$L_{QMC}(f) = \frac{\mu(\Omega)}{M} \sum_{i=1}^{M} f(P_i) \approx L_{INT}(f) = \int_\Omega f(P) dP, \tag{2.9}$$

dove $\Omega \subset \mathbb{R}^2$ é un insieme compatto, $f\colon \Omega \to \mathbb{R}$ continua, $N = \dim(\Psi) = \dim(\mathbb{P}_n^2(\Omega))$ e i punti

$$X_M = \{P_i\}_{i=1}^M \subset \Omega\,, \;\; M > N \tag{2.10}$$

sono estratti da una sequenza a bassa discrepanza su Ω.

Nota 6. *A seconda dei domini, $\mu(\Omega)$ può rappresentare il volume (che sarà il nostro caso) o l'area superficiale di Ω. Generalmente tale quantità non è nota esattamente e quindi si procede come segue. Dapprima si genera una sequenza a bassa discrepanza di cardinalità M_0 su un volume o superficie $\mathcal{B} \supseteq \Omega$ (detto bounding box) e di seguito si estraggono M punti da tale sequenza usando un'opportuna funzione in-domain che stabilisce se un punto appartenga o meno a Ω. Di conseguenza, vale la seguente approssimazione:*

$$\mu(\Omega) \approx \mu(\mathcal{B}) \frac{M}{M_0}.$$

Nota 7. *La positività dell'operatore L_{QMC} per ogni $f \in \mathbb{P}_n^2$ è garantita dalla condizione di determinazione di una base di $\mathbb{P}_n^2$ a partire dall'insieme X_M. Quest'ultima è, in particolare, equivalente al richiedere che* $\dim(\mathbb{P}_n^2(X_M)) = N = \dim(\mathbb{P}_n^2(\Omega))$ *o anche:*

$$\text{rank}(V_M) = N,$$

dove

$$V_M = V^{(N)}(X_M) = (\phi_j(P_i)) \in \mathbb{R}^{M \times N}$$

è una matrice di Vandermonde generalizzata e $\{\phi_j\}_{j=1\dots N}$ *è una base di* $\mathbb{P}^2_n$. *Inoltre, si noti che, poichè* X_M *è una sequenza di punti, per* $k \leq M$ *si può definire la matrice*

$$V_k = V^{(N)}(X_k) = (V_M)_{i,j}\ ,\ \ 1 \leq i \leq k,\ 1 \leq j \leq N.$$

Ora, si supponga che $M \gg N$ e, applicando il Teorema 2.2 e quanto affrontato nella Sezione 2.1, proviamo ad esibire un insieme di Tchakaloff X_m, con $N \leq m < M$. Consideriamo il seguente sistema lineare sotto-determinato (noto come *sistema dei momenti coniugati*)

$$V_m^t u = p = V_M^t e\ ,\ \ e = \frac{\mu(\Omega)}{M}(1,\dots,1)^t \in \mathbb{R}^M, \tag{2.11}$$

dove $u \in \mathbb{R}^m$, $u \geq 0$. Si noti che è possibile riscrivere 2.11 come:

$$\sum_{i=1}^{m} u_i \phi_j(P_i) = \frac{\mu(\Omega)}{M} \sum_{k=1}^{M} \phi_j(P_k)\ ,\ \ j \in \{1,\dots,N\}. \tag{2.12}$$

Risolviamo 2.11 sfruttando l'algoritmo di Lawson-Hanson al problema NNLS (NonNegative Least-Squares) associato:

$$\min_{u \geq 0} \|V_m^t u - p\|_2. \tag{2.13}$$

Fissata una tolleranza $\epsilon > 0$, accettiamo la soluzione di 2.13 qualora

$$\|V_m^t u - p\|_2 < \epsilon \tag{2.14}$$

Segue che l'insieme X_m è determinato dalle componenti non nulle del vettore u soddisfacente 2.14. Precisamente: definiti i pesi $\{w_k\} = \{u_i : u_i > 0\}$ e i nodi $\{\mathcal{T}_k\} = \{P_i : u_i > 0\}$, il funzionale L_{QMC} viene compresso in L^*_{QMC} definito da:

$$L^*_{QMC}(f) = \sum_{k=1}^{\nu} w_k f(\mathcal{T}_k)\ ,\ \ \nu = \mathrm{card}(\{\mathcal{T}_k\}) \leq N \ll M. \tag{2.15}$$

Da 2.12, si ricava la seguente

Proposizione 2.1. *Se* $f \in \mathbb{P}^2_n(\Omega)$ *allora*

$$L^*_{QMC}(f) = L_{QMC}(f).$$

Dimostrazione. Si fissi $f \in \mathbb{P}^2_n(\Omega)$. Allora $f = \sum_{j=1}^{N} a_j \phi_j$ per opportuni $a_j \in \mathbb{R}$, $j \in \{1,\dots,N\}$. Si deduce la seguente catena di uguaglianze:

$$\begin{aligned} L_{QMC}(f) &= \frac{\mu(\Omega)}{M} \sum_{k=1}^{M} f(P_k) = \frac{\mu(\Omega)}{M} \sum_{k=1}^{M} \sum_{j=1}^{N} a_j \phi_j(P_k) = \sum_{j=1}^{N} a_j \sum_{k=1}^{M} \frac{\mu(\Omega)}{M} \phi_j(P_k) \overset{(2.12)}{=} \\ &= \sum_{j=1}^{N} a_j \sum_{i=1}^{m} u_i \phi_j(P_i) = \sum_{i=1}^{m} u_i \sum_{j=1}^{N} a_j \phi_j(P_i) = \sum_{i=1}^{m} u_i f(P_i) = L^*_{QMC}(f), \end{aligned}$$

dove nell'ultima uguaglianza abbiamo compattato la sommatoria considerando i termini relativi agli $u_i > 0$ □

Si noti che una rappresentazione della forma (2.15) con la scelta $m = M$ è garantita dal Teorema 2.1. Tuttavia, tale scelta del parametro risulta estremamente costoso a livello computazionale in quanto si dovrebbe risolvere direttamente il problema NNLS:

$$\min_{u\geq 0}\|V_M^t u - p\|_2\ ,\ \ M \gg N.$$

D'altro canto, per abbattere il costo computazionale, si potrebbe risolvere 2.13 considerando una successione di problemi parametrizzati da $m := m_1, m_2, m_3 \dots$ con $m_1 < m_2 < m_3 < \cdots \leq M$,

$$\min_{u\geq 0}\|V_{m_j}^t u - p\|_2\ ,\ \ j = 1, 2, 3, \dots\ ,\ \ m_1 \geq N.$$

Una tale scelta corrisponde a una successione crescente di insiemi densi di Tchakaloff $X_{m_1} \subset X_{m_2} \subset \cdots \subseteq X_M$ ed estrarremo i punti dall'insieme X_{m_j} se

$$\|V_{m_j}^t u - p\|_2 < \epsilon, \tag{2.16}$$

dove $\epsilon > 0$ è una tolleranza fissata a priori. Di seguito forniamo un esempio di parametri tale da soddisfare 2.16 in poche iterazioni:

$$\begin{cases} m_1 \geq N \\ m_{l+1} = (1+\theta)m_l \end{cases}$$

con $\theta \in (0; 1)$.

Nota 8. *Possiamo applicare l'algoritmo visto in precedenza nel caso in cui Ω sia dato da un'unione finita di insiemi disgiunti, ossia $\Omega = \bigcup_{l=1}^{L} \Omega_l$. In tal caso, i punti che generano una sequenza a bassa discrepanza saranno presi dalle bounding box $\mathcal{B}_l \supset \Omega_l$. Di conseguenza, definiti $Y_l = \{P_{l,i}\}_{i=1}^{M_l}$ le sequenze a bassa discrepanza di Ω_l, $1 \leq l \leq L$, avremo che la sequenza a bassa discrepanza relativa ad Ω sarà $X = \bigcup_{l=1}^{L} Y_l$, con $M = \mathrm{card}(X) = \sum_{l=1}^{L} M_l$. Si noti che, per avere una sequenza uniformemente distribuita su Ω, i punti devono essere scelti alternativamente in ogni Ω_l, ossia prendiamo il primo punto su ciascun Ω_l, il secondo punto su ciascun Ω_l e così via, sicchè considereremo la sequenza $\{P_{1,1}, P_{2,1}, \dots, P_{L,1}, P_{1,2}, P_{2,2}, \dots, P_{L,2}, \dots\}$. Pertanto, per l'additività dell'integrale si ottiene che il funzionale quasi-Monte Carlo è dato da:*

$$L_{QMC}(f) = \sum_{l=1}^{L}\sum_{i=1}^{M_l} w_{l,i} f(P_{l,i}) \approx \sum_{l=1}^{L}\int_{\Omega_l} f(P)dP = \int_{\Omega} f(P)dP,$$

ove $w_{l,i} = \mu(\Omega_l)/M_l$ per $i = 1, \dots, M_l$.

2.3 Analisi dell'errore

Presentiamo ora un risultato sull'errore di approssimazione tra l'integrale esatto e la formula compressa:

Teorema 2.3. *Siano $\Omega \subset \mathbb{R}^2$ un insieme compatto, $f \in C(\Omega; \|\cdot\|_{\infty,\Omega})$ e sia $p_n^* \in \mathbb{P}_n^2$ il polinomio di miglior approssimazione di f su Ω. Allora*

$$\left|L_{INT}(f) - L^*_{QMC}(f)\right| \leq |L_{INT}(f) - L_{QMC}(f)| + 2\mu(\Omega)E_n(f;\Omega), \tag{2.17}$$

ove

$$E_n(f;\Omega) = \min_{p\in\mathbb{P}_n^2} \|f-p\|_{\infty,\Omega} = \|f - p_n^*\|_{\infty,\Omega} \tag{2.18}$$

è l'errore compiuto dal polinomio di miglior approssimazione di f

Dimostrazione. Si osservi che:

$$|L_{QMC}(f) - L_{QMC}(p_n^*)| = \left|\sum_{i=1}^{M} \frac{\mu(\Omega)}{M} f(P_i) - \sum_{i=1}^{M} \frac{\mu(\Omega)}{M} p_n^*(P_i)\right| \tag{2.19}$$

$$\leq \frac{\mu(\Omega)}{M} \sum_{i=1}^{M} |f(P_i) - p_n^*(P_i)| \leq \mu(\Omega)\|f - p_n^*\|_{\infty,\Omega}$$

$$\left|L^*_{QMC}(p_n^*) - L^*_{QMC}(f)\right| = \left|\sum_{k=1}^{\nu} w_k p_n^*(\mathcal{T}_k) - \sum_{k=1}^{\nu} w_k f(\mathcal{T}_k)\right| \tag{2.20}$$

$$\leq \sum_{k=1}^{\nu} w_k \left|p_n^*(\mathcal{T}_k) - f(\mathcal{T}_k)\right| \leq \left(\sum_{k=1}^{\nu} w_k\right) \|f - p_n^*\|_{\infty,\Omega}.$$

Inoltre, poiché

$$\sum_{k=1}^{\nu} w_k = \sum_{k=1}^{\nu} w_k \cdot 1 \overset{2.11}{=} \sum_{i=1}^{M} \frac{\mu(\Omega)}{M} \cdot 1 = \mathrm{vol}(\Omega), \tag{2.21}$$

si ha, per la disuguaglianza triangolare, che

$$\begin{aligned}\left|L_{QMC}(f) - L^*_{QMC}(f)\right| \leq{}& |L_{QMC}(f) - L_{QMC}(p_n^*)| + \underbrace{\left|L_{QMC}(p_n^*) - L^*_{QMC}(p_n^*)\right|}_{=0 \text{ per } 2.11} \\ &+ \left|L^*_{QMC}(p_n^*) - L^*_{QMC}(f)\right| \\ \overset{2.19,2.20}{\leq}{}& \mu(\Omega)\|f - p_n^*\|_{\infty,\Omega} + \left(\sum_{k=1}^{\nu} w_k\right) \|f - p_n^*\|_{\infty,\Omega} \\ \overset{2.21}{=}{}& 2\mu(\Omega)\|f - p_n^*\|_{\infty,\Omega}.\end{aligned} \tag{2.22}$$

In definitiva:

$$\begin{aligned}\left|L_{INT}(f) - L^*_{QMC}(f)\right| &\leq |L_{INT}(f) - L_{QMC}(f)| + \left|L_{QMC}(f) - L^*_{QMC}(f)\right| \\ &\overset{2.22}{\leq} |L_{INT}(f) - L_{QMC}(f)| + 2\mu(\Omega)\|f - p^*_n\|_{\infty,\Omega}.\end{aligned}$$

□

Il Teorema 2.3 ci rivela che l'operatore relativo alla compressione QMC, L^*_{QMC}, preserva l'errore di approssimazione dato dal funzionale QMC, L_{QMC}, a meno di un fattore additivo, dato dall'errore di miglior approssimazione di f rispetto alla norma su Ω. Si ricordi che quest'ultima quantità, a seconda della regolarità dell'integranda f, può essere stimata dai Teoremi di Jackson multivariati.

Capitolo 3

Risultati numerici

In questo ultimo capitolo si intende approfondire numericamente quanto discusso finora.

Ciò che ci si aspetta di osservare è che le regole QMC e CQMC applicate all'integrazione numerica di un dominio bivariato producano risultati simili, col vantaggio che nel caso compresso si utilizzano un numero molto ridotto di nodi (e opportunamente pesati) rispetto alle formule QMC classiche.

Per i nostri scopi, ci occuperemo del calcolo approssimato di integrali su dischi, ellissi, poligoni convessi e su regioni di piano dati da operazioni insiemistiche su suddetti insiemi, come ad esempio l'unione, l'intersezione o la differenza.

La tecnica utilizzata sarà la seguente

- si determina la bounding box $\mathcal{B}$ del dominio;
- si definisce su $\mathcal{B}$ una sequenza a bassa discrepanza sul quadrato unitario $[0,1]^2$ e si mappano i primi N punti nella bounding box $\mathcal{B}$;
- si determinano mediante una funzione di tipo *indomain* quei punti tra i precedenti che appartengono al dominio di integrazione Ω;
- si definisce la relativa formula QMC;
- si applica l'algoritmo per il calcolo della formula CQMC (cf. [3]), utilizzando opportunamente l'algoritmo di Lawson-Hanson.

Nei prossimi paragrafi si illustreranno alcuni esempi, svolti usando il software *'Matlab R2022b'* con processore *'Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz 2.71 GHz'* e RAM da 4,0 GB.

3.1 Gestione numerica dei domini

Relativamente alla definizione di insiemi bivariati e delle corrispondenti bounding box, si è creata una shell in Matlab che utilizza routines basate su due differenti strutture dati ovvero le *struct* e le *classi*.

Visto il proposito di dotare la routine CQMC di una shell che permetta più facilmente ad un utente di definire il dominio bivariato di integrazione, anche mediante operazioni

insiemistiche su domini più semplici, abbiamo verificato che tali strutture risultano equalmente onerose dal punto di vista computazionale, e permettono entrambe di definire facilmente una certa varietà di domini di integrazione anche aventi geometrie particolarmente complicate.

Di seguito si illustreranno, per ciascuna struttura dati, le funzioni utili alle esigenze sopra indicate.

3.1.1 Struct

Nel caso dell'utilizzo di *struct*, abbiamo introdotto le routines:

- la funzione `define_domain` che definisce un dominio bivariato.

  Tale funzione ammette come argomenti un nome (ovvero una stringa) che può essere `disk` se si vuole un disco, `ellipse` un ellisse, `polygon` un poligono; inoltre possono essere inseriti alcuni parametri necessari nel modellizzarlo. In particolare i parametri per i vari domini sono:

  - le coordinate del centro e del raggio per il disco;
  - le coordinate del centro, la misura dei semiassi orizzontale e verticale per l'ellisse;
  - le coordinate dei vertici in senso antiorario per il poligono (senza il vincolo che il primo e l'ultimo punto coincidano).

  Si sottolinea inoltre che:

  - le coordinate del centro del disco e dell'ellisse sono passate come array contenente ordinatamente l'ascissa e l'ordinata del centro;
  - le coordinate dei vertici del poligono sono passate come matrice avente due colonne di cui la prima conterrà le ascisse dei vertici e la seconda le ordinate. Inoltre le coordinate dei vertici sono ordinate in senso antiorario (con il vertice iniziale e finale non necessariamente coincidenti).

  Vediamo quanto esposto finora su degli esempi. La chiamata:

```
define_domain('disk',[1 0], 3);
```

  genererà una struct che rappresenta un disco centrato in (1,0) e di raggio 3. Si noti che è possibile omettere la parte dei parametri come segue:

```
define_domain('polygon');
```

  In tal caso si avrà un poligono con parametri di *default*, ossia il quadrato unitario. L'omissione dei parametri è prevista anche per i restanti insiemi: per il disco si avrà

quello centrato nell'origine e raggio unitario: per l'ellisse quello centrato nell'origine, asse orizzontale e verticale lunghi rispettivamente 2 e 1.

Oltre a tutti i parametri necessari a definire un determinato dominio, la struct ad essa associata prevede la proprietà `bounding box`, la quale rappresenta il più piccolo rettangolo, con lati paralleli agli assi, contenente il dominio stesso. Per costruirlo, si calcolano i più piccoli intervalli in cui variano le ascisse e le ordinate dei punti del bordo risalendo così ai vertici della bounding box. Per la definizione della bounding box, si è usato il costruttore predefinito in Matlab `polyshape` con argomento una matrice avente per righe le coordinate dei vertici della bounding box calcolati precedentemente;

- `boundingbox_operation`: dati due domini (passati come primi argomenti della funzione), esegue un'operazione insiemistica fissata tra le due. Quest'ultima sarà passata come parametro e denotata con un unico carattere: `'U'` per l'unione, `'I'` per l'intersezione e `'D'` per la differenza. Produce in output un oggetto `polyshape`;

  Nota 9. *Si noti che la bounding box relativa all'unione di due domini (rispettivamente intersezione e differenza) è il più piccolo rettangolo contenente l'unione dei due insiemi (rispettivamente intersezione e differenza).*

- `indomain`: funzione che determina se uno o più punti sono interni al dominio (o, a richiesta, sul bordo).

  Ha per argomenti ordinatamente i punti che si vogliono testare (memorizzati in una matrice avente per righe le ascisse e le ordinate dei punti) e un dominio. Produce in output un array booleano in cui all'i-esima posizione si ha 0 o 1 se il punto all'i-esima riga della matrice in input è esterno o rispettivamente interno al dominio considerato;

- `plotdomain`: genera le coordinate di alcuni punti del bordo che delimita il dominio considerato. Tale routine funge da supporto per il plot.

3.1.2 Classi

Nel caso delle *classi* abbiamo introdotto

- `disk`: classe che modellizza un disco.

  Tale classe ha per proprietà il centro e il raggio ed ha due costruttori:

  - uno che ha per argomenti ordinatamente le coordinate del centro (contenute in un vettore) e il raggio;
  - uno di default che genera un'istanza di tipo `'disk'` relativa ad un disco centrato nell'origine e raggio unitario;

- `ellipse`: classe che modellizza un'ellisse.

  Tale classe ha per proprietà il centro, il semiasse orizzontale e verticale ed ha due costruttori:

  - uno che ha per argomenti ordinatamente le coordinate del centro (contenute in un vettore), la misura del semiasse orizzontale e verticale;
  - uno di default che genera un'istanza di tipo `'ellipse'` relativa ad un'ellisse centrato nell'origine e semiassi orizzontale e verticale rispettivamente pari a 2 e 1;

- `polygon`: classe che modellizza un poligono.

  Tale classe ha per proprietà una matrice con righe le coordinate dei vertici ordinate in senso antiorario ed ha due costruttori:

  - uno che ha per argomenti una matrice che ha per righe le coordinate dei vertici;
  - uno di default che genera un'istanza di tipo `'polygon'` relativa al quadrato unitario;

Per le tre classi appena presentate si sono definiti i seguenti metodi:

- `in_domain`: determina quali punti test sono interni o meno al dominio;
- `plotdomain`: genera le coordinate di alcuni punti del bordo che delimita il dominio considerato. Tale routine funge da supporto per il plot.

Inoltre, si osservi che:

- ogni classe è dotata della proprietà `bounding_box`, che rappresenta la bounding box relativa all'istanza considerata;
- per la gestione dell'unione, intersezione e differenza tra insiemi, si è ricorsa alla funzione `boundingbox_operation`.

Si ricorda che i metodi sopra citati e questi ultimi due aspetti sono stati implementati numericamente in modo analogo rispetto a quanto detto nella sottosezione 3.1.1.

3.1.3 Esempio sulla definizione del dominio di integrazione e della sua bounding box

In questo paragrafo andiamo ad illustrare un esempio relativo alla definizione della bounding box generata dall'unione di due dischi usando in sequenza sia le struct che le classi

```
d1 = define_domain('disk',[1 -1], 2);
d2 = define_domain('disk',[-2 3], 1);
bb = boundingbox_operation(d1,d2,'U');
```

```
d1 = disk([1 -1], 2);
d2 = disk([-2 3], 1);
bb = boundingbox_operation(d1,d2,'U');
```

Dettagliatamente:

- si definiscono due dischi `d1` e `d2` rispettivamente centrati in (1,-1), (-2,3) e di raggio 2 ed 1;
- si calcola la bounding box `bb` associata all'unione di `d1` e `d2` con l'ausilio delle bounding box dei domini (che sono già predisposte in quanto proprietà).

3.1.4 Determinazione di formule QMC e CQMC

Relativamente alla determinazione dei punti a bassa discrepanza che per i nostri scopi saranno i punti di Halton e al calcolo delle formule QMC e compresse, ovvero CQMC, citiamo alcune funzioni utili agli scopi appena detti (per la loro documentazione dettagliata si veda l'appendice):

- `haltonseq`: dati due interi `card` e `dim`, tale routine genera i primi `card` punti della sequenza di Halton nello spazio `dim`-variato. Tali punti hanno coordinate comprese tra 0 e 1;
- `cqmc`: dati i nodi QMC e il grado di precisione, genera i nodi e i pesi della formula compressa usando l'algoritmo di Lawson-Hanson.

3.2 Esperimenti numerici

In questa sezione si presenteranno alcuni esempi di calcolo di formule QMC e CQMC su domini bivariati ottenuti mediante operazione insiemistica su dischi, ellissi e poligoni.

Per ogni esempio si considereranno gradi di precisione variabili e per ognuno di essi si raccoglieranno particolari dati, quali:

- il numero di punti a bassa discrepanza "*card QMC*" usati nella bounding box associata al dominio di riferimento,
- il numero di tali punti "*card QMC in*" appartenenti al dominio stesso,
- il numero di punti "*card CQMC*" appartenenti al dominio stesso e usati per la costruzione della formula compressa e il rapporto "*compr ratio*" tra tali quantità;
- l'errore "*mom res CQMC*" sulla ricostruzione dei momenti (in norma 2) (cf. [3] pag. 1);

3.2.1 Esperimenti sul disco e sull'ellisse

In questo primo esempio, consideriamo rispettivamente un disco centrato in $(-1, 0)$ di raggio 4 e una ellisse centrata in $(2, -2)$ con semiasse orizzontale e verticale pari a $\sqrt{12}$ e $\sqrt{6}$ rispettivamente.

Vediamo di seguito le righe di codice fondamentali utilizzate nell'implementare i tests.

- Per definire i sottodomini `d1` e `d2` tramite i quali si ottiene Ω mediante operazioni insiemistiche, basta effettuare, rispettivamente nel caso degli *struct* e delle *classi*, le seguenti assegnazioni

```
d1 = define_domain('disk',[-1 0], 4);
d2 = define_domain('ellipse',[2 -2],sqrt(12),sqrt(6));
```

```
d1 = disk([-1 0], 4);
d2 = ellipse([2 -2],sqrt(12),sqrt(6));
```

- attraverso la chiamata

```
bb=boundingbox_operation(d1,d2,operation_parm);
```

si ottiene la bounding box `bb` contenente

(a) l'unione di `d1` con `d2` (qualora `operation_parm='U'`),

(b) l'intersezione di `d1` con `d2` (qualora `operation_parm='I'`),

(c) la differenza di `d1` con `d2` (qualora `operation_parm='D'`);

- ricorrendo alla funzione

```
pts_ref=haltonseq(100000,2);
```

generiamo i primi 100000 punti di Halton bivariati `pts_ref` relativamente al quadrato $[0,1]^2$. Ricordiamo che essi hanno coordinate comprese tra 0 e 1, pertanto devono essere riscalate attraverso opportune omotetie di coefficienti determinati dagli estremi della bounding box determinata al passo precedente, ottenendo cosí punti di Halton `H0` nella bounding box del dominio di integrazione;

- mediante le assegnazioni

```
inside1=indomain(H0,domain1);
inside2=indomain(H0,domain2);
```

si determinano quali punti di Halton appartengono al dominio `d1` (nel nostro caso un disco) e al dominio `d2` (nel nostro caso un'ellisse); di seguito mediante operazioni insiemistiche si ottengono quali punti `pts_bb` tra quelli in `H0` appartengono a Ω;

- nota l'area $A_{\texttt{bb}}$ della bounding box `bb` di Ω e al numero M dei primi $N = 100000$ punti Halton che appartengono a Ω, ricaviamo che l'area A_Ω di Ω approssimandola con $\frac{M}{N} A_{\texttt{bb}}$;

- dal punto precedente ricaviamo che la formula QMC avrá nodi `pts_bb` e pesi tutti uguali e pari a

$$\frac{A_{\texttt{bb}}}{N} \approx \frac{A_\Omega}{M};$$

- infine risulta sufficiente chiamare

```
tol=10^(-14);
[nodes,weights]=cqmc(deg,pts_bb,tol);
```

per ottenere i nodi `nodes` e i relativi pesi `weights` della formula compressa CQMC, avente grado di precisione `deg` (rispetto alla misura discreta definita da QMC) e tolleranza `tol` sui momenti (ad esempio `tol`) pari a 10^{-14}.

Qualora la variabile `tol` non sia dichiarata, di default le verrà assegnato il valore 10^{-10}.

Quali domini prenderemo in considerazione l'unione, l'intersezione e la differenza del disco `d1` e dell'ellisse `d2` sopra definiti.

Inoltre sono stati considerati i primi 10^5 punti della sequenza di Halton scalata nella bounding box contenente l'insieme risultante dalla operazione insiemistica tra i domini precedentemente citati ed effettuata la compressione suggerita in [3], con gradi di precisione 5, 10, 15, 20.

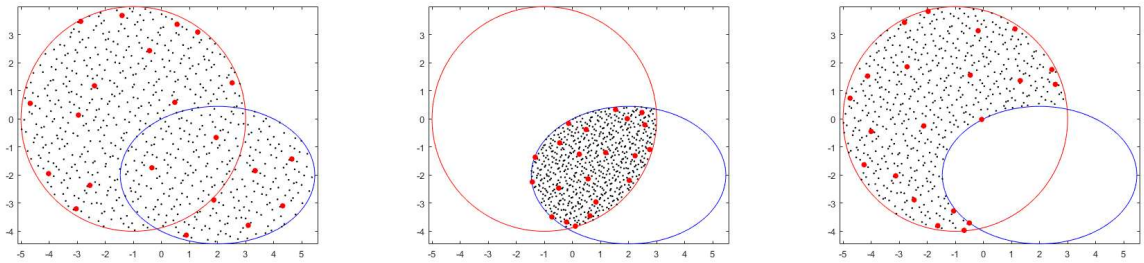
I risultati ottenuti sono in linea con quelli esposti in [3]. In particolar modo si vede come al crescere del grado serve tipicamente un numero maggiore di iterazioni all'interno dell'algoritmo di compressione utilizzato per determinare la formula CQMC.

Risulta importante sottolineare che i punti di Halton ottenuti mediante la routine `haltonseq`, propongono sempre la stessa sequenza essendo punti deterministici e quindi i risultati sui momenti nella riga relativa a *mom res CQMC* sono indipendenti dall'esperimento.

Come ci si aspetta, al crescere del grado il tempo richiesto dalla compressione passa da una frazione di quello utilizzato in QMC (nei nostri test per gradi di precisione fino a 10) all'ordine di alcuni secondi (ad esempio a grado 15 e 20).

In tutti i test esposti, l'algoritmo che calcola la formula CQMC comunque calcola sempre la formula a pesi positivi richiesta con un errore sui momenti dell'ordine di 10^{-15}.

Come previsto dal teorema di Tchakaloff, il numero di punti *card CQMC* risulta non superiore alla dimensione dello spazio dei polinomi di grado totale pari e inferiore al grado di precisione d, ovvero $(d+1)(d+2)/2$ che sarà 21 a grado 5, 66 a grado 10, 136 a grado 15 e 231 a grado 20. In realtà in tutti i casi in analisi si ha che *card CQMC* coincide proprio con questi valori. Questo si può già vedere nelle Tabelle 3.1, 3.2 e 3.3 presentate qua di seguito.



(a) Unione di `d1` e `d2`. (b) Intersezione di `d1` e `d2`. (c) Differenza di `d1` e `d2`.

Figura 3.1: 1000 punti a bassa discrepanza (in nero) e 21 CQMC (in rosso) con grado di precisione 5 relativi all'unione (3.1a), intersezione (3.1b) e differenza (3.1c) tra il disco `d1` e l'ellisse `d2`

deg	5	10	15	20
card QMC	100000			
card QMC in	71440			
card CQMC compr ratio	21 3.40e+0.3	66 1.08e+03	136 5.25e+02	231 3.09e+02
mom res CQMC iter 1 iter 2 iter 3	6.0e-16	9.3e-16	1.0e-02 1.9e-15	4.6e-01 4.0e-02 3.9e-15

Tabella 3.1: Dati relativi a QMC e CQMC nel caso del dominio di integrazione Ω ottenuto come unione tra il disco `d1` e l'ellisse `d2`

deg	5	10	15	20
card QMC	100000			
card QMC in	69099			
card CQMC compr ratio	21 3.29e+0.3	66 1.05e+03	136 5.08e+02	231 2.99e+02
mom res CQMC iter 1 iter 2	3.9e-16	7.7e-16	1.8e+308 1.7e-15	1.8e+308 3.6e-15

Tabella 3.2: Dati relativi a QMC e CQMC nel caso del dominio di integrazione Ω ottenuto come intersezione tra il disco `d1` e l'ellisse `d2`

deg	5	10	15	20
card QMC	100000			
card QMC in	57093			
card CQMC compr ratio	21 2.72e+0.3	66 8.65e+02	136 4.20e+02	231 2.47e+02
mom res CQMC iter 1 iter 2 iter 3 iter 4	 3.8e-16	 1.7e-15	 5.3e-01 1.2e-02 2.3e-15	 1.6e+00 6.7e-01 1.8e+308 3.7e-15

Tabella 3.3: Dati relativi a QMC e CQMC nel caso del dominio di integrazione Ω ottenuto come differenza tra il disco `d1` e l'ellisse `d2`

3.2.2 Esperimenti su un dominio definito da un ellisse e un poligono

In questo secondo esempio, i cui risultati sono visibili in Tabella 3.4, consideriamo un ellisse centrato in (-0.5,-1) e semiassi orizzontale e verticale rispettivamente pari a 1 e 0.5 e un dominio a forma di lettera N (si veda Figura 3.2). Di seguito, si riportano le porzioni di codici relativi alla generazione di suddetti insiemi (in successione si riportano quelli in cui si sono usate le struct e le classi):

```
d1 = define_domain('ellipse',[-0.5 -1],1,0.5);
d2 = define_domain('polygon',P);
```

```
d1 = ellipse([-0.5 -1],1,0.5);
d2 = polygon(P);
```

Per ottenere la formula CQMC si procede di seguito come indicato nell'esempio precedente.

Si noti che `P` è una matrice avente dimensione $N \times 2$ con N pari al numero di vertici del bordo che definisce il poligono a forma di lettera N. In aggiunta, `P` ha per righe le coordinate dei vertici (che ricordiamo essere ordinati in senso antiorario).

Il dominio di integrazione Ω consiste nell'intersezione tra l'ellisse `d1` e il poligono `d2` definiti precedentemente. Si sottolinea che il dominio risultante è non connesso. Nella bounding box corrispondente all'insieme appena citato, si sono presi in considerazione i primi 44563 punti della sequenza di Halton e partire da questi si è effettuata la compressione con gradi di precisione 5, 10, 15, 20.

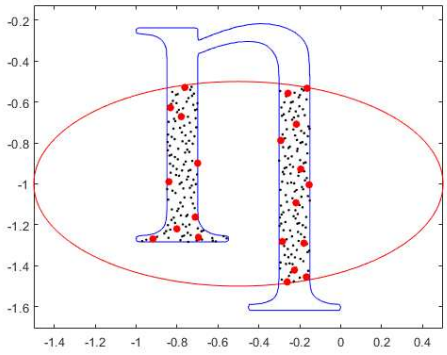


Figura 3.2: 1000 punti a bassa discrepanza (in nero) e 21 CQMC (in rosso) con grado di precisione 5 relativi all'intersezione tra l'ellisse `d1` e il poligono a forma di N definito da `d2`.

deg	5	10	15	20
card QMC	44563			
card QMC in	12195			
card CQMC compr ratio	21 5.81e+0.2	66 1.85e+02	136 8.97e+01	231 5.28e+01
mom res CQMC iter 1 iter 2 iter 3	 5.7e-16	 3.4e-01 1.1e-15	 3.0e-01 1.3e-15	 3.9e+00 6.3e-01 4.8e-15

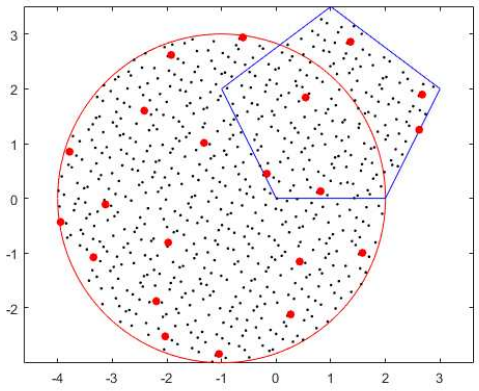
Tabella 3.4: Dati relativi all'intersezione disco e dominio a forma di lettera N.

3.2.3 Calcolo di alcuni integrali su un dominio dato dall'unione e dall'intersezione di un disco e un pentagono.

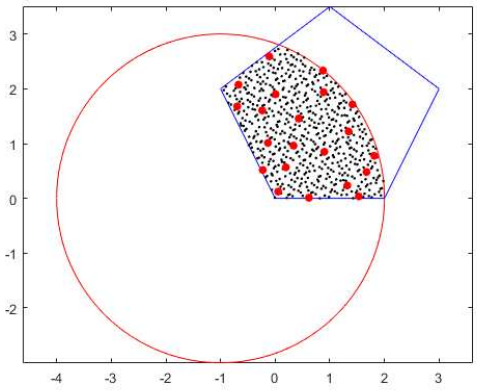
Come ultimo esempio, consideriamo dei domini che possono essere di interesse per i metodi di tipo *virtual elements* e prendiamo in esame un disco centrato in $(-1, 0)$ e raggio 3 e un pentagono regolare con vertici in $(0, 0), (2, 0), (3, 2), (1, 3.5), (-1, 2)$. Di seguito, si riportano le porzioni di codici relativi alla generazione di suddetti insiemi (in successione si riportano quelli in cui si sono usate le struct e le classi):

```
d1 = define_domain('disk',[-1 0], 3);
d2 = define_domain('polygon',[0 0; 2 0; 3 2; 1 3.5; -1 2]);
```

```
d1 = disk([-1 0], 3);
d2 = polygon([0 0; 2 0; 3 2; 1 3.5; -1 2]);
```



(a) Unione di `d1` e `d2`.



(b) Intersezione di `d1` e `d2`.

Figura 3.3: 1000 punti a bassa discrepanza (in nero) e 21 CQMC (in rosso) con grado di precisione 5 relativi all'unione (3.3a) e all'intersezione (3.3b) tra il disco `d1` e il pentagono regolare `d2`.

In questi domini vogliamo valutare gli errori assoluti commessi tra le formule QMC e CQMC rispetto all'integrale esatto delle seguenti funzioni test:

$$f_1(P) = \exp(-\|P - P_0\|_2^2) \tag{3.1}$$
$$f_2(P) = \|P - P_0\|_2 \tag{3.2}$$
$$f_3(P) = \|P - P_0\|_2^3 \tag{3.3}$$

dove abbiamo denotato con $P = (x, y)$, P_0 un punto interno ad entrambi i domini ovvero $P_0 = (1, 1)$ e $\|\cdot\|_2$ la distanza euclidea in $\mathbb{R}^2$.

Si evidenzia che la funzione f_1 è analitica mentre f_2 e f_3 pur essendo continue sono poco regolari.

Quale primo esempio consideriamo l'unione dei sottodomini `d1` e `d2`.

Intendendo utilizzare 10^5 punti di Halton, eseguiremo dapprima le chiamate

```
bb=boundingbox_operation(d1,d2,'U');
pts_ref=haltonseq(10^5,2);
```

in cui calcoliamo in sequenza la bounding box `bb` relativa all'unione del disco `d1` e del pentagono `d2` precedentemente inizializzati e i 100000 punti di Halton `pts_ref` nel quadrato $[0, 1]^2$ (che poi verranno opportunamente riscalati nella bounding box `bb` precedentemente calcolata).

Per la determinazione delle formule QMC e CQMC, i codici sono analoghi a quelli dell'esempio con il disco e l'ellisse.

Infine per la valutazione dell'integrale `I`, una volta noti i vettori colonna relativi alla valutazione `fx` della funzione nei nodi e pesi `w`, basterá effettuare l'assegnazione `I=w'*fx`.

Usando 10^6 punti di Halton nelle suddette bounding box e calcolando l'approssimazione degli integrali con QMC, si ottiene una migliore approssimazione degli integrali esatti

delle funzioni test. Per cui, utilizzeremo come valore di riferimento per calcolare gli errori questa migliore approssimazione. In particolare, denotati con $L_U(f_i)$ tali quantità relative rispettivamente all'unione di `d1` e `d2`, con $i = 1, 2, 3$, si sono ricavati i seguenti risultati:

$$L_U(f_1) = 3.050244543482617$$
$$L_U(f_2) = 84.775404124752910$$
$$L_U(f_3) = 1022.871151629675$$

Nella tabella 3.5

- riportiamo come nelle sottosezioni precedenti le proprietà del metodo CQMC, rispetto QMC, citando in particolare la cardinalità *card CQMC* da paragonare a quella di QMC ovvero *QMC in* e il rapporto di compressione *compr ratio*; come nei casi precedenti si è sempre raggiunta lo scopo di ottenere una formula il cui errore *mom res CQMC* è al di sotto della tolleranza richiesta di 10^{-14}, con un numero di iterazioni dell'algoritmo proposto in [3] crescente col grado di precisione *deg*;

- mostriamo gli errori assoluti tra la formula QMC e l'integrale di riferimento delle funzioni test rispettivamente sull'unione tra il disco `d1` e il pentagono regolare `d2`;

Nella tabella 3.6 forniamo una approssimazione degli errori assoluti tra la formula QMC e l'integrale di riferimento delle funzioni test sull'unione tra il disco `d1` e il pentagono regolare `d2`, mentre nella tabella 3.7 indichiamo gli errori assoluti tra la formula CQMC e una approssimazione dell'integrale esatto con integrande le funzioni test f_1, f_2, f_3 al variare del grado dello spazio di precisione.

Ovviamente gli errori trovati non dipendono dalla struttura dati usata per modellizzare i domini presi in considerazione in quest'esempio.

deg	5	10	15	20
card QMC	100000			
card QMC in	69854			
card CQMC compr ratio	21 3.33e+0.3	66 1.06e+03	136 5.14e+02	231 3.02e+02
mom res CQMC iter 1 iter 2	 1.0e-15	 5.6e-16	 1.8e+308 1.9e-15	 2.3e-01 2.2e-15

Tabella 3.5: Dati relativi a QMC e CQMC nel caso del dominio di integrazione Ω ottenuto come unione tra il disco `d1` e il pentagono regolare `d2`

deg	err.ass.
$E(f_1)$	5.73e-04
$E(f_2)$	2.43e-02
$E(f_3)$	4.05e-01

Tabella 3.6: Errori assoluti *err.ass.* tra la formula QMC e una approssimazione dell'integrale esatto, nel caso le integrande siano funzioni test f_1, f_2, f_3 nel caso dell'unione di `d1` e `d2`.

deg	5	10	15	20
$E^*(f_1)$	5.52e-01	2.87e-02	2.00e-03	4.91e-04
$E^*(f_2)$	2.19e-01	1.18e-01	4.34e-02	4.79e-02
$E^*(f_3)$	1.45e+01	3.68e-01	4.03e-01	4.01e-01

Tabella 3.7: Errori assoluti tra la formula CQMC e una approssimazione dell'integrale esatto con integrande le funzioni test f_1, f_2, f_3 nel caso dell'unione di `d1` e `d2`.

Da un confronto tra le tabelle 3.6 e 3.7, si vede immediatamente che

- per la funzione f_1 risulta necessario che la compressione raggiunga grado di precisione 20 affinché QMC e CQMC compiano errori paragonabili rispetto all'integrale esatto;
- per la funzione f_2 risulta necessario che la compressione raggiunga grado di precisione 15 affinché QMC e CQMC compiano errori paragonabili rispetto all'integrale esatto;
- per la funzione f_3 risulta necessario che la compressione raggiunga grado di precisione 10 affinché QMC e CQMC compiano errori paragonabili rispetto all'integrale esatto.

Quale ulteriore esempio, consideriamo il caso del dominio di integrazione Ω ottenuto intersecando il disco `d1` col pentagono `d2`.

Specializzando il codice al caso dell'intersezione, definiamo la bounding box `bb` e 100000 punti di Halton `pts_ref` nel quadrato $[0,1]^2$.

```
bb=boundingbox_operation(d1,d2,'I');
pts_ref=haltonseq(10^5,2);
```

In seguito mappiamo `pts_ref` dal quadrato $[0,1]^2$ alla bounding box `bb`, determiniamo mediante la funzione `indomain` quali sono all'interno di `d1` e `d2` e quindi facilmente quelli nell'intersezione tra `d1` e `d2`. Quindi dall'applicazione di `cqmc` con tolleranza 10^{-14} e la valutazione dell'integrale `I`, una volta noti i vettori colonna relativi alla valutazione `fx`

della funzione integranda nei nodi e pesi `w`, concludiamo come in precedenza che `I=w'*fx` é una approssimazione dell'integrale richiesto.

Per ottenere il valore di riferimento per la generica integranda f_j, abbiamo definito 10^6 punti di Halton nella bounding box `bb` e con un processo analogo a quello mostrato precedentemente calcolato una migliore approssimazione dell'integrale esatto mediante QMC.

Otteniamo quali valori di riferimento $L_I(f_j)$, $j = 1, 2, 3$

$$L_I(f_1) = 2.308049752574254$$
$$L_I(f_2) = 5.579864721531544$$
$$L_I(f_3) = 9.684972364741302$$

deg	5	10	15	20
card QMC	100000			
card QMC in	61099			
card CQMC	21	66	136	231
compr ratio	2.91e+0.3	9.26e+02	4.49e+02	2.64e+02
mom res CQMC				
iter 1	3.1e-16	1.0e-15	1.9e-15	2.2e-01
iter 2				2.6e-15

Tabella 3.8: Dati relativi a QMC e CQMC nel caso del dominio di integrazione Ω ottenuto come intersezione tra il disco `d1` e il pentagono regolare `d2`.

deg	err.ass.
$E(f_1)$	4.98e-04
$E(f_2)$	1.60e-03
$E(f_3)$	3.20e-03

Tabella 3.9: Errori assoluti *err.ass.* tra la formula QMC e una approssimazione dell'integrale esatto, nel caso le integrande siano funzioni test f_1, f_2, f_3.

deg	5	10	15	20
$E^*(f_1)$	3.20e-03	5.26e-04	4.99e-04	4.98e-04
$E^*(f_2)$	4.21e-02	1.42e-02	3.80e-03	2.50e-03
$E^*(f_3)$	1.47e-02	3.90e-03	3.60e-03	3.20e-03

Tabella 3.10: Errori assoluti tra la formula CQMC e una approssimazione dell'integrale esatto con integrande le funzioni test f_1, f_2, f_3, nel caso del dominio di integrazione Ω ottenuto come intersezione di `d1` e `d2`.

Da un confronto tra le tabelle 3.9 e 3.10, si vede immediatamente che

- per la funzione f_1 risulta necessario che la compressione raggiunga grado di precisione 10 affinché QMC e CQMC compiano errori paragonabili rispetto all'integrale esatto;
- per la funzione f_2 risulta necessario che la compressione raggiunga grado di precisione 15 affinché QMC e CQMC compiano errori paragonabili rispetto all'integrale esatto;
- per la funzione f_3 risulta necessario che la compressione raggiunga grado di precisione 10 affinché QMC e CQMC compiano errori paragonabili rispetto all'integrale esatto.

Mediante questi esempi abbiamo quindi illustrato che comprimendo la formula iniziale mediante CQMC si possono ottenere comunque risultati prossimi a quelli di QMC, col vantaggio di utilizzare considerevolmente molti meno nodi.

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 ed eventuali algoritmi e funzioni di supporto sono disponibili open-source al link:

https://github.com/mattiasantoro1/Codici_tesi

Di seguito si riportano le funzioni relative alla gestione dei domini bivariati con l'ausilio delle struct e presentate nella sottosezione 3.1.1.

```
function domain = define_domain(type,varargin)
% FUNCTION NAME:
%   define_domain.
%
% DESCRIPTION:
%   define a bivariate domain: disk, ellipse or polygon.
%
% INPUT:
%   type - (string) name of the domain. Up to now there are
   three options:
%   'disk','ellipse','polygon' if respectively a disk,
   ellipse, polygon is
%   needed to be defined;
%   varargin - (cell) set of parameters useful to define
   domain.
%
% OUTPUT:
%   domain - (struct) bivariate domain containing all the
   variables useful
%   for its definition and its bounding box.
%
% AUTHOR: M.Santoro.
% LAST UPDATE: 02/13/2024.
```

```
if nargin == 0
    return
end

switch lower(type)
    case 'disk'
        % DISK
        domain.name = 'disk';
        if nargin == 1
            % DEFAULT DISK
            domain.center = [0 0];
            domain.radius = 1;
        else
            if nargin == 3 && length(varargin{1}) == 2 &&
                isnumeric(varargin{1}) ...
                    && isnumeric(varargin{2})
                % USING ARGUMENTS TO MODEL A DISK
                domain.center = varargin{1};
                domain.radius = varargin{2};
            else
                return
            end
        end
        % DEFINING BOUNDING BOX
        xlimit = [domain.center(1)-domain.radius domain.
           center(1)+domain.radius];
        ylimit = [domain.center(2)-domain.radius domain.
           center(2)+domain.radius];
        domain.bounding_box = polyshape([xlimit(1) xlimit
           (2) xlimit(2) xlimit(1)],[ylimit(1) ylimit(1)
           ylimit(2) ylimit(2)]);

    case 'ellipse'
        % ELLIPSE
        domain.name = 'ellipse';
        if nargin == 1
            domain.center = [0 0];
            % DEFAULT ELLIPSE
            domain.a = 2;
            domain.b = 1;

        else
            if nargin == 4 && length(varargin{1}) == 2 &&
```

```
                isnumeric(varargin{1}) ...
                    && isnumeric(varargin{2}) &&
                       isnumeric(varargin{3})
                % USING ARGUMENTS TO MODEL AN ELLIPSE
                domain.center = varargin{1};
                domain.a = varargin{2};
                domain.b = varargin{3};
            else
                return
            end
        end
        % DEFINING BOUNDING BOX
        xlimit = [domain.center(1)-domain.a domain.center
           (1)+domain.a];
        ylimit = [domain.center(2)-domain.b domain.center
           (2)+domain.b];
        domain.bounding_box = polyshape([xlimit(1) xlimit
           (2) xlimit(2) xlimit(1)],[ylimit(1) ylimit(1)
           ylimit(2) ylimit(2)]);

    case 'polygon'
        % POLYGON
        domain.name = 'polygon';
        if nargin == 1
            % DEFAULT POLYGON
            domain.vertices = [0 0; 1 0; 1 1; 0 1];

        else
            s = size(varargin{1});
            if nargin == 2 && isnumeric(varargin{1}) && s
               (1) >= 3 && s(2) == 2
                % USING ARGUMENTS TO MODEL A POLYGON
                domain.vertices = varargin{1};
            else
                return
            end
        end
        % DEFINING BOUNDING BOX
        xlimit = [min(domain.vertices(:,1)) max(domain.
           vertices(:,1))];
        ylimit = [min(domain.vertices(:,2)) max(domain.
           vertices(:,2))];
        domain.bounding_box = polyshape([xlimit(1) xlimit
           (2) xlimit(2) xlimit(1)],[ylimit(1) ylimit(1)
```

```
                ylimit(2) ylimit(2)]);

    otherwise
        return
    end
end
```

```
function b = boundingbox_operation(domain1,domain2,operation)
% FUNCTION NAME:
%   boundingbox_operation.
%
% DESCRIPTION:
%   This function performs the given set operation between
   two domains
%   (modelled using define_domain routine) and compute the
   bounding
%   box that contains the final domain.
%
% INPUT:
%   domain1 - (struct) first domain;
%   domain2 - (struct) second domain;
%   operation - (string) set operation we want to perform
%   'U': union, 'I':intersection, 'D': difference.
%
% OUTPUT:
%   b - (polyshape) polyshape object that has the vertices of
    the resulting
%   bounding box.
%
% AUTHOR: M.Santoro.
% LAST UPDATE: 02/13/2024.

    % DETERMINING LIMITS OF FIRST DOMAIN BOUNDING BOX
    a1 = min(domain1.bounding_box.Vertices(:,1)); b1 = max(
       domain1.bounding_box.Vertices(:,1));
    c1 = min(domain1.bounding_box.Vertices(:,2)); d1 = max(
       domain1.bounding_box.Vertices(:,2));
    % DETERMINING LIMITS OF SECONDO DOMAIN BOUNDING BOX
    a2 = min(domain2.bounding_box.Vertices(:,1)); b2 = max(
       domain2.bounding_box.Vertices(:,1));
    c2 = min(domain2.bounding_box.Vertices(:,2)); d2 = max(
       domain2.bounding_box.Vertices(:,2));
    % DETERMINING LIMITS OF FINAL BOUNDING BOX
    switch operation
```

```
        case 'U'
            % UNION
            xlimit = [min(a1,a2) max(b1,b2)];
            ylimit = [min(c1,c2) max(d1,d2)];
        case 'I'
            % INTERSECTION
            xlimit = [max(a1,a2) min(b1,b2)];
            ylimit = [max(c1,c2) min(d1,d2)];
        case 'D'
            % DIFFERENCE
            xlimit = [a1 b1];
            ylimit = [c1 d1];
    end
    % COMPUTING FINAL BOUNDING BOX
    b = polyshape([xlimit(1) xlimit(2) xlimit(2) xlimit(1)],[
       ylimit(1) ylimit(1) ylimit(2) ylimit(2)]);
end
```

```
function inside = indomain(points,domain)
% FUNCTION NAME:
%   indomain.
%
% DESCRIPTION:
%   this function determines if some points are in a domain.
%
% INPUT:
%   points: 2xN matrix containing the points we want to test;
%   domain - (struct) set of variables that parameterizes the
    bivariate
%   domain (modelled using define_domain routine).
%
% OUTPUT:
%   inside: logical array that has as many entry as the
   tested point. In particular:
%       0: point is out of the domain, 1: point is in the
   domain.
%
% AUTHOR: M.Santoro.
% LAST UPDATE: 02/14/2024

% PREALLOCATING IN-DOMAIN ARRAY
inside = [];
switch domain.name
    case 'disk'
```

```
        inside = ((points(1,:) - domain.center(1)).^2 + (
           points(2,:) - domain.center(2)).^2 < domain.radius
            ^2);
    case 'ellipse'
        inside = ((points(1,:) - domain.center(1)).^2/domain.
           a ^2 + (points(2,:) - domain.center(2)).^2/domain.
           b ^2 < 1);
    case 'polygon'
        polygon = polyshape(domain.vertices(:,1),domain.
           vertices(:,2));
        inside = isinterior(polygon,points(1,:),points(2,:))
           ';
end
```

```
function [x,y] = plotdomain(domain)
% FUNCTION NAME:
%   plotdomain.
%
% DESCRIPTION:
%   This function provides some points that belong to the
   boundary of a
%   given domain.
%
% INPUT:
%   domain - (struct) bivariate domain modelled using
   define_domain
%   routine.
%
% OUTPUT:
%   x - (array) x-coordinates points;
%   y - (array) y-coordinates points.
%
% AUTHOR: M.Santoro.
% LAST UPDATE: 02/10/2024.

    % PREALLOCATING BOUNDARY POINTS COORDINATES
    x = []; y = [];
    switch domain.name
        case 'disk'
            t=-pi:0.01:pi;
            x = domain.center(1)+domain.radius*cos(t);
            y = domain.center(2)+domain.radius*sin(t);
        case 'ellipse'
            t=-pi:0.01:pi;
```

```
            x = domain.center(1)+domain.a*cos(t);
            y = domain.center(2)+domain.b*sin(t);
        case 'polygon'
            t=0:0.1:1;
            s = size(domain.vertices);
            for i = 1:1:s(1)-1
                x = [x t*domain.vertices(i+1,1)+(1-t)*domain.
                   vertices(i,1)];
                y = [y t*domain.vertices(i+1,2)+(1-t)*domain.
                   vertices(i,2)];
            end
            x = [x t*domain.vertices(1,1)+(1-t)*domain.
               vertices(i+1,1)];
            y = [y t*domain.vertices(1,2)+(1-t)*domain.
               vertices(i+1,2)];
    end
end
```

Le prossime funzioni sono relative, invece, alla gestione dei domini bivariati con l'utilizzo delle classi. Si ricorda che esse sono state presentate nella sottosezione 3.1.2.

```
classdef disk
% This class is used to model a bivariate disk.
%
% PROPERTIES:
%   center - (array) 1x2 float array containing in sequence x
    and y
%   coordinates of the disk center;
%   radius - (float) disk radius;
%   bounding_box - (polyshape) polyshape instance that
   represents the
%   bounding box of a disk.
%
% METHODS (documentation can be found below)
%   disk: constructor;
%   in_domain: given a set of points to test, this function
   determines
%   which of them are in the disk or not;
%   plotdomain: given a disk instance, compute some points of
    its boundary.
%
% AUTHOR: M.Santoro.
% LAST UPDATE: 02/14/2024.
```

```
properties
    center;
    radius;
    bounding_box;
end

methods
    function obj = disk(c,r)
    % Construct an instance of this class. If c and r are
        set, the
    % constructor creates a disk instance with center in
       c and
    % radius r. If none of them are set, this method
       constructs a disk
    % instance with center in (0,0) and radius 1 (default
        case)
        switch nargin
            case 0
                % DEFAULT CASE
                obj.center = [0 0];
                obj.radius = 1;
            case 2
                if r > 0
                % Radius must be positive
                    obj.center = c;
                    obj.radius = r;
                else
                    return
                end
            otherwise
                return
        end
        % CREATE BOUNDING BOX OF DISK INSTANCE
        xlimit = [obj.center(1)-obj.radius obj.center(1)+
           obj.radius];
        ylimit = [obj.center(2)-obj.radius obj.center(2)+
           obj.radius];
        obj.bounding_box = polyshape([xlimit(1) xlimit(2)
            xlimit(2) xlimit(1)],[ylimit(1) ylimit(1)
           ylimit(2) ylimit(2)]);
    end

    function bool = in_domain(points,obj)
    % INPUT:
```

```
        %   points - (matrix) Nx2 matrix whose rows contain
           in sequence x
        %   and y coordinates of the test points.
        %
        % OUTPUT
        %   bool - (array) 1xN logical array. In particular:
        %        0: point is out the disk, 1: point is in the
           disk.
            bool = ((points(1,:) - obj.center(1)).^2 + (
               points(2,:) - obj.center(2)).^2 < obj.radius
               ^2);
        end

        function [x,y] = plotdomain(obj)
        % OUTPUT:
        %   x - (array) x-coordinates points.
        %   y - (array) y-coordinates points.
            t=-pi:0.01:pi;
            x = obj.center(1)+obj.radius*cos(t);
            y = obj.center(2)+obj.radius*sin(t);
        end
    end
end
```

```
classdef ellipse
% This class is used to model a bivariate ellipse.
%
% PROPERTIES:
%   center - (array) 1x2 float array containing in sequence x
    and y
%   coordinates of the ellipse center;
%   a - (float) ellipse x-semiaxis;
%   b - (float) ellipse y-semiaxis;
%   bounding_box - (polyshape) polyshape instance that
   represents the
%   bounding box of an ellipse.
%
% METHODS (documentation can be found below)
%   ellipse: constructor;
%   in_domain: given a set of points to test, this function
   determines
%   which of them are in the ellipse or not;
%   plotdomain: given an ellipse instance, compute some
   points of its
```

```
%   boundary.
%
% AUTHOR: M.Santoro.
% LAST UPDATE: 02/14/2024.

    properties
        center;
        a;
        b;
        bounding_box;
    end

    methods
        function obj = ellipse(c,a,b)
        % Construct an instance of this class. If c,a,b are
           set, the
        % constructor creates an ellipse instance with center
            in c and
        % x,y-semiaxis respectively a,b. If none of them are
           set, this
        % method constructs an ellipse instance with center
           in (0,0) and
        % x,y-semiaxis respectively 2,1 (default case)
            switch nargin
                case 0
                    obj.center = [0 0];
                    obj.a = 2;
                    obj.b = 1;
                case 3
                    % a,b must be positive
                    if a > 0 && b > 0
                        obj.center = c;
                        obj.a = a;
                        obj.b = b;
                    else
                        return
                    end
                otherwise
                    return
            end
            % CREATE BOUNDING BOX OF ELLIPSE INSTANCE
            xlimit = [obj.center(1)-obj.a obj.center(1)+obj.a
               ];
            ylimit = [obj.center(2)-obj.b obj.center(2)+obj.b
```

```
                ];
            obj.bounding_box = polyshape([xlimit(1) xlimit(2)
                xlimit(2) xlimit(1)],[ylimit(1) ylimit(1)
                ylimit(2) ylimit(2)]);
        end

        function bool = in_domain(points,obj)
        % INPUT:
        %   points - (matrix) Nx2 matrix whose rows contain
           in sequence x
        %   and y coordinates of the test points.
        %
        % OUTPUT
        %   bool - (array) 1xN logical array. In particular:
        %       0: point is out the ellipse, 1: point is in
           the ellipse.
            bool = ((points(1,:) - obj.center(1)).^2/obj.a ^2
                + (points(2,:) - obj.center(2)).^2/obj.b ^2 <
                1);
        end

        function [x,y] = plotdomain(obj)
        % OUTPUT:
        %   x - (array) x-coordinates points;
        %   y - (array) y-coordinates points.
            t=-pi:0.01:pi;
            x = obj.center(1)+obj.a*cos(t);
            y = obj.center(2)+obj.b*sin(t);
        end
    end
end
```

```
classdef polygon
% This class is used to model a plane polygon.
%
% PROPERTIES:
%   vertices - (matrix) Nx2 matrix containing
   counterclockwise in sequence
%   x and y coordinates of the polygon. Note that first and
   last entries
%   do not have to be the same;
%   bounding_box - (polyshape) polyshape instance that
   represents the
%   bounding box of a polygon.
```

```
%
% METHODS (documentation can be found below)
%   polygon: constructor;
%   in_domain: given a set of points to test, this function
   determines
%   which of them are in the polygon or not;
%   plotdomain: given a polygon instance, compute some points
    of its
%   boundary.
%
% AUTHOR: M.Santoro.
% LAST UPDATE: 02/14/2024.

    properties
        vertices
        bounding_box;
    end

    methods
        function obj = polygon(v)
        % Construct an instance of this class. If v is set,
           the
        % constructor creates a disk instance whose vertices
           are the
        % entries of v. Otherwise, this method constructs the
            unitary
        % square (default case)
            switch nargin
                case 0
                    obj.vertices = [0 0; 1 0; 1 1; 0 1];
                case 1
                    obj.vertices = v;
            end
            % CREATE BOUNDING BOX OF POLYGON INSTANCE
            xlimit = [min(obj.vertices(:,1)) max(obj.vertices
               (:,1))];
            ylimit = [min(obj.vertices(:,2)) max(obj.vertices
               (:,2))];
            obj.bounding_box = polyshape([xlimit(1) xlimit(2)
                xlimit(2) xlimit(1)],[ylimit(1) ylimit(1)
               ylimit(2) ylimit(2)]);
        end

        function bool = in_domain(points,obj)
```

```
        % INPUT:
        %   points - (matrix) Nx2 matrix whose rows contain
           in sequence x
        %   and y coordinates of the test points.
        %
        % OUTPUT
        %   bool - (array) 1xN logical array. In particular:
        %       0: point is out the polygon, 1: point is in
           the polygon.
            polygon = polyshape(obj.vertices(:,1),obj.
               vertices(:,2));
            bool = isinterior(polygon,points(1,:),points(2,:)
               )';
        end

        function [x,y] = plotdomain(obj)
        % OUTPUT:
        %   x - (array) x-coordinates points;
        %   y - (array) y-coordinates points.
            t=0:0.1:1;
            s = size(obj.vertices);
            x = []; y = [];
            for i = 1:1:s(1)-1
                x = [x t*obj.vertices(i+1,1)+(1-t)*obj.
                   vertices(i,1)];
                y = [y t*obj.vertices(i+1,2)+(1-t)*obj.
                   vertices(i,2)];
            end
            x = [x t*obj.vertices(1,1)+(1-t)*obj.vertices(i
               +1,1)];
            y = [y t*obj.vertices(1,2)+(1-t)*obj.vertices(i
               +1,2)];
        end
    end
end
```

```
function b = boundingbox_operation(class1,class2,operation)
% FUNCTION NAME:
%   boundingbox_operation.
%
% DESCRIPTION:
%   This function performs the given set operation between
   two domains
%   (modelled using disk, ellipse amd polygon classes) and
```

```
   compute the
%   final bounding box.
%
% INPUT:
%   class1 - first domain;
%   class2 - second domain;
%   operation - (string) set operation we want to perform
%   'U': union, 'I':intersection, 'D': difference.
%   NOTE: class1 and class2 are instances of either disk,
   ellipse or
%   polygon class.
%
% OUTPUT:
%   b - (polyshape) polyshape object that has the vertices of
    the resulting
%   bounding box.
%
    % FIRST INSTANCE BOUNDING BOX PARAMETERS
    a1 = min(class1.bounding_box.Vertices(:,1)); b1 = max(
       class1.bounding_box.Vertices(:,1));
    c1 = min(class1.bounding_box.Vertices(:,2)); d1 = max(
       class1.bounding_box.Vertices(:,2));
    % SECOND INSTANCE BOUNDING BOX PARAMETERS
    a2 = min(class2.bounding_box.Vertices(:,1)); b2 = max(
       class2.bounding_box.Vertices(:,1));
    c2 = min(class2.bounding_box.Vertices(:,2)); d2 = max(
       class2.bounding_box.Vertices(:,2));
    % FINAL BOUNDING BOX PARAMETERS
    switch operation
        case 'U'
            % UNION
            xlimit = [min(a1,a2) max(b1,b2)];
            ylimit = [min(c1,c2) max(d1,d2)];
        case 'I'
            % INTERSECTION
            xlimit = [max(a1,a2) min(b1,b2)];
            ylimit = [max(c1,c2) min(d1,d2)];
        case 'D'
            % DIFFERENCE
            xlimit = [a1 b1];
            ylimit = [c1 d1];
    end
    % COMPUTING FINAL BOUNDING BOX
    b = polyshape([xlimit(1) xlimit(2) xlimit(2) xlimit(1)],[
```

```
        ylimit(1) ylimit(1) ylimit(2) ylimit(2)]);
end
```

Bibliografia

[1] F. Caravenna, P. Dai Pra, Probabilità: un'introduzione attraverso modelli e applicazioni.

[2] Harald Niederreiter, Random Number Generation and Quasi-Monte Carlo Methods, SIAM, Philadelphia (1992).

[3] G. Elefante, A. Sommariva e M. Vianello, CQMC: an improved code for low-dimensional Compressed Quasi-MonteCarlo cubature, Dolomites Research Notes on Approximation, Volume 15, Issue 2, 2022, pp. 92-100.

[4] G. Elefante, A. Sommariva e M. Vianello, Tchakaloff-like compression of QMC volume and surface integration on union of balls, sottomesso.