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

Esperimenti numerici sulla esattezza della quadratura nell'iperinterpolazione

Relatore:
Prof. Alvise Sommariva

Laureanda:
Agnese Passi
Matricola: 2038860

Anno Accademico 2025/2026

24 aprile 2026

Introduzione

Nel recente articolo *On the role of Marcinkiewicz-Zygmund constants in polynomial approximation by orthogonal bases*, gli autori hanno definito un algoritmo per valutare le costanti di Marcinkiewicz-Zygmund di alcune formule di cubatura su classici domini multivariati come l'intervallo, il quadrato, il disco, il triangolo, la sfera.

La loro importanza è stata sottolineata in *On the quadrature exactness in hyperinterpolation.* L'iperinterpolazione classica di Sloan consiste nel ricondurre l'approssimazione di una funzione continua f a una proiezione ortogonale sullo spazio di polinomi $\mathbb{P}_n$ di grado al più n, espressa in una base ortonormale, e nel sostituire i coefficienti (definiti tramite integrali) con approssimazioni discrete ottenute mediante una formula di quadratura con grado di esattezza algebrica ADE=$2n$. Sorge naturale chiedersi cosa succeda qualora si utilizzi una formula con ADE $< 2n$. Nella pubblicazione sopra menzionata si stabiliscono stime dell'errore di approssimazione qualora la costante di Marcinkiewicz-Zygmund sia minore strettamente di 1, evidenziando i vantaggi di valori prossimi a 0.

In questo ambito, nel caso dell'intervallo e del quadrato, dopo il calcolo delle costanti di Marcinkiewicz-Zygmund, si sono evidenziati i limiti delle formule basate sulla quadratura gaussiana e di come invece si possano usare certe formule interpolatorie a m punti basate su Clenshaw-Curtis per l'iperinterpolazione a grado n con $m + 1 < 2n$.

Quale alternativa, in questo lavoro consideriamo regole di tipo Fejér che, come quelle di Clenshaw-Curtis, sono basate su nodi di *tipo* Chebyshev.

Dopo un capitolo introduttivo sull'iperinterpolazione e sul rilassamento dell'esattezza nella quadratura, descriviamo le formule di tipo Fejér e come esse possano essere calcolate numericamente.

Nella sezione numerica, consideriamo i test numerici nel caso delle regole di Fejér, sia nell'intervallo $[-1, 1]$ che nel quadrato $[-1, 1]^2$. Di seguito mostriamo come l'iperinterpolazione rilassata perda rapidamente di accuratezza nel caso in cui le costanti di Marcinkiewicz-Zygmund siano prossime a 1, mentre il metodo basato sui minimi quadrati mediante matrice di Gram sia meno soggetto a queste problematiche.

In appendice forniamo i codici open-source utilizzati negli esperimenti numerici.

Indice

Capitolo 1

Iperinterpolazione

Il termine *iperinterpolazione* è stato introdotto dal matematico Ian H. Sloan nel 1995 nell'articolo [4]. L'iperinterpolazione è un metodo di approssimazione polinomiale per funzioni continue definite su insiemi compatti o, più in generale, su varietà e regioni multivariate. L'idea centrale consiste nel ricondurre l'approssimazione ad una proiezione ortogonale di f sullo spazio di polinomi di grado al più n, espressa in una base ortonormale, e nel sostituire i coefficienti (definiti tramite integrali) con approssimazioni discrete ottenute mediante una formula di quadratura di grado sufficientemente alto. Di seguito verranno introdotte alcune definizioni preliminari fino ad arrivare alla definizione rigorosa.

1.1 Nozioni preliminari

Sia $\Omega \subset \mathbb{R}^s$ una regione limitata (oppure una varietà chiusa e regolare di dimensione inferiore, come ad esempio la sfera $\mathbb{S}^2 \subset \mathbb{R}^3$) dotata di una misura $d\omega$ di volume finito

$$\int_\Omega d\omega = V < \infty.$$

Indichiamo con $C(\Omega)$ lo spazio delle funzioni continue su Ω e con $L^2(\Omega)$ lo spazio delle funzioni quadrato-integrabili sempre su Ω. Si denoti con $\mathbb{P}_n(\Omega) \subset L^2(\Omega)$ lo spazio dei polinomi su Ω di grado al più n e la cui dimensione è $d = d_n = \dim(\mathbb{P}_n(\Omega))$, munito del prodotto interno

$$\langle v, z \rangle = \int_\Omega vz \, d\omega. \tag{1}$$

Definizione 1.1.1 (Base Ortogonale)**.** Una base $\{p_i\}_{i=1,\dots,d}$ di $\mathbb{P}_n$ si dice ortonormale se:

$$\langle p_i, p_j \rangle = \delta_{ij} \qquad 1 \le i, j \le d \tag{2}$$

con δ_{ij} delta di Kronecker.

Definizione 1.1.2 (Proiezione Ortogonale)**.** Sia $\mathbb{P}_n$ lo spazio dei polinomi di grado al più n e $\{p_1, \dots, p_d\} \in \mathbb{P}_n$ una sua base ortonormale. Data f una funzione, la sua proiezione

ortogonale $T_n : L^2(\Omega) \to \mathbb{P}_n(\Omega)$ che proietta la funzione $f \in L^2(\Omega)$ nello spazio dei polinomi di grado al più n, è definita (unicamente) da:

$$T_n f := \sum_{i=1}^{d} \langle f, p_i \rangle p_i \tag{3}$$

con $\langle f, p_i \rangle = \int_\Omega f\, p_i \, d\omega$ coefficiente i-esimo di Fourier.

Per definire l'iperinterpolante si deve considerare una formula di quadratura a m punti della forma

$$\sum_{j=1}^{m} w_j \, g(x_j) \approx \int_\Omega g \, d\omega, \tag{4}$$

dove gli $x_j \in \Omega$ sono i punti di quadratura e i $w_j > 0$ i pesi per $j = 1, 2, \ldots, m$, la quale è *esatta per tutti i polinomi di grado minore o uguale a* $2n$, cioè

$$\sum_{j=1}^{m} w_j \, g(x_j) = \int_\Omega g \, d\omega \qquad \forall\, g \in \mathbb{P}_{2n}. \tag{5}$$

Osservazione. Si parla di formula di cubatura quando l'obiettivo è quello di approssimare integrali doppi.

È utile definire il prodotto interno discreto su $C(\Omega)$ come

$$\langle v, z \rangle_m := \sum_{i=1}^{m} \omega_i v(x_i) z(x_i) \qquad \forall\, v, z \in C(\Omega) \tag{6}$$

Segue la definizione di $L_n f$, analoga a quella di $T_n f$, dove il prodotto interno (1) viene sostituito con il prodotto interno discreto (6).

Definizione 1.1.3 (Operatore di iperinterpolazione)**.** L'*operatore di iperinterpolazione* $L_n : C(\Omega) \to \mathbb{P}_n$ *è definito come*

$$L_n f = \sum_{i=1}^{d} \langle f, p_i \rangle_m \, p_i, \tag{7}$$

dove $\{p_i\}_{i=1}^{d}$ *è una base ortonormale di* $\mathbb{P}_n$ *in* Ω*, rispetto a una certa misura e*

$$\langle v, z \rangle_m = \sum_{j=1}^{m} w_j v(x_j) z(x_j).$$

L'iperinterpolazione può essere interpretata come una *versione discreta della proiezione ortogonale* da $C(\Omega)$ su $\mathbb{P}_n$ rispetto al prodotto interno discreto $L^2(\Omega)$. Inoltre L_n è invariante, come T_n, rispetto al cambiamento di base ortonormale di $\mathbb{P}_n$.
Sia $\{p_1, \ldots, p_d\}$ una base di $\mathbb{P}_n$, ortonormale rispetto al prodotto interno (1). Il seguente lemma dimostra che la base $\{p_1, \ldots, p_d\}$ è ortonormale anche rispetto al prodotto interno discreto (6), qualsiasi sia la funzione peso coinvolta ω.

Lemma 1.1.1. *Per ogni* $1 \leq i, j \leq d$

$$\langle p_i, p_j \rangle_m = \delta_{i,j} \tag{8}$$

dove $\delta_{i,j}$ *è il delta di Kronecker.*

Dimostrazione. Poiché $p_i p_j$ è un polinomio di grado $\leq 2n$, segue da (2) e (5) che:

$$\langle p_i, p_j \rangle_m = \langle p_i, p_j \rangle = \delta_{i,j}.$$

□

Viene definita ora una condizione necessaria sul numero di nodi affinché una formula di quadratura possa essere utilizzata per generare un'iperinterpolante.

Lemma 1.1.2. *Se una formula di quadratura è esatta per tutti i polinomi di grado* $\leq 2n$*, allora il numero di punti di quadratura* m *soddisfa* $m \geq d_n$*.*

Dimostrazione. Sia Q la matrice $d \times m$ con elementi $q_{jk} = \omega_k^{\frac{1}{2}} p_j(x_k)$. Allora (8) afferma che le righe di Q sono ortogonali, quindi linearmente indipendenti, e dunque il rango di Q è d. Quindi, poiché $rank(Q) \leq m$, segue $m \geq d$. □

Definizione 1.1.4. Una formula di quadratura con m punti, esatta per tutti i polinomi di grado $\leq 2n$, è detta minimale se $m = d_n$.

Il risultato seguente afferma che l'approssimazione $L_n f$ possiede la proprietà classica di interpolazione se e solo se la formula di quadratura è minimale.

Lemma 1.1.3. *La classica formula di interpolazione*

$$L_n f(x_k) = f(x_k) \qquad 1 \leq k \leq m \tag{9}$$

vale per ogni $f \in C(\Omega)$ *se e solo se la formula di quadratura è minimale.*

Dimostrazione. Se $m = d$, la matrice Q del Lemma 1.1.5 è quadrata e, per il Lemma 1.1.4, soddisfa $QQ^T = \mathbb{I}_d$, da cui si ottiene l'identità

$$\sum_{j=1}^{d} p_j(x_k) p_j(x_\ell) = \omega_k^{-1} \delta_{k\ell} \qquad 1 \leq k, \ell \leq m. \tag{10}$$

Da (7) segue subito (9). Viceversa, se (9) vale per ogni f, allora deve valere (10), che implica che Q è quadrata, quindi $m = d$. □

Osservazione. L'iperinterpolazione è perciò una generalizzazione della classica interpolazione polinomiale e si differenzia da essa perchè il numero di nodi di quadratura $m \geq d = dim(\mathbb{P}_n)$.

Infine, si osserva che $L_n f$ risulta *esatta per ogni polinomio* $p \in \mathbb{P}_n$, ossia $L_n p = p$. Questo teorema è detto teorema di proiezione.

Teorema 1.1.1 (Teorema di proiezione)**.** *Se $f \in \mathbb{P}_n(\Omega)$, allora $L_n f = f$.*

Dimostrazione. Dall'ipotesi $f \in \mathbb{P}_n(\Omega)$ si ha:

$$f = \sum_{i=1}^{d} a_i p_i$$

Per definizione di iperinterpolazione (7) si ha che:

$$L_n f = \sum_{i=1}^{d} \langle f, p_i \rangle_m \, p_i = \sum_{i=1}^{d} \left\langle \sum_{j=1}^{d} a_j p_j \right\rangle_m p_i = \sum_{i=1}^{d} \sum_{j=1}^{d} a_j \langle p_j, p_i \rangle_m p_i = \sum_{i=1}^{d} a_i p_i = f$$

□

Prima di passare al paragrafo successivo, definiamo alcune nozioni che ci porteranno ad enunciare il teorema che ci dà un limite superiore per la norma dell'approssimante e, dunque, una stima per essa.

Definizione 1.1.5 (Norma infinito)**.** Siano Ω e $f \in C(\Omega)$. La norma infinito è definita da:

$$\|f\|_\infty = \sup_{x \in \Omega} |f(x)| \tag{11}$$

Definizione 1.1.6 (Norma L^2)**.** Siano Ω un dominio, $f \in L^2(\Omega)$ e $d\omega$. La norma L^2 è definita da:

$$\|f\|_2 = \left(\int_\Omega |f|^2 d\omega \right)^{\frac{1}{2}} \tag{12}$$

Definizione 1.1.7 (Polinomio di miglior approssimazione)**.** Siano Ω un dominio e $f \in C(\Omega)$, $p^* \in \mathbb{P}_n$ si dice polinomio di miglior approssimazione di f in $\mathbb{P}_n$ se vale:

$$E_n(f) = \inf_{p \in \mathbb{P}_n} \|f - p\|_\infty = \|f - p^*\|_\infty \tag{13}$$

dove $E_n(f)$ è l'errore di miglior approssimazione uniforme di f.

Le proprietà fondamentali introdotte da Sloan in [4] dell'iperinterpolazione sono:

- **1. Linearità.** L'operatore di iperinterpolazione L_n è lineare, cioè:

$$L_n(\alpha f + \beta g) = \alpha L_n f + \beta L_n g, \qquad \forall f, g \in C(\Omega), \ \forall \alpha, \beta \in \mathbb{R}.$$

Questo deriva direttamente dalla definizione:

$$L_n f(x) = \sum_{i=1}^{N} (f, u_i)_{\mu_n} \, u_i(x),$$

poiché il prodotto scalare discreto

$$(f, g)_{\mu_n} = \sum_{j=1}^{M} w_j \, f(x_j) \, g(x_j)$$

è lineare in f.

- **2. Stabilità in norma L^2.** Sia

$$V = \int_\Omega d\omega$$

la misura totale del dominio. Allora l'iperinterpolante soddisfa la seguente stima di stabilità:

$$\|L_n f\|_2 \leq \sqrt{V} \, \|f\|_\infty.$$

Questa disuguaglianza, dimostrata da Sloan (1995), significa che L_n non amplifica l'errore: è un operatore limitato da una costante indipendente da n. In particolare, se la formula di cubatura ha pesi positivi e grado di esattezza almeno $2n$, allora la norma operatore di L_n in L^2 è controllata da $\sqrt{V}$.

- **3. Stima dell'errore.** Sia

$$E_n(f) = \inf_{p \in \mathbb{P}_n} \|f - p\|_\infty$$

il miglior errore di approssimazione polinomiale di grado $\leq n$. Allora vale la stima dell'errore di iperinterpolazione:

$$\|f - L_n f\|_2 \leq 2 \sqrt{V} \, E_n(f).$$

Ciò significa che $L_n f$ approssima f in norma L^2 con lo stesso ordine del miglior polinomio approssimante:

$$\|f - L_n f\|_2 = O(E_n(f)).$$

Quindi, se $E_n(f) \to 0$ (cioè se i polinomi sono densi in $C(\Omega)$), allora $L_n f \to f$ in L^2.

La stabilità in norma L^2 e la stima dell'errore sono introdotte da Sloan nel seguente teorema:

Teorema 1.1.2. *Data $f \in C(\Omega)$, sia $L_n f \in \mathbb{P}_n$ definita come in (7), dove i nodi di quadratura $x_j \in \Omega$ e i pesi $\omega_j > 0$ per $j = 1, 2, \ldots, m$ nel prodotto discreto soddisfano (5). Allora*

(1) $\|L_n f\|_2 \leq V^{1/2} \, \|f\|_\infty$,

(2) $\|L_n f - f\|_2 \leq 2V^{1/2} \, E_n(f)$,

(3) $\|L_n f - f\|_2 \to 0$ *per* $n \to \infty$, *ovvero* $\lim_{n\to\infty} \|L_n f - f\|_2 = 0$.

1.2 Rilassamento dell'esattezza nella quadratura

La costruzione dell'iperinterpolante di grado n richiede necessariamente l'utilizzo di una formula di quadratura a pesi positivi con grado di esattezza $2n$, avente nodi in Ω. A tal proposito, in [3], viene esaminato cosa succede se si rilassa il grado di esattezza da $2n$ a $n+k$.

La formula dell'operatore è sempre del tipo

$$L_n f = \sum_{i=1}^{d} \langle f, p_i \rangle_m \, p_i, \qquad \langle f, g \rangle_m = \sum_{j=1}^{m} w_j \, f(x_j) \, g(x_j),$$

dove $\{p_i\}$ è una base L^2-ortonormale di $\mathbb{P}_n$ e si utilizza una formula di quadratura a pesi positivi $\{(x_j, w_j)\}_{j=1}^{m}$,.

Nel caso inesatto per l'iperinterpolante di grado n si fa la seguente ipotesi:

Ipotesi 1.2.1 La formula di quadratura a m punti (4), con nodi $x_j \in \Omega$ e pesi $w_j > 0$ per $j = 1, 2, \ldots, m$, ha grado di esattezza $n+k$ con $0 < k \leq n$, dove $n, k \in \mathbb{N}$.

Da cui segue la definizione:

Definizione 1.2.1 (Iperinterpolazione con formula di quadratura a esattezza rilassata)**.** Sia $\langle \cdot, \cdot \rangle_m$ una formula di quadratura con m punti che soddisfa l' Ipotesi 1.2.1 e sia $\{p_i\}_{i=1}^{d_n} \subset P_n$ una base ortonormale di $\mathbb{P}_n$. Dato $f \in C(\Omega)$, l'iperinterpolante di grado n di f è definito come

$$L_n f = \sum_{i=1}^{d} \langle f, p_i \rangle_m \, p_i. \tag{14}$$

Le formule (7) e (14) essenzialmente coincidono, eccetto per il fatto che il grado di esattezza della quadratura è stato rilassato. Quindi (14) è anch'esso una versione discreta della proiezione ortogonale da $C(\Omega)$ su $\mathbb{P}_n$ rispetto al prodotto interno L_n^2 (1).

Denominiamo con L_n^S l'operatore iperinterpolante originale di Sloan e definiamo quali sono i benefici e i costi che comporta il rilassamento di grado. Una conseguenza immediata è certamente che diminuiscono i punti richiesti per la formula di quadratura. Infatti, sappiamo che una formula di quadratura con m punti e grado di esattezza $2n$ richiede $m \geq d$ punti di quadratura e si dice minimale se $m = d$. Ma rilassando l'esattezza a $n+k$ otteniamo il seguente teorema:

Teorema 1.2.1. *Il numero di punti di quadratura per l'iperinterpolazione* (14) *soddisfa*

$$m \geq \begin{cases} d_{\frac{n+k}{2}} = \dim \mathbb{P}_{\frac{n+k}{2}}, & \text{se } n+k \text{ è pari}, \\ d_{\frac{n+k+1}{2}} = \dim \mathbb{P}_{\frac{n+k+1}{2}}, & \text{se } n+k \text{ è dispari} \end{cases}$$

Si nota quindi che per le formule di quadratura minimali usate nella costruzione dell'iperinterpolante, il numero di punti di quadratura richiesto può essere ridotto in modo

considerevole da $d_n = dim\mathbb{P}_n$ a $\dim \mathbb{P}_{\frac{n+k}{2}}$ oppure a $\dim \mathbb{P}_{\frac{n+k+1}{2}}$, a seconda della parità di $n+k$. Inoltre, si possono considerare, per costruire iperinterpolanti in modo efficiente, anche tutte quelle formule poco pratiche che utilizzano un numero maggiore di nodi per ottenere il grado di esattezza $2n$.

Questo rilassamento, ovviamente, non è privo di costi. L'iperinterpolante originale (7) è una proiezione per $f \in \mathbb{P}_n$, cioè

$$L_n^S f = f \text{ per ogni } f \in \mathbb{P}_n.$$

Tuttavia, a causa della perdita di alcuni gradi di esattezza, questa proprietà si conserva solo per polinomi di grado al più k, come affermato nel seguente lemma.

Lemma 1.2.1. *Se $f \in \mathbb{P}_k$ allora l'operatore L_n definito nella Definizione 1.2.1 soddisfa*

$$L_n f = f.$$

Dimostrazione Per $f \in \mathbb{P}_k$, esso può essere espresso come $f = \sum_{i=1}^{d_k} a_i \, \phi_i$ con $a_i = \int_\Omega f \, \phi_i \, d\omega$, dove $d_k = \dim \mathbb{P}_k$. Il grado di esattezza $n+k$ garantisce che $\langle \phi_{\ell'}, \phi_\ell \rangle_m = \delta_{\ell\ell'}$ con $1 \le \ell' \le d_k,\ 1 \le \ell \le d_n$. Pertanto

$$L_n f = \sum_{\ell=1}^{d_n} \left(\sum_{\ell=1}^{d_k} a_{\ell'} \, \langle \phi_{\ell'}, \phi_\ell \rangle_m \right) \phi_\ell = \sum_{\ell=1}^{d_n} \left(\sum_{\ell'=1}^{d_k} a_{\ell'} \, \delta_{\ell\ell'} \right) p_\ell = \sum_{\ell=1}^{d_k} a_\ell \, p_\ell.$$

Da cui segue $L_n f = f$. □

Il Lemma 1.2.1 indica che il grado di esattezza $2n$ può essere rilassato almeno fino al $n+1$ in caso contrario, la proprietà di $L_n f = f$ per ogni $f \in \mathbb{P}_k$ non si mantiene per nessuno spazio polinomiale non banale. Inoltre, si potrebbe pensare che per l'iperinterpolazione a esattezza rilassata (14) valga $L_n f = f$ per $f \in \mathbb{P}_{\frac{n+k}{2}}$, deducendolo dal fatto, come si è visto, che per L_n^S con grado di esattezza $2n$ vale $L_n^S f = f$ per ogni $f \in \mathbb{P}_n$. Tuttavia, questo non è vero, perché nella dimostrazione del Lemma 1.2.1, il prodotto scalare discreto $\langle p_{\ell'}, p_\ell \rangle_m$ con grado di esattezza $n+k$ non è necessariamente il delta di Kronecker $\delta_{\ell\ell'}$ per $p_{\ell'} \in \mathbb{P}_{\frac{n+k}{2}}$ e $p_\ell \in \mathbb{P}_n$.

Un altro costo di questo rilassamento è che il tasso di convergenza di L_n^S viene rallentato da $E_n(f)$ a $E_k(f)$. H. Sloan in [4] dimostra le seguenti disuguaglianze:

$$\|L_n f\|_2 \le \sqrt{V} \, \|f\|_\infty. \tag{15}$$

e

$$\|f - L_n f\|_2 \le 2\sqrt{V} \, E_n(f). \tag{16}$$

dove $E_n(f)$ denota il miglior errore uniforme di approssimazione di f mediante un polinomio in $\mathbb{P}_n$, cioè:

$$E_n(f) = \inf_{p \in \mathbb{P}_n} \|f - p\|_\infty.$$

Il teorema che seguirà mostra come cambia la stima dell'errore quando il grado di esattezza è rilassato. Esso utilizza questa proprietà fondamentale.

Proprietà di Marcinkiewicz-Zygmund Si assume che esista $\eta \in [0,1)$ tale che

$$\left| \sum_{j=1}^{m} \omega_j \varphi(x_j)^2 - \int_\Omega \varphi^2 \, d\omega \right| \leq \eta \int_\Omega \varphi^2 \, d\omega, \qquad \forall \, \varphi \in \mathbb{P}_n. \tag{17}$$

Se $k = n$, cioè se il grado di esattezza non è rilassato, allora $\eta = 0$.

Teorema 1.2.2. *Dato $f \in C(\Omega)$, sia $L_n f \in \mathbb{P}_n$ definito da* (14)*, dove la formula di quadratura a m punti* (4) *soddisfa sia l'ipotesi 1.2.1 con $0 < k \leq n$ sia la proprietà di Marcinkiewicz-Zygmund* (17) *con $\eta \in [0,1)$. Allora vale*

$$\|L_n f\|_2 \leq \sqrt{V} \, \frac{1}{\sqrt{1-\eta}} \|f\|_\infty \tag{18}$$

e

$$\|L_n f - f\|_2 \leq \left(\frac{1}{\sqrt{1-\eta}} + 1 \right) \sqrt{V} \, E_k(f). \tag{19}$$

L'iperinterpolante $L_n f$ può non convergere a f quando $n \to \infty$ se k è fissato. Se k è inoltre positivamente correlato a n, allora

$$\|L_n f - f\|_2 \to 0 \qquad \text{per } n \to \infty. \tag{20}$$

Se $k = n$, cioè il grado di esattezza della quadratura non è rilassato, allora il risultato di stabilità (18), la stima dell'errore (19) e il risultato di convergenza (20) coincidono con quelli di L_n^S. Se $0 < k \leq n$, allora la stima dell'errore ora è controllata da $E_k(f)$ anziché da $E_n(f)$. Poiché $E_k(f) \geq E_n(f)$, se $k < n$, la stima (19) mette in evidenza l'effetto del rilassamento dell'esattezza della quadratura: è possibile utilizzare meno punti di quadratura rispetto all'iperinterpolazione originale, ma la stima dell'errore corrispondente risulta leggermente amplificata. Inoltre, se $k \leq 0$, cioè se il grado di esattezza della quadratura è rilassato a n o addirittura a un valore inferiore, allora il teorema 1.2.2 non fornisce alcuna informazione sulla convergenza.

Prima di dimostrare il Teorema 1.2.2 definiamo alcune nozioni utili in seguito.

L'iperinterpolante $L_n f$ può essere decomposto come:

$$L_n f := L_k f \, + \, (L_n - L_k) f, \tag{21}$$

dove $L_n - L_k : C(\Omega) \to \mathbb{P}_n$ è un operatore lineare che associa a $f \in C(\Omega)$

$$(L_n - L_k) \, f := \sum_{i=d_k+1}^{d_n} \langle f, p_i \rangle_m p_i \in \mathbb{P}_n.$$

Nella dimostrazione del Teorema 1.2.2 vengono trattate separatamente $L_k f$ e $(L_n - L_k) \, f$.

Per la prima componente, il grado di esattezza della quadratura $n + k \geq 2k$ porta a

$$\langle L_k f, L_k f \rangle = \langle L_k f, L_k f \rangle_m. \tag{22}$$

per la seconda componente, l'ortogonalità di $\{p_i\}$ implica

$$\langle (L_n - L_k)\, f, (L_n - L_k)\, f \rangle = \sum_{i=d_k+1}^{d_n} \langle f, p_i \rangle_m^2 = \langle f, (L_n - L_k)\, f \rangle_m. \tag{23}$$

Per dimostrare il teorema, si introduce prima il seguente Lemma che coinvolge $\langle L_k f, L_k f \rangle_m$ e $\langle f, (L_n - L_k)\, f \rangle_m$.

Lemma 1.2.2. *Assumiamo le ipotesi del Teorema 1.2.2. Sia $L_k : C(\Omega) \to \mathbb{P}_k$ l'operatore di iperinterpolazione di grado k, definito tramite una quadratura a m punti con grado di esattezza $n + k$. Allora:*

(a) $\langle f - L_k f,\ \phi \rangle_m = 0$ *e* $\langle f - L_n f,\ \phi \rangle_m = 0 \qquad \forall\, \phi \in P_k.$

(b) $\langle L_k f,\ L_k f \rangle_m + \langle f - L_k f,\ f - L_k f \rangle_m = \langle f,\ f \rangle_m.$

(c) $\langle L_k f,\ L_k f \rangle_m + \langle L_n f - L_k f,\ L_n f - L_k f \rangle_m = \langle L_n f,\ L_n f \rangle_m.$

(d) $\langle f - L_n f,\ f - L_n f \rangle_m + 2\langle f,\ L_n f - L_k f \rangle_m = \langle f - L_k f,\ f - L_k f \rangle_m + \langle L_n f - L_k f,\ L_n f - L_k f \rangle_m.$

Dimostrazione Teorema 1.2.2 Dalla decomposizione (21) si ha che

$$\begin{aligned} \|L_n f\|_2^2 &= \langle L_n f, L_n f \rangle \\ &= \langle L_k f + (L_n - L_k) f,\ L_k f + (L_n - L_k) f \rangle \\ &= \langle L_k f, L_k f \rangle + \langle (L_n - L_k) f,\ (L_n - L_k) f \rangle . \end{aligned}$$

dove l'ultimo passaggio vale perché $\langle L_k f, \langle (L_n - L_k)\, f \rangle = 0$, dal punto (d) del Lemma 1.2.2. Le osservazioni (22) e (23) portano quindi a

$$\|L_n f\|_2^2 = \langle L_k f, L_k f \rangle_m \ + \ \langle f, (L_n - L_k)\, f \rangle_m.$$

Per ricavare il risultato di stabilità (18) sommando le equazioni nei punti $(b), (c), (d)$ del Lemma 1.2.2 e svolgendo semplici calcoli, si ottiene

$$2\langle L_k f, L_k f \rangle_m + 2\langle f,\ (L_n - L_k) f \rangle_m + \langle f - L_n f,\ f - L_n f \rangle_m = \langle f, f \rangle_m + \langle L_n f, L_n f \rangle_m. \tag{24}$$

Poiché

$$\|L_n f\|_2^2 + \langle f - L_n f,\ f - L_n f \rangle_m + \sigma_{n,k,f} = \langle f, f \rangle_m,$$

con

$$\sigma_{n,k,f} = \langle L_n f - L_k f,\ L_n f - L_k f \rangle - \langle L_n f - L_k f,\ L_n f - L_k f \rangle_m, \tag{25}$$

rappresenta l'errore nel calcolare, tramite la formula di quadratura (4) con grado di esattezza $n+k$, l'integrale di $(L_nf - L_kf)^2$ su Ω. Usando quindi (25) e l'osservazione (23), si ha

$$\langle f,\, (L_n - L_k)f\rangle_m = \langle L_nf - L_kf,\ L_nf - L_kf\rangle = \langle L_nf - L_kf,\ L_nf - L_kf\rangle_m + \sigma_{n,k,f}.$$

Insieme al punto (c) del Lemma 1.2.2, segue che

$$\langle L_nf, L_nf\rangle_m = \langle L_kf, L_kf\rangle_m + \langle L_nf - L_kf,\ L_nf - L_kf\rangle_m = \langle L_kf, L_kf\rangle_m + \langle f,\, (L_n - L_k)f\rangle_m - \sigma_{n,k,f}.$$

Quindi, sostituendo la somma $(\langle L_kf, L_kf\rangle_m + \langle f,\, (L_n - L_k)f\rangle_m$ nel membro sinistro di (24) con $\langle L_nf, L_nf\rangle_m + \sigma_{n,k,f}$, otteniamo

$$\langle L_kf, L_kf\rangle_m + \langle f,\, (L_n - L_k)f\rangle_m + \sigma_{n,k,f} + \langle f - L_nf,\ f - L_nf\rangle_m = \langle f, f\rangle_m. \tag{26}$$

Dalla proprietà Marcinkiewicz-Zygmund (17)

$$|\sigma_{n,k,f}| \le \eta\, \langle L_nf - L_kf,\ L_nf - L_kf\rangle = \eta\, \langle f,\, (L_n - L_k)f\rangle_m.$$

Pertanto, insieme alla non negatività di $\langle f - L_nf,\ f - L_nf\rangle_m$, l'equazione (26) fornisce

$$\langle L_kf, L_kf\rangle_m + (1-\eta)\, \langle f,\, (L_n - L_k)f\rangle_m \le \langle f, f\rangle_m,$$

cioè

$$\langle f,\, (L_n - L_k)f\rangle_m \le \frac{1}{1-\eta}\Big(\langle f, f\rangle_m - \langle L_kf, L_kf\rangle_m\Big).$$

Ne segue che

$$\|L_nf\|_2^2 = \langle L_kf, L_kf\rangle_m + \langle f,\, (L_n - L_k)f\rangle_m \le \frac{1}{1-\eta}\langle f, f\rangle_m - \frac{\eta}{1-\eta}\langle L_kf, L_kf\rangle_m \le \frac{1}{1-\eta}\langle f, f\rangle_m,$$

e il risultato di stabilità (18) segue da

$$\langle f, f\rangle_m = \sum_{j=1}^{m} w_j f(x_j)^2 \le \sum_{j=1}^{m} w_j \|f\|_\infty^2 = V\, \|f\|_\infty^2.$$

La stima dell'errore (19) si ricava con un argomento standard. Per ogni $\phi \in \mathbb{P}_k$, dal Lemma 1.2.1 si ha

$$L_nf - f = L_n(f - \phi) - (f - \phi).$$

Usando la stabilità (18), otteniamo

$$\begin{aligned}\|L_n f - f\|_2 &= \|L_n(f-\phi) - (f-\phi)\|_2 \\ &\leq \|L_n(f-\phi)\|_2 + \|f-\phi\|_2 \\ &\leq \frac{V^{1/2}}{\sqrt{1-\eta}}\|f-\phi\|_\infty + V^{1/2}\|f-\phi\|_\infty \\ &= \sqrt{\Big(\frac{1}{1-\eta}+1\Big)}\, V^{1/2}\, \|f-\phi\|_\infty.\end{aligned}$$

Poiché ciò vale per ogni $\phi \in \mathbb{P}_k$, ne segue che

$$\|L_n f - f\|_2 \leq \sqrt{\Big(\frac{1}{1-\eta}+1\Big)}\, V^{1/2} \inf_{\phi\in\mathbb{P}_k} \|f-\phi\|_\infty = \sqrt{\Big(\frac{1}{1-\eta}+1\Big)}\, V^{1/2}\, E_k(f).$$

Se k è fissato, allora $E_k(f)$ è fissato, e ciò suggerisce che non si può concludere alcun risultato di convergenza di $L_n f$ quando $n \to \infty$. D'altra parte, se k è positivamente correlato con n, allora $E_k(f) \to 0$ e dunque $\|L_n f - f\|_2 \to 0$ quando $n \to \infty$. □

Alcuni studi recenti hanno permesso di estendere i risultati ottenuti a un caso di rilassamento totale, affrontato nel seguente articolo [2], dove viene eliminata l'ipotesi di esattezza della quadratura e si assume come ipotesi solo la disuguaglianza MZ (17). In questo caso si parla di "unfettered" hyperinterpolation, poiché non è più vincolata dall'ipotesi di esattezza delle formule di quadratura.

Capitolo 2

Ruolo delle costanti deboli di Marcinkiewicz-Zygmund

In questo capitolo analizziamo quando si può usare una formula di quadratura (campionamento con pesi) per fare una buona approssimazione polinomiale *tipo Fourier*, anche se la quadratura non è esatta fino a grado $2n$.

Nell'articolo [1] vengono calcolate numericamente le costanti L^2 di Marcinkiewicz–Zygmund (MZ) associate a formule di cubatura S su domini multivariati Ω, con particolare attenzione al loro ruolo nell'approssimazione polinomiale tramite basi ortogonali.

Nei paragrafi successivi saranno ripresi tutti i risultati affrontati nel capitolo precedente, considerando formule di quadratura rilassate e sarà studiata in maniera approfondita la proprietà di Marcinkiewicz–Zygmund (17).

2.1 Calcolo delle costanti deboli di Marcinkiewicz-Zygmund

La formula di cubatura con pesi positivi $\omega_i > 0$

$$S(f) = \sum_{i=1}^{d} w_i f(x_i) \approx I(f) = \int_{\Omega} f(x) d\mu. \tag{1}$$

gode della proprietà Marcinkiewicz–Zygmund (MZ) su Ω, rispetto a una misura μ, se esistono costanti $A > 0$ e $B > 0$ tali che

$$A\|p\|_{L^2}^2 \leq S(p^2) \leq B\|p\|_{L^2}^2 \qquad \forall p \in \mathbb{P}_n,\ \|p\|_{L^2}^2 = I(p^2). \tag{2}$$

In questa formulazione *debole* della disuguaglianza, le costanti A e B sono indipendenti da $p \in \mathbb{P}_n$ ma dipendono da n. Fissata una base $\mu-$ortonormale $\{p_j\}$ di $\mathbb{P}_n$, si introduce come in (17)

$$\eta = \eta(n) = \max\{|A-1|, |B-1|\} < 1 \tag{3}$$

Se η non è troppo vicina a 1, allora grazie alla disuguaglianza MZ,

$$|S(p^2) - I(p^2)| \leq \eta I(p^2) \qquad \forall p \in \mathbb{P}_n, \tag{4}$$

e per quanto visto l'operatore di iperinterpolazione a esattezza rilassata

$$L_n f = \sum_{j=1}^{d_n} S(f\, p_j) p_j \qquad d_n = \dim(\mathbb{P}_n) \tag{5}$$

può essere usato per approssimare $f \in C(\Omega)$, anche se la formula non integra esattamente in $\mathbb{P}_{2n}$ e quindi non è una proiezione in tale spazio. Come poi affermato nel Teorema 1.2.2, nel caso in cui S sia una formula di cubatura esatta in $\mathbb{P}_{n+k}$ con $k \leq n$, si dimostra che

$$\|f - L_n f\|_{L^2} \leq \left(1 + \frac{1}{\sqrt{1-\eta}}\right) \sqrt{\mu(\Omega)} E_k(f). \tag{6}$$

In particolare, per $k = n$ si ha $\eta = 0$ e si recupera la classica iperinterpolazione di Sloan (7).

In generale, però, la perdita della proprietà di proiezione può indurre un errore addizionale, spesso interpretato come *aliasing*. In [1] questo fenomeno emerge quando S è una formula di cubatura quasi-Monte Carlo (QMC), dove, poiché non sono progettate per essere esatte in $\mathbb{P}_{2n}$, non è garantita l'iperinterpolazione classica e, dunque, nelle stime dell'errore viene prodotto un termine aggiuntivo detto *termine di aliasing*. Se $0 < \eta < 1$:

$$\|f - L_n f\|_{L^2} \leq \left(1 + \sqrt{1+\eta}\right) \sqrt{\mu(\Omega)} E_n(f) + \|p_n^* - L_n p_n^*\|_{L^2}, \tag{7}$$

dove p_n^* è la migliore approssimazione uniforme di f in $\mathbb{P}_n$, cioè $\|f - p_n^*\|_\infty = E_n(f)$, mentre il secondo addendo nel membro destro è il termine di aliasing.

Per calcolare numericamente le costanti $A > 0$ e $B > 0$ nell'articolo [1], vengono identificate come il più piccolo e il più grande autovalore della matrice di Gram della misura discreta corrispondente alla regola di cubatura. Quindi, sotto le condizioni definite precedentemente (3) su A e B, la matrice di Gram risulta ben condizionata e ciò permette di calcolare con accuratezza la proiezione ortogonale discreta (equivalente a una proiezione ai minimi quadrati pesati) su $\mathbb{P}_n$:

$$G_n f = \sum_{j=1}^{d_n} c_j p_j \qquad \{c_j\} = G^{-1}\{S(f p_j)\}, \qquad G = (S(p_i p_j))_{1 \leq i,j \leq d_n}. \tag{8}$$

Per quanto riguarda l'errore di approssimazione ai minimi quadrati, segue il seguente risultato.

Teorema 2.1.1. *Sia $G_n f$ il polinomio ai minimi quadrati pesati definito da* (1) *e* (8)*, dove*

$f \in C(\Omega)$ e la formula di cubatura soddisfa la proprietà MZ (2)-(3). *Allora vale la stima*

$$\|f - G_n f\|_{L^2} \leq \left(1 + \sqrt{cond_2(G)}\right) \sqrt{\mu(\Omega)} E_n(f) \leq \left(1 + \sqrt{\frac{1+\eta}{1-\eta}}\right) \sqrt{\mu(\Omega)} E_n(f) \quad (9)$$

che, nel caso in cui la formula di cubatura sia esatta sulle costanti, può essere raffinata in

$$\|f - G_n f\|_{L^2} \leq \left(1 + \frac{1}{\sqrt{A}}\right) \sqrt{\mu(\Omega)} E_n(f) \leq \left(1 + \frac{1}{\sqrt{1-\eta}}\right) \sqrt{\mu(\Omega)} E_n(f) \quad (10)$$

Dimostrazione. Dalla disuguaglianza MZ (2) e dal fatto che $G_n f$ è una proiezione ortogonale, possiamo scrivere la catena di disuguaglianze

$$\begin{aligned}
\|f - G_n f\|_{L^2} &\leq \|f - p_n^*\|_{L^2} + \|p_n^* - G_n p_n^*\|_{L^2} + \|G_n(p_n^* - f)\|_{L^2} \\
&= \|f - p_n^*\|_{L^2} + \|G_n(p_n^* - f)\|_{L^2} \\
&\leq \|f - p_n^*\|_{L^2} + \frac{1}{\sqrt{A}} \sqrt{S((G_n(p_n^* - f))^2)} \\
&= \|f - p_n^*\|_{L^2} + \frac{1}{\sqrt{A}} \|G_n(p_n^* - f)\|_{\ell_w^2} \\
&\leq \|f - p_n^*\|_{L^2} + \frac{1}{\sqrt{A}} \|p_n^* - f\|_{\ell_w^2} \\
&\leq \left(\sqrt{\mu(\Omega)} + \sqrt{\frac{\sum w_i}{A}}\right) \|f - p_n^*\|_{L^2} = \left(\sqrt{\mu(\Omega)} + \sqrt{\frac{\sum w_i}{A}}\right) E_n(f).
\end{aligned}$$

Ora, se la formula di cubatura è esatta almeno sulle costanti, abbiamo $\sum_i w_i = \mu(\Omega)$ e, dunque, vale (10):

$$\|f - G_n f\|_{L^2} \leq \left(1 + \frac{1}{\sqrt{A}}\right) \sqrt{\mu(\Omega)} E_n(f) \leq \left(1 + \frac{1}{\sqrt{1-\eta}}\right) \sqrt{\mu(\Omega)} E_n(f).$$

In alternativa, dalla disuguaglianza MZ, poiché i pesi sono positivi,

$$\sum_i w_i = |S(1)| \leq B\|1\|_{L^2}^2 = B \int_\Omega 1 d\mu = B\mu(\Omega),$$

e quindi, usando $cond_2(G) = B/A$ e l'ultima riga della serie di disuguaglianze sopra, si ottiene (9):

$$\|f - G_n f\|_{L^2} \leq \left(1 + \sqrt{cond_2(G)}\right) \sqrt{\mu(\Omega)} E_n(f) \leq \left(1 + \sqrt{\frac{1+\eta}{1-\eta}}\right) \sqrt{\mu(\Omega)} E_n(f)$$

che vale anche se la formula di cubatura S non è esatta sulle costanti. □

Si noti che se S è una regola di cubatura esatta in $\mathbb{P}_{n+k}$ con $k < n$, allora (10) è una stima migliore di (6). Inoltre, per una formula di cubatura generale che soddisfa (2)-(3), sia (9) che (10) non contengono alcun termine di aliasing, a differenza di (7), poiché $G_n f$ è una proiezione.

Si enuncia e dimostra ora un risultato fondamentale che aiuta a calcolare numericamente

le costanti deboli di Marcinkiewicz–Zygmund.

Teorema 2.1.2 (Caratterizzazione tramite autovalori delle costanti deboli di Marcinkiewicz–Zygmund)**.** *Sia S una regola di cubatura a pesi positivi, e sia $\{\phi_j\}_{j=1}^{d_n}$ una base μ-ortonormale dello spazio polinomiale $\mathbb{P}_n$. Si indica con G la corrispondente matrice di Gram con componenti*

$$G_{j,k} = S(\phi_j, \phi_k), \qquad 1 \le j, k \le d_n. \tag{11}$$

Allora le costanti ottimali $A = A(n)$ e $B = B(n)$ che soddisfano la disuguaglianza debole di Marcinkiewicz–Zygmund (2) *sono precisamente il minimo e il massimo autovalore della matrice di Gram G:*

$$A = \lambda_{\min}(G) \qquad B = \lambda_{max}(G). \tag{12}$$

Di conseguenza, la costante η è definita come in (3)*.*

Dimostrazione. Poiché la disuguaglianza (2) è banalmente verificata dal polinomio nullo per qualunque $A, B > 0$, ci si può restringere al caso $\|p\|_{L^2} = 1$, cioè trovare

$$A = \min_{\substack{p\in\mathbb{P}_n \\ \|p\|_{L^2}=1}} |S(p^2)|, \qquad B = \max_{\substack{p\in\mathbb{P}_n \\ \|p\|_{L^2}=1}} |S(p^2)|. \tag{13}$$

Si suppone che $p = \sum_{j=1}^{d_n} c_j \phi_j$ e che $\|p\|_{L^2}^2 = I(p^2) = 1$, cioè $\sum_{j=1}^{d_n} c_j^2 = 1$. Posto $c = \{c_j\}$, si ha

$$\begin{aligned}
S(p^2) &= \sum_{i=1}^{M} w_i\, p(x_i)^2 = \sum_{i=1}^{M} w_i \left(\sum_{j=1}^{d_n} c_j\, \phi_j(x_i) \right)^2 \\
&= \sum_{i=1}^{M} w_i \left(\sum_{j=1}^{d_n} c_j^2\, \phi_j(x_i)^2 + 2 \sum_{1\le j<k\le d_n} c_j c_k\, \phi_j(x_i)\phi_k(x_i) \right) \\
&= \sum_{j=1}^{d_n} c_j^2 \sum_{i=1}^{M} w_i\, \phi_j(x_i)^2 + 2 \sum_{1\le j<k\le d_n} c_j c_k \sum_{i=1}^{M} w_i\, \phi_j(x_i)\phi_k(x_i) \\
&= \sum_{j=1}^{d_n} c_j^2\, S(\phi_j^2) + 2 \sum_{1\le j<k\le d_n} c_j c_k\, S(\phi_j\phi_k) \\
&= \sum_{j=1}^{d_n} c_j^2\, G_{j,j} + 2 \sum_{1\le j<k\le d_n} c_j c_k\, G_{j,k} \\
&= c^T G c.
\end{aligned} \tag{14}$$

Quindi il problema di trovare i valori estremi in (13) equivale a risolvere i problemi di ottimizzazione

$$\min_{c^Tc=1} c^T G c \qquad \max_{c^Tc=1} c^T G c \tag{15}$$

In altre parole, bisogna minimizzare/massimizzare la forma quadratica associata alla matrice di Gram G sulla sfera unitaria. Per il teorema min-max:

- A è il minimo autovalore $\lambda_{\min}(G)$ della matrice semidefinita positiva G;
- B è il massimo autovalore $\lambda_{\max}(G)$ della matrice semidefinita positiva G.

Si osserva che A può essere anche 0, poiché G è solo semidefinita positiva. In questo caso la formula non possiede una proprietà di Marcinkiewicz–Zygmund, poiché è richiesto che $A > 0$. Inoltre, i polinomi p_A e p_B, i cui coefficienti nella base ortonormale sono le componenti degli autovettori unitari relativi agli autovalori estremi di G, soddisfano

$$A = S(p_A^2), \qquad B = S(p_B^2). \tag{16}$$

Ora si può determinare il più piccolo $\eta \geq 0$ tale che valga (4). Ci si può restringere alla norma L^2 unitaria, ottenendo

$$\eta = \max_{\substack{p \in \mathbb{P}_n \\ \|p\|_{L^2}=1}} |I(p^2) - S(p^2)|. \tag{17}$$

Si considera $p = \sum_{j=1}^{d_n} c_j \phi_j$ e supponiamo $I(p^2) = 1$, cioè $\sum_{j=1}^{d_n} c_j^2 = 1$. Da (14),

$$\begin{aligned} \big|I(p^2) - S(p^2)\big| &= \big|1 - S(p^2)\big| \\ &= \left| 1 - \left(\sum_{j=1}^{d_n} c_j^2 \, S(\varphi_j^2) + 2 \sum_{1 \leq j < k \leq d_n} c_j c_k \, S(\varphi_j \varphi_k) \right) \right| \\ &= \left| 1 - \left(\sum_{j=1}^{d_n} c_j^2 \, G_{j,j} + 2 \sum_{1 \leq j < k \leq d_n} c_j c_k \, G_{j,k} \right) \right| \\ &= \big|1 - c^T G c\big| = \big|c^T c - c^T G c\big| = \big|c^T (I_d - G) c\big|. \end{aligned} \tag{18}$$

Per ottenere η dobbiamo massimizzare il membro destro di (18) su tutti i vettori c con $\|c\|_2 = 1$, il che equivale a calcolare il raggio spettrale della matrice d'errore $E = I_d - G$, dove I_d è la matrice identità. In altre parole,

$$\eta = \|E\|_2 = \max\{|1 - \lambda_{\min}(G)|, |1 - \lambda_{\max}(G)|\} = \max\{|1 - A|, |1 - B|\}. \tag{19}$$

□

Si osservi che il risultato precedente non assume che la formula S abbia un certo grado di esattezza e può essere applicato a misure μ generali, anche discrete.

Inoltre se si suppone che una successione di formule S_m, $m = 1, 2, \ldots$, sia convergente sui polinomi, ciò è una condizione necessaria e sufficiente per la convergenza di $C(\Omega)$ (generalizzazione multivariata del teorema di Polya-Steklov). Allora la successione delle matrici di Gram G_m converge alla matrice identità e i loro autovalori convergono a 1. Di conseguenza, fissato arbitrariamente $\eta \in (0, 1)$, esiste $m^* = m^*(n) \in \mathbb{N}$ tale che S_{m^*} soddisfi (4) per $p \in \mathbb{P}_n$. Un esempio è dato dalle formule QMC basate su un numero convergente

di punti a bassa discrepanza su Ω, poiché esse sono convergenti su ogni polinomio fissato e, in particolare, sui polinomi $\phi_j\phi_k \in \mathbb{P}_{2n}$ con $1 \leq j,k \leq d_n$.

L'ultima osservazione riguarda invece il peggior errore in (4). Infatti, sia $\hat{c} \in \mathbb{R}^{d_n}$ un autovettore unitario di $E = I_d - G$ relativo all'autovalore $\lambda_{\max}(E)$ (di modulo massimo) di E, dall'interpretazione algebrica lineare del problema, il polinomio $\hat{p} \in \mathbb{P}_n$ di norma L^2 uguale a 1, che realizza l'errore peggiore in (4), è

$$\hat{p} = \sum_{j=1}^{d_n} \hat{c}_j \phi_j.$$

2.1.1 Caso delle formule gaussiane S_k a k punti

Nell'articolo [1] si cerca di rispondere alla domanda: una formula di quadratura a grado algebrico di esattezza $ADE = m$ può avere $\eta < 1$ per qualche $n > \lfloor \frac{m}{2} \rfloor$. Infatti, se ciò si verificasse, sarebbe possibile applicare, per esempio, l'iperinterpolazione a esattezza rilassata.

Viene studiato il caso delle formule gaussiane a k punti S_k, che hanno $ADE = m = 2k-1$ rispetto a una funzione peso w sull'intervallo (a,b), eventualmente non limitato. Immediatamente si verifica che se $n \in \mathbb{N}$ e $n < k$, allora $2n < 2k-1$, e quindi, in virtù dell'ADE della formula, si ha $\eta = 0$.

Ora si considera $\eta = k$. Sia $p = \sum_{i=0}^{n} c_i\psi_i$, dove $\{\psi_i\}_{i=0,\dots,n}$ è la famiglia triangolare di polinomi ortonormali rispetto al peso w, cioè $\deg(\psi_i) = i$ e

$$\int_a^b \psi_i(x)\psi_j(x)w(x)dx = \delta_{i,j} \qquad i,j = 0,\dots,n.$$

Se si indica con $\{\psi_i^*\}_{i=0,\dots,n}$ la famiglia triangolare di polinomi ortogonali monici rispetto al peso w in (a,b), si osserva che se $\psi_n(x) = \alpha_n x^n + \cdots + \alpha_0$, allora

$$\psi_n(x) = \alpha_n \psi_n^*(x),$$

poiché ψ_n^* è monico. In particolare, $\alpha_n = \frac{1}{\|\psi_n^*\|_{L_w^2}}$.

Si ricorda che se $f \in C^{2n}(a,b)$, allora

$$\int_a^b f(x)w(x)dx - S_n(f) = \frac{D^{(2n)}f(\xi)}{(2n)!}\|\psi_n^*\|_{L_w^2}^2, \tag{20}$$

dove $\|f\|_{L_w^2}^2 = \int_a^b f(x)^2 w(x)dx$ e $\xi \in (a,b)$.

Ora:

- si suppone $\|p\|_{L_w^2} = 1$, ossia $\sum_{i=0}^{n} c_i^2 = 1$;
- sia $q_{n-1} = \sum_{i=0}^{n-1} c_i\psi_i \in \mathbb{P}_{n-1}$, così che $p = c_n\psi_n + q_{n-1}$.

Dalla formula precedente (20), per qualche $\xi \in (a,b)$ si ottiene:

$$\int_a^b p(x)^2\, w(x)\, dx - S_n(p^2) = \frac{D^{(2n)}p^2(\xi)}{2n!}\, \|\psi_n^*\|_{L^2_w}^2 = \frac{D^{(2n)}(c_n\psi_n + q_{n-1})^2(\xi)}{2n!}\, \|\psi_n^*\|_{L^2_w}^2 \quad (21)$$
$$= c_n^2\, \frac{D^{(2n)}\psi_n^2(\xi)}{2n!}\, \|\psi_n^*\|_{L^2_w}^2 = c_n^2 \frac{2n!\alpha_n^2}{2n!}\frac{1}{\alpha_n^2} = c_n^2.$$

Poiché $\|p\|_{L^2_w} = \sum_{i=0}^n c_i^2 = 1$, se $n = k$ ne segue che

$$\eta = \max_{\substack{p\in\mathbb{P}_n \\ \|p\|_{L^2_w}=1}} |I(p^2) - S(p^2)| = c_n^2 \quad (22)$$

e il massimo è 1. Quindi, per le formule gaussiane a m punti, quando $n = k$ la quantità η vale 1. Poiché η non è decrescente in n, otteniamo $\eta \geq 1$ per $n \geq k$. Ne segue il seguente teorema.

Teorema 2.1.3. *Sia S_k una formula gaussiana a k punti rispetto la funzione peso w: $(a,b) \to \mathbb{R}_+$ con (a,b) non è necessariamente limitato. Allora per $n \geq k$ vale*

$$\eta = \max_{\substack{p\in\mathbb{P}_n \\ \|p\|_{L^2_w}=1}} |I(p^2) - S(p^2)| \geq 1.$$

Osservazione Si osservi che da (22) si ricava anche che i polinomi detti di *peggior errore* (in inglese *worst case error*) sono $\hat{p}_n = \pm\psi_n$.

Nel caso di domini a prodotto (rettangoli) e misure prodotto $d\mu = d\mu_1 d\mu_2$, è frequente usare formule di quadratura a prodotto tensoriale $S_k = S_k^{(1)} \otimes S_k^{(2)}$ costruite a partire da formule gaussiane univariate con grado di esattezza $m = 2k-1$. Tuttavia, tali formule non garantiscono $\eta < 1$ quando il grado n raggiunge il numero k di punti della formula univariata: scegliendo $n = k$ e un opportuno polinomio $p(x,y)$, si ottiene infatti $|I(p^2) - S_k(p^2)| = 1$. Poiché η è non decrescente in n, segue che $\eta \geq 1$ per ogni $n \geq k$. La stessa conclusione vale su "box" (parallelepipedi) di dimensione arbitraria per misure prodotto e regole gaussiane a prodotto tensoriale.

Capitolo 3

Formule di quadratura Fejér e Clenshaw-Curtis

In questo breve capitolo introduciamo alcune formule di quadratura classiche basate sui nodi di Chebyshev, in particolare le formule di Fejér e di Clenshaw-Curtis. Queste formule sono ampiamente utilizzate nell'integrazione numerica e nell'approssimazione polinomiale grazie alla loro buona stabilità numerica e alla possibilità di calcolare efficientemente i pesi mediante trasformate discrete. Nel nostro caso, per studiare sperimentalmente il comportamento delle costanti di Marcinkiewicz–Zygmund di queste formule e il loro ruolo nella costruzione degli operatori di iperinterpolazione.

Nel lavoro [5] vengono proposti algoritmi per il calcolo delle formule di Fejér di tipo I e II e delle formule di Clenshaw-Curtis, in modo che i loro nodi $\{x_k\}_{k=1,\dots,n}$ e i corrispondenti pesi $\{w_k\}$ forniscano una formula di quadratura di tipo interpolatorio per una generica funzione peso w

$$\int_{-1}^{1} f(x)w(x)dx \approx \sum_{k=1}^{n} w_k f(x_k),$$

dove, quindi, il suo grado di precisione algebrica è pari a $n-1$. Questo significa che

$$\int_{-1}^{1} p_s(x)w(x)dx = \sum_{k=1}^{n} w_k p_s(x_k),$$

per ogni p_s appartenente all'insieme $\mathbb{P}_{n-1}$ dei polinomi algebrici di grado minore o uguale a $n-1$. In particolare, i nodi $\{x_k\}$ sono di tipo Chebyshev, mentre i pesi $\{w_k\}$ sono calcolati tramite somme di funzioni trigonometriche.

Questi risultati erano già stati discussi da Waldvogel nel seguente articolo [6] dove, però, i pesi delle formule di quadratura di Fejér e Clenshaw-Curtis sono ottenuti come trasformata discreta di Fourier di un vettore definito di numeri razionali o algebrici (FFT o iFFT) nel caso specifico della funzione peso di Legendre $w(x) \equiv 1$. L'articolo [5] generalizza le tecniche di Waldvogel, mostrando come, per funzioni peso generali w, le formule di quadratura di Fejér e di Clenshaw-Curtis siano collegate alle trasformate discrete del coseno

(DCT) e del seno (DST), permettendo di calcolare i nodi e i pesi una volta noti i momenti modificati pesati dei polinomi di Chebyshev di prima o seconda specie.

Facciamo quindi un breve cenno a queste formule di quadratura, rimandando ai lavori sopra elencati per ulteriori dettagli.

3.1 Formula di Fejér di tipo I

La formula di Fejér di tipo I con n punti nell'intervallo di riferimento $(-1, 1)$ ha nodi

$$x_k = \cos(\theta_k), \qquad \theta_k = \frac{(2k-1)\pi}{2n}, \qquad k = 1, \ldots, n, \tag{1}$$

cioè i nodi di Chebyshev. Riscrivendo il polinomio di Chebyshev di grado m come $T_m(x) = \cos(m \arccos(x))$, si ha, per una funzione peso generale $w : (-1, 1) \to \mathbb{R}$,

$$w_k = \frac{1}{n}\left(\gamma_0 \, + \, 2\sum_{m=1}^{n-1} \gamma_m T_m(x_k)\right), \qquad \text{con} \qquad \gamma_m = \int_{-1}^{1} T_m(x) w(x) dx. \tag{2}$$

Pertanto

$$w_k = \frac{1}{n}\left(\gamma_0 \, + \, 2\sum_{m=1}^{n-1} \gamma_m \cos\left(\frac{m(2k-1)\pi}{2n}\right)\right), \qquad k = 1, \ldots, n. \tag{3}$$

Le quantità (3) sono calcolate nell'articolo [5] tramite una trasformata discreta di tipo III (abbreviata come DCT III).

3.2 Formula di Fejér di tipo II

La formula di Fejér di tipo II con n punti ha nodi

$$x_k = \cos(\theta_k), \qquad \theta_k = \frac{k\pi}{n+1}, \qquad k = 1, \ldots, n \tag{4}$$

cioè gli zeri del polinomio di Chebyshev di seconda specie di grado n

$$U_n(x) = \frac{\sin((n+1)\arccos(x))}{\sin(\arccos(x))}. \tag{5}$$

Sia $w : (-1, 1) \to \mathbb{R}$ una funzione peso. Nell'articolo [5] vengono determinati i pesi $\{w_k\}$ usando la funzione peso di Legendre $w \equiv 1$ e la trasformata discreta del seno di tipo I. Di seguito viene riportata solo una parte della dimostrazione. Dalla formula di Darboux-Christoffel si ha

$$U_n(x) = \frac{2(x_k - x)}{U_{n+1}(x_k)} \sum_{s=0}^{n} U_s(x_k) U_s(x). \tag{6}$$

Siano $\lambda_s = \int_{-1}^{1} U_s(x)w(x)dx$ i momenti pesati dei polinomi di Chebyshev di seconda specie (rispetto a w). Allora

$$\begin{aligned} w_k &= \frac{1}{U_n'(x_k)} \int_{-1}^{1} \frac{U_n(x)}{x - x_k}\, w(x)\, dx \\ &= -\frac{2}{U_n'(x_k)U_{n+1}(x_k)} \sum_{s=0}^{n} U_s(x_k) \int_{-1}^{1} U_s(x)\, w(x)\, dx \\ &= -\frac{2}{U_n'(x_k)U_{n+1}(x_k)} \sum_{s=0}^{n} U_s(x_k)\lambda_s. \end{aligned} \tag{7}$$

Poiché $U_{n+1}(x_k) = T_{n+1}(x_k)$, $U_n(x_k) = 0$, $T_{n+1}^2(x_k) = 1$, e

$$U_n'(x) = \frac{(n+1)T_{n+1}(x) - x\, U_n(x)}{x^2 - 1},$$

otteniamo immediatamente

$$\frac{1}{U_n'(x_k)U_{n+1}(x_k)} = \frac{x_k^2 - 1}{(n+1)T_{n+1}^2(x_k)} = \frac{x_k^2 - 1}{n+1}. \tag{8}$$

Di conseguenza, per $k = 1, \ldots, n$

$$\begin{aligned} w_k &= \frac{2(1 - x_k^2)}{n+1} \sum_{s=0}^{n} U_s(x_k)\lambda_s = \frac{2\big(1 - \cos^2(\theta_k)\big)}{n+1} \sum_{s=0}^{n} \lambda_s \frac{\sin\big((s+1)\theta_k\big)}{\sin(\theta_k)} \\ &= \frac{2\sin(\theta_k)}{n+1} \sum_{s=0}^{n} \lambda_s \sin\big((s+1)\theta_k\big) = \frac{2\sin(\theta_k)}{n+1} \sum_{s=0}^{n-1} \lambda_s \sin\big((s+1)\theta_k\big). \end{aligned} \tag{9}$$

dove nell'ultima uguaglianza viene usato il fatto che $\sin\big((n+1)\theta_k\big) = 0$.

3.3 Formula di Clenshaw-Curtis

La formula di Clenshaw-Curtis con n nodi nell'intervallo di riferimento $[-1, 1]$ ha nodi

$$x_k = \cos\left(\frac{(k-1)\pi}{n-1}\right) \qquad k = 1, \ldots, n. \tag{10}$$

Si verifica immediatamente che tali nodi coincidono con quelli della formula di Fejér di tipo II con $n - 2$ nodi, a cui si aggiungono gli estremi dell'intervallo $-1, 1$. Ponendo

$$P_n(x) \equiv (1 - x^2)U_{n-2}(x), \qquad \Pi_n(x) \equiv \prod_{k=1}^{n}(x - x_k),$$

si ha, per $j = 2, \ldots, n-1$, poiché $U_{n-2}(x_2) = \cdots = U_{n-2}(x_{n-1}) = 0$,

$$w_j = \frac{1}{\Pi_n'(x_j)} \int_{-1}^{1} \frac{\Pi_n(x)}{x - x_j}\, w(x)\, dx = \frac{1}{P_n'(x_j)} \int_{-1}^{1} \frac{P_n(x)}{x - x_j}\, w(x)\, dx$$
$$= \frac{1}{(1 - x_j^2) U_{n-2}'(x_j)} \int_{-1}^{1} \frac{(1 - x^2) U_{n-2}(x)}{x - x_j}\, w(x)\, dx. \tag{11}$$

Definendo

$$w^*(x) = (1 - x^2) w(x),$$

si ottiene

$$w_j = \frac{1}{(1 - x_j^2) U_{n-2}'(x_j)} \int_{-1}^{1} \frac{U_{n-2}(x)}{x - x_j}\, w^*(x)\, dx. \tag{12}$$

Confrontando (12) con la prima equazione in (7), si osserva che, se

$$w_k^{(f2)} = \frac{1}{U_{n-2}'(x_k)} \int_{-1}^{1} \frac{U_{n-2}(x)}{x - x_k}\, w^*(x)\, dx, \qquad k = 1, \ldots, n-2,$$

sono i pesi della regola di Fejér di tipo II con $n-2$ nodi rispetto al peso w^*, allora

$$w_j = \frac{1}{1 - x_j^2}\, w_{j-1}^{(f2)}, \qquad j = 2, \ldots, n-1.$$

In altre parole, i pesi $w_2, \ldots, w_{n-1}$ della formula di Clenshaw-Curtis rispetto a w si ottengono dai pesi della regola di Fejér di tipo II con peso modificato w^*.

Capitolo 4

Risultati numerici

In questo capitolo verificheremo numericamente i risultati e le considerazioni sviluppate fino ad ora. Prima, però, è utile riassumere brevemente il contenuto della parte numerica dell'articolo [1]. Gli autori cercano di valutare le costanti di Marcinkiewicz–Zygmund associate ad alcune note formule di cubatura su domini significativi come l'intervallo, il quadrato, il disco, il triangolo e la sfera. In particolare, intendono rispondere numericamente se una formula di quadratura con grado algebrico di esattezza $ADE = m$ possa avere $\eta < 1$ per qualche $n > \lfloor \frac{m}{2} \rfloor$?

Come visto nel Capitolo 2, nel caso delle formule gaussiane ciò non accade. Di seguito, invece, riporteremo il comportamento osservato negli altri domini considerati. Per ciascuno di essi, gli autori impostano sostanzialmente lo stesso esperimento numerico. Scelgono una o più formule da testare e confrontare, fissano un grado di esattezza m e un grado polinomiale n, valutano una base ortonormale di $\mathbb{P}_n$ adatta a quel dominio, costruiscono la matrice di Gram discreta G, calcolano $E = I - G$, la costante di Marcinkiewicz-Zygmund $\eta = \|E\|_2$ e il numero di condizionamento $cond_2(G)$. Quindi, ciò che varia nei diversi casi è la scelta della misura di riferimento, la base ortonormale e la famiglia di formule da testare. Per quanto riguarda i dettagli, rimandiamo all'articolo [1]. Di seguito saranno riportati solo i risultati ottenuti per l'intervallo e il quadrato, anche per formule basate sulle regole di Fejér.

4.1 Esperimenti sull'intervallo

In questa sezione intendiamo svolgere esperimenti sull'intervallo $[-1, 1]$, riprendendo il metodo dell'articolo [1] descritto sopra e aggiungendo il confronto con le formule di quadratura di Fejér di tipo I e di tipo II. Poiché la misura di riferimento è la misura di Lebesgue, la base ortonormale è data dai polinomi di Legendre ortonormali.

A tal fine utilizziamo una routine Matlab che implementa automaticamente, per ciascuna formula considerata, il calcolo delle costanti di Marcinkiewicz–Zygmund.

La routine principale impiegata è `demo_MZ_interval`, che organizza l'intera batteria di esperimenti numerici. Essa riceve in input un vettore di valori del grado algebrico di

esattezza `m`, un vettore di gradi polinomiali `n`, un indice che identifica la famiglia di formule di quadratura da analizzare, alcuni parametri ausiliari per la stampa delle statistiche e il salvataggio di figure e tabelle. La funzione non esegue direttamente tutti i calcoli, ma, per ciascuna formula selezionata, richiama la routine `make_test`, alla quale delega il calcolo delle costanti di MZ, degli estremi spettrali della matrice di Gram e del relativo numero di condizionamento. La costruzione concreta della formula di quadratura è affidata alla funzione `define_rule`, nella quale la variabile `pset_type` identifica il tipo di formula e `deg` rappresenta il grado algebrico di esattezza m.

Nel caso di Gauss-Legendre, la routine utilizzata accetta direttamente in input il grado di esattezza desiderato.

Diversamente, per le formule di Fejér, la funzione costruttiva `cub_interval_fejer` è parametrizzata dal numero di nodi e dal tipo di formula definito da `ftype`, cui assegniamo un valore pari a `1` nel caso di Fejér I e a `2` per Fejér II. Per questo motivo, per poter confrontare le diverse formule a parità di grado di esattezza m, inizialmente, nel codice si procede a tradurre il valore di m nel corrispondente numero di nodi della formula di Fejér. In particolare, la formula di Fejér viene costruita mediante il comando `cub_interval_fejer(card,ftype)`, con `card` uguale al numero di nodi scelto, nel nostro caso pari a `m+1` cosí da avere una formula con ADE=m.

Ora fissati m e n, la routine `make_test` costruisce dapprima la formula base di quadratura corrispondente e ne estrae nodi e pesi e successivamente, mediante la funzione `vandermonde_orthonormal`, valuta nei nodi di quadratura la base ortonormale di Legendre di $\mathbb{P}_n$, cioè, in questo caso, i primi $n+1$ polinomi di Legendre ortonormali rispetto alla misura di Lebesgue. In seguito, a partire dalle valutazioni della base ortonormale e dei pesi della formula di quadratura, tramite i codici `compute_GRAM` e `compute_MZ_constant` vengono costruite la matrice di Gram, la matrice di errore $E = I - G$, la costante di Marcinkiewicz–Zygmund $\eta = \|E\|_2$, gli estremi spettrali A e B (autovalori della matrice G) e il numero di condizionamento. Infine, i risultati vengono visualizzati da `plot_MZ_figures`, che produce grafici delle curve di livello della costante η nel piano (m, n) e i grafici relativi al condizionamento della matrice di Gram. Nel nostro caso, sono stati considerati i livelli

$$\eta \approx 10^{-12}, \qquad \eta = 10^{-1}, \qquad \eta = 1, \qquad \eta = 10,$$

in modo da distinguere:

- la regione di quasi esattezza numerica;
- una regione intermedia in cui la costante è ancora piccola;
- la soglia critica $\eta = 1$;
- una regione in cui il comportamento diventa nettamente sfavorevole.

Dalla Figura (4.1) notiamo che le curve di livello della costante di Marcinkiewicz-Zygmund η per il caso gaussiano presentano un salto quando n passa da $k = \lfloor m/2 \rfloor$ a $k + 1$, poiché

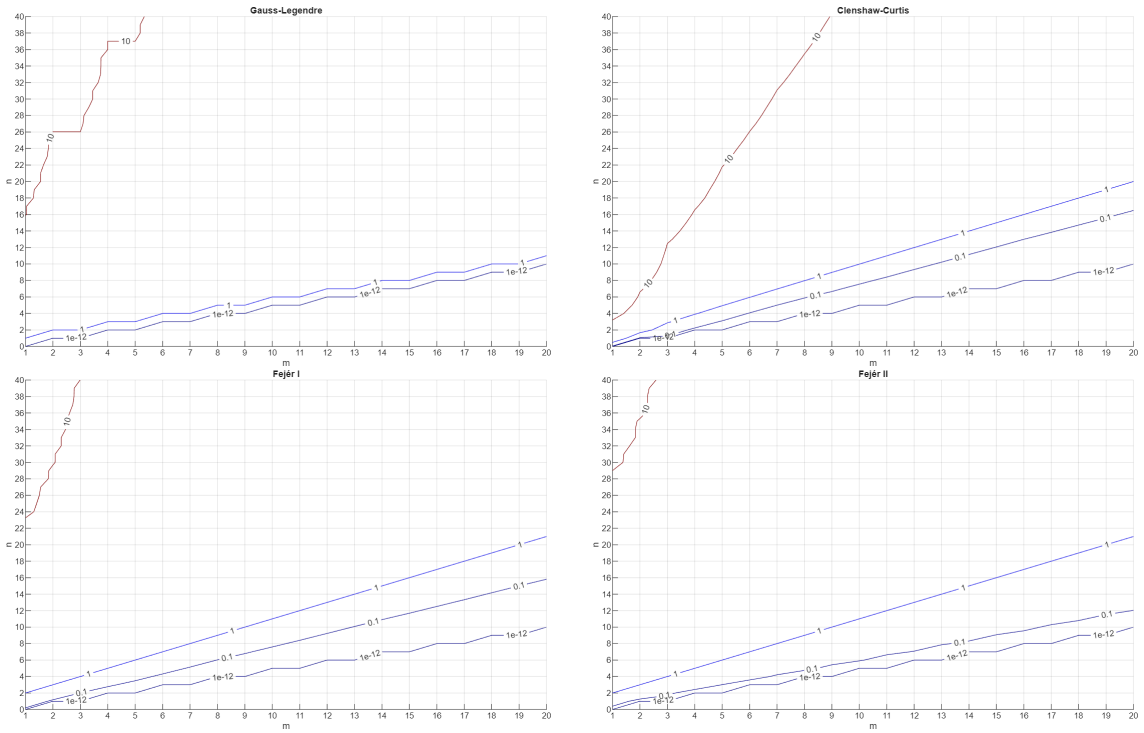

Figura 4.1: Curve di livello delle costante di Marcinkiewicz-Zygmund η in (4) per le formule di Gauss-Legendre (in alto a sinistra), di Clenshaw-Curtis (in alto a destra), di Fejér di tipo I (in basso a sinistra) e di Fejér di tipo II (in basso a destra). Per tutte le figure abbiamo analizzato il caso in cui $m = 1, \ldots, 20$ e $n = 0, 1, \ldots, 40$.

la costante MZ varia bruscamente dai valori prossimi alla precisione di macchina a 1, come previsto dalla teoria.

Per le formule di Clenshaw-Curtis e per quelle di Fejér, invece, la variazione è molto più lenta. Le curve di livello $\eta \approx 0$ e $\eta = 1$, in entrambi i casi, divergono in modo significativo mentre per Clenshaw-Curtis, per un dato m, la costante MZ risulta strettamente minore di 1 per $\lfloor m/2 \rfloor < n \leq m - 1$, invece per Fejér, poiché $n = m + 1$, η risulta strettamente minore di 1 per $n < m$. Possiamo perciò concludere che anche le formule di quadratura di Fejér, analogamente a quelle di Clenshaw-Curtis, riescono ad approssimare integrali di polinomi di grado più alto di quanto ci si aspetterebbe, conservando anch'esse la proprietà di Marcinkiewicz-Zygmund oltre la soglia $n = \lfloor m/2 \rfloor$.

Confrontiamo ora il numero di condizionamento della matrice di Gram. Dalla Figura (4.2) notiamo che le formule di Fejér si comportano in modo simile a Clenshaw-Curtis, infatti, a differenza di Gauss-Legendre, la matrice di Gram rimane ben condizionata su una regione di piano (m, n) che si estende fino a circa $n = m$. Oltre questa soglia $\text{cond}_2(G)$ cresce rapidamente fino a rendere G quasi singolare.

Procedendo ora come nell'articolo [1], confrontiamo gli errori relativi in norma L^2 dell'iperinterpolazione classica di Sloan, dell'iperinterpolazione in versione *unfettered* e dell'approssimazione ai minimi quadrati, mantenendo fissa la formula di quadratura di partenza. Dai risultati ottenuti, infatti, vediamo che le formule di quadratura di Clenshaw-Curtis e di Fejér sono buoni candidati per l'iperinterpolazione ad esattezza rilassata. Poiché la misura

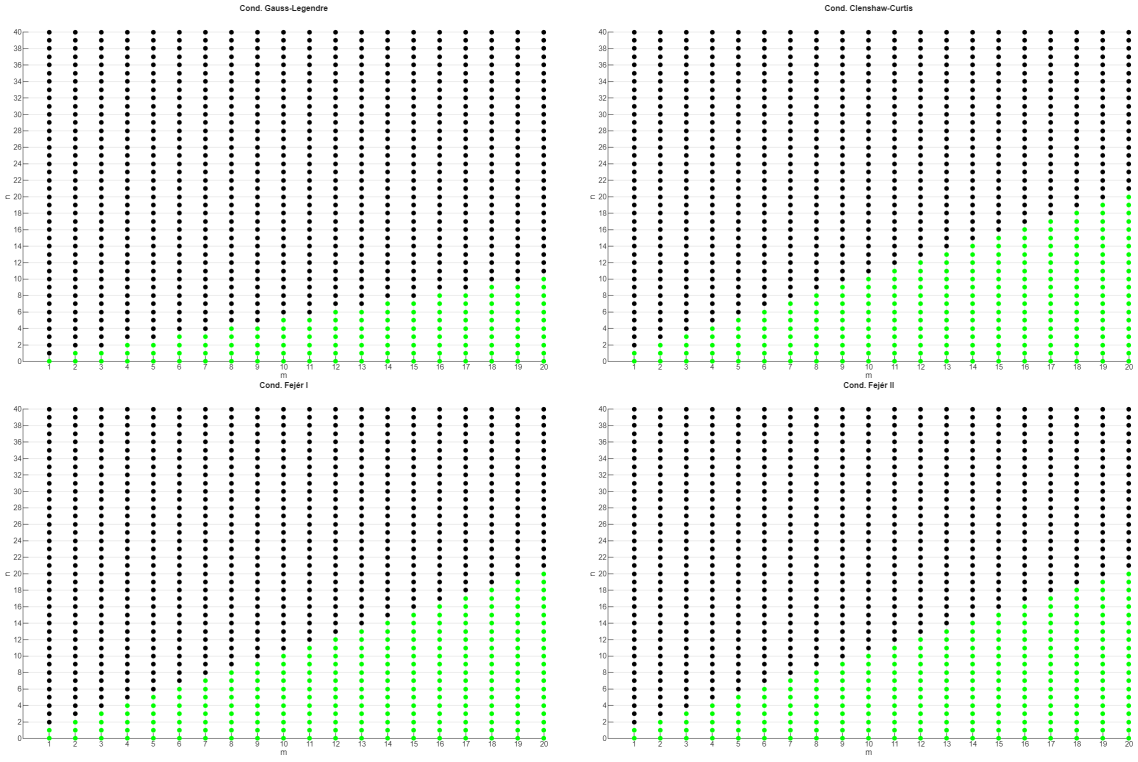

Figura 4.2: Condizionamento in norma 2 della matrice di Gram associata alle formule di Gauss-Legendre (in alto a sinistra), di Clenshaw-Curtis (in alto a destra), di Fejér di tipo I (in basso a sinistra) e di Fejér di tipo II (in basso a destra). Per le prime due figure in alto si ha $m =$ADE, mentre nelle due in basso m è il numero di nodi, e per tutte le figure abbiamo analizzato il caso in cui $m = 1, \ldots, 20$ e $n = 0, 1, \ldots, 40$. Punti verdi: $\text{cond}_2(G) \in [1, 10)$. Punti neri: $\text{cond}_2(G) \in [10^7, +\infty)$.

di riferimento è la misura di Lebesgue, si sceglie come base ortonormale di $\mathbb{P}_n$ la base di Legendre prodotto a grado totale. La routine Matlab usata in questo caso dell'intervallo è `demo_FTEST_interval` che confronta:

- L: iperinterpolazione rilassata costruita con una formula di quadratura di grado algebrico di esattezza m;
- H: iperinterpolazione classica costruita con una formula di grado algebrico di esattezza $2m$;
- G: approssimazione ottenuta tramite la matrice di Gram associata alla formula con grado di esattezza m, cioè una proiezione discreta (minimi quadrati pesati).

Fissato m, per ogni grado polinomiale $n = 1, \ldots, m$, la routine costruisce i tre approssimanti nella base ortonormale di Legendre e ne valuta l'errore rispetto a diverse funzioni test. Esse sono:

- $f_1(x) = e^{-x^2}$
- $f_2(x) = (0.5 + x)^{15}$
- $f_3(x) = \sin(\pi x)$

- $f_4(x) = |x - 0.5|^3$
- $f_5(x) = |x - 0.5|^7$

Gli errori sono misurati in L^2 mediante una formula di quadratura di riferimento sufficientemente accurata. Infine, la routine produce tre grafici in scala semilogaritmica:

- il primo mostra l'errore dell'iperinterpolazione rilassata costruita con la formula di grado m;
- il secondo mostra l'errore dell'iperinterpolazione classica;
- il terzo mostra l'errore del metodo dei minimi quadrati basato sulla matrice di Gram.

In ciascun grafico, le diverse curve corrispondono alle diverse funzioni test, mentre l'asse orizzontale rappresenta il grado polinomiale n.

Inoltre, come avevamo visto anche sopra, nel caso delle formule di Fejér, la routine di Matlab è parametrizzata dal numero di nodi e non dal grado algebrico di esattezza. Per rendere il confronto coerente con le altre formule considerate (Gauss-Legendre e Clenshaw-Curtis), nel codice si fissa comunque un valore m e si sceglie una formula di Fejér con $m + 1$ nodi, che ha grado di esattezza pari a m. Riportiamo ora i grafici delle formule di quadratura di Fejér di tipo I e di tipo II, rimandando all'articolo [1] per il confronto con Clenshaw-Curtis.

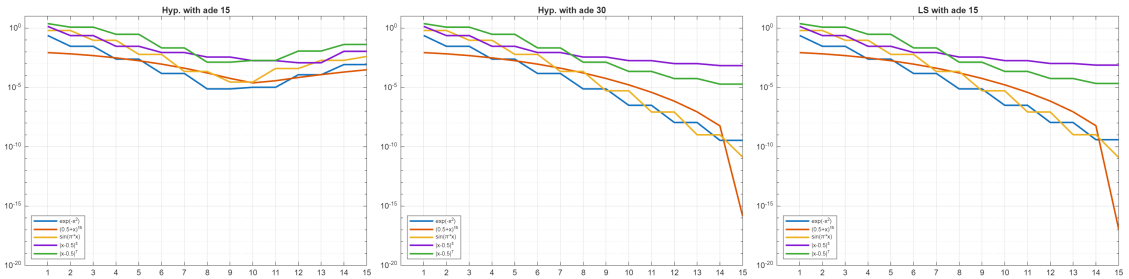

Figura 4.3: Confronti numerici sull'intervallo unitario [-1,1]. Errore di iperinterpolazione ai gradi $1, 2, \ldots, 15$ ottenuto con la formula di Fejér di tipo I con ADE=15 (a sinistra), con la formula di Gauss-Legendre con ADE=30 (al centro), ed errore ai minimi quadrati ottenuto usando la formula di Fejér di tipo I con ADE=15 (a destra)

Le Figure (4.3) e (4.4) mostrano che, per gradi polinomiali bassi, i tre metodi considerati producono risultati sostanzialmente coincidenti. Tuttavia, al crescere di n, l'iperinterpolazione rilassata costruita con una formula di grado algebrico di esattezza $ADE = 15$ perde rapidamente accuratezza, mentre il metodo basato sulla matrice di Gram mantiene errori molto più contenuti. In particolare, la proiezione discreta ai minimi quadrati ottenuta tramite la matrice di Gram risulta nettamente migliore dell'iperinterpolazione *unfettered* costruita con la stessa formula e, in molti casi, fornisce errori quasi indistinguibili da quelli dell'iperinterpolazione classica costruita con una formula di grado $ADE = 30$. Questo conferma che l'uso della matrice di Gram consente di recuperare quasi interamente l'accuratezza del caso classico pur partendo da una formula di quadratura meno accurata.

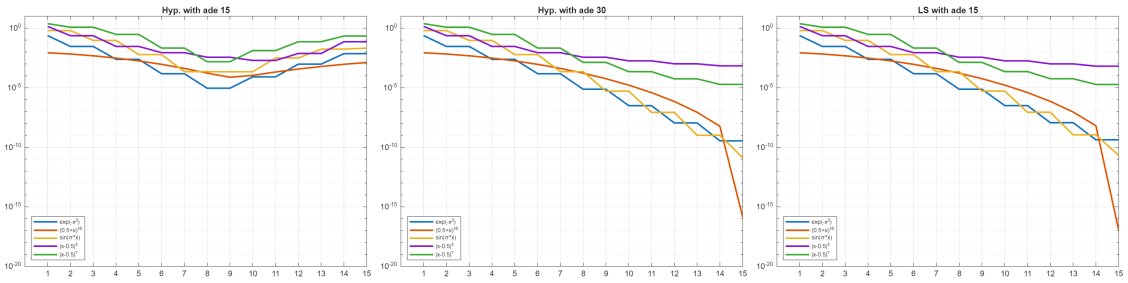

Figura 4.4: Confronti numerici sull'intervallo unitario [-1,1]. Errore di iperinterpolazione ai gradi $1, 2, \ldots, 15$ ottenuto con la formula di Fejér di tipo II con ADE=15 (a sinistra), con la formula di Gauss-Legendre con ADE=30 (al centro), ed errore ai minimi quadrati ottenuto usando la formula di Fejér di tipo II con ADE=15 (a destra)

4.2 Esperimenti sul quadrato

Consideriamo ora il caso del quadrato unitario $[-1,1]^2$. A tal proposito richiamiamo brevemente come si ottengono le formule di tipo tensoriale, dette anche formule prodotto, che verranno utilizzate per gli esperimenti.

A tal proposito, consideriamo dapprima un dominio normale bivariato

$$\Omega = \{(x,y) : a \leq x \leq b,\ \psi(x) \leq y \leq \phi(x)\}$$

dove $\psi, \phi : [a,b] \to \mathbb{R}$ sono due funzioni sufficientemente regolari. Ponendo

$$g(x) := \int_{\psi(x)}^{\phi(x)} f(x,y)dy,$$

e usando la regola

$$\int_a^b g(x)dx \approx \sum_{i=1}^{n} w_i g(x_i),$$

si ottiene, per il calcolo integrale elementare,

$$\begin{aligned} I(f) := \int_\Omega f(x)\, d\Omega = \int_a^b \left(\int_{\psi(x)}^{\phi(x)} f(x,y)dy \right) dx = \int_a^b g(x)\, dx \\ \approx \sum_{i=1}^{n} w_i g(x_i) = \sum_{i=1}^{n} w_i \int_{\psi(x)}^{\phi(x)} f(x_i, y)\, dy \end{aligned}$$

A questo punto, ciascuno degli n integrali interni può essere a sua volta approssimato mediante una formula di quadratura a m punti.

Si noti che anche se l'intervallo di integrazione dipende dall'indice i, questo non costituisce un problema, poiché si può riscalare la formula (ad esempio si può usare una formula di Gauss-Legendre traslata da $[-1,1]$ a $[\psi(x_i), \phi(x_i)]$). Se infatti

$$\int_{\psi(x_i)}^{\phi(x_i)} f(x_i, y)\, dy \approx \sum_{j=1}^{m} v_{j,i} f(x_i, y_{j,i}),$$

si ottiene infine la formula di cardinalità mn

$$I(f) \approx \sum_{i=1}^{n} w_i \sum_{j=1}^{m} v_{j,i} f(x_i, y_{j,i}) = \sum_{i=1}^{n} \sum_{j=1}^{m} w_i v_{j,i} f(x_i, y_{j,i}).$$

Si osserva quindi che l'idea di base consiste nel costruire una formula multidimensionale componendo opportunamente formule univariate. Inoltre, in questa costruzione non si è fatta alcuna ipotesi sul grado di esattezza delle formule impiegate.

Ritornando al nostro caso del quadrato unitario $[-1, 1]^2$, la situazione è più semplice, poiché il dominio è il prodotto cartesiano di due intervalli uguali. Infatti, il quadrato unitario può essere visto come un caso particolare di dominio naturale, scegliendo $a = -1$, $b = 1$ e assumendo come funzioni le costanti $\psi(x) = -1$ e $\phi(x) = 1$. In questo contesto, la formula tensoriale si ottiene direttamente come prodotto di due formule univariate, una della variabile x e una della variabile y. Nel seguito, richiederemo inoltre che tali formule abbiano un grado di esattezza fissato.

A differenza dell'articolo [1], in questa sezione confrontiamo le formule prodotto tensoriale basate su formule di Gauss-Legendre, su formule di Clenshaw-Curtis e su formule di Fejér, utilizzando un procedimento del tutto analogo a quello descritto per l'intervallo. Tramite la routine Matlab `demo_MZ_square` otteniamo i risultati numerici riportati nelle figure successive.

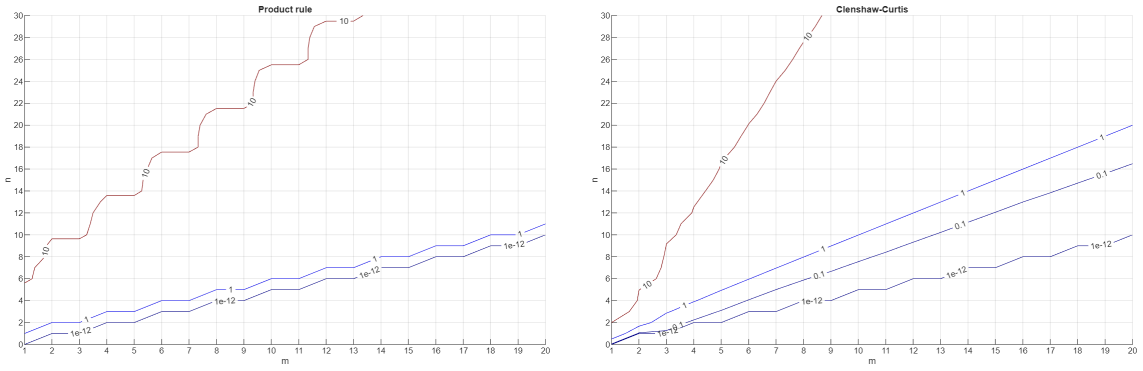

Figura 4.5: Curve di livello delle costante di Marcinkiewicz-Zygmund η in (4) per le formule di Gauss-Legendre (a sinistra) e di Clenshaw-Curtis (a destra). Con m indichiamo ADE dove $m = 1, \ldots, 20$ e $n = 0, 1, \ldots, 30$.

Dalle figure (4.5) e (4.6) notiamo che, anche nel caso del quadrato, la formula prodotto di Clenshaw-Curtis mostra un comportamento migliore rispetto a quella di Gauss-Legendre. In particolare, nel caso gaussiano le curve di livello della costante MZ η passano molto rapidamente dalla precisione di macchina a valori prossimi a 1. Allo stesso tempo, condizionamento cond_2 della matrice di Gram cresce bruscamente non appena si supera la soglia $\lfloor m/2 \rfloor$.

Nel caso di Clenshaw-Curtis, invece, la crescita della costante MZ risulta molto più graduale. Analogamente a quanto osservato sull'intervallo, essa rimane strettamente minore di 1 anche per valori di n superiori a $\lfloor m/2 \rfloor$, più precisamente per $\lfloor \frac{m}{2} \rfloor < n < m - 1$. Anche il

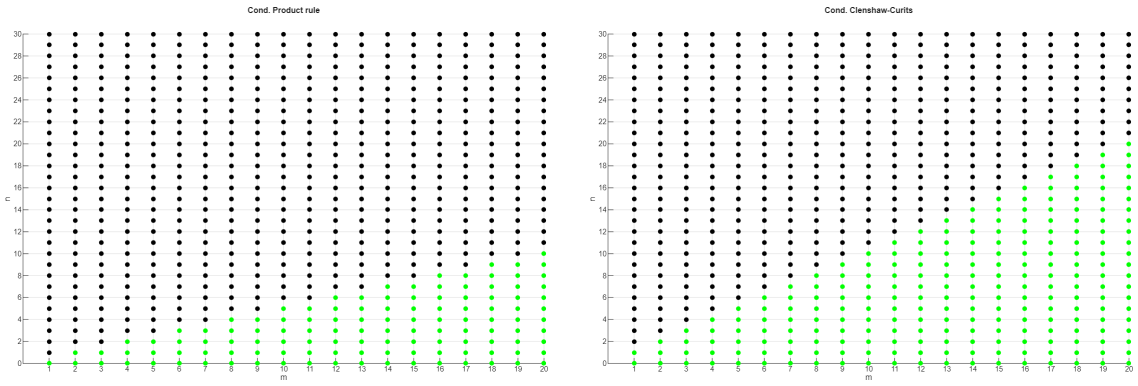

Figura 4.6: Condizionamento in norma 2 della matrice di Gram associata alle formule prodotto di Gauss-Legendre (a sinistra) e di Clenshaw-Curtis (a destra). Con m indichiamo ADE dove $m = 1, \ldots, 20$ e $n = 0, 1, \ldots, 30$. Punti verdi: $\text{cond}_2(G) \in [1, 10)$. Punti neri: $\text{cond}_2(G) \in [10, +\infty)$.

grafico del condizionamento conferma questo comportamento: per $\lfloor \frac{m}{2} \rfloor < n \leq m$, il valore di cond_2 rimane moderato, inferiore a 10, mentre soltanto per $n > m$ cresce rapidamente e la matrice G diventa quasi singolare.

Nel complesso, questi risultati mostrano che, anche nel quadrato, la formula prodotto di Clenshaw-Curtis preserva più a lungo buone proprietà di stabilità rispetto alla formula di Gauss-Legendre, consentendo di lavorare con gradi polinomiali più elevati prima che la costante η superi la soglia critica.

Infine, analizziamo cosa succede nel caso in cui vengono utilizzate le formule prodotto di Fejér di tipo I e di tipo II. Dalla Figura (4.7) si osserva che, analogamente a quanto accade per Clenshaw-Curtis, anche le formule prodotto di Fejér di tipo I e II mostrano una crescita più graduale della costante η rispetto al caso gaussiano. In particolare, le curve di livello segnalano che la condizione di η minore di 1 si mantiene su una regione più ampia del piano (m, n). Anche dal punto di vista del condizionamento, le due formule di Fejér si comportano in modo favorevole: la matrice di Gram rimane ben condizionata fino a gradi relativamente elevati, mentre la perdita di stabilità si manifesta per valori più grandi di n.

Similmente all'intervallo, utilizziamo una routine Matlab analoga `demo_FTEST_square` e confrontiamo l'iperinterpolazione classica, l'iperinterpolazione rilassata o *unfettered* e l'approssimazione ai minimi quadrati.

Le funzioni test rispetto alle quali vengono valutati gli errori sono:

- $f_1(x, y) = \exp(-(x^2 + y^2))$
- $f_2(x, y) = (0.5 + x + 0.1y)^{15}$
- $f_3(x, y) = \sin(\pi x + \pi y)$
- $f_4(x, y) = d((x, y), (0.5, 0.5))^3 = ((x - 0.5)^2 + (y - 0.5)^2)^{3/2}$
- $f_5(x, y) = d((x, y), (0.5, 0.5))^7 = ((x - 0.5)^2 + (y - 0.5)^2)^{7/2}$

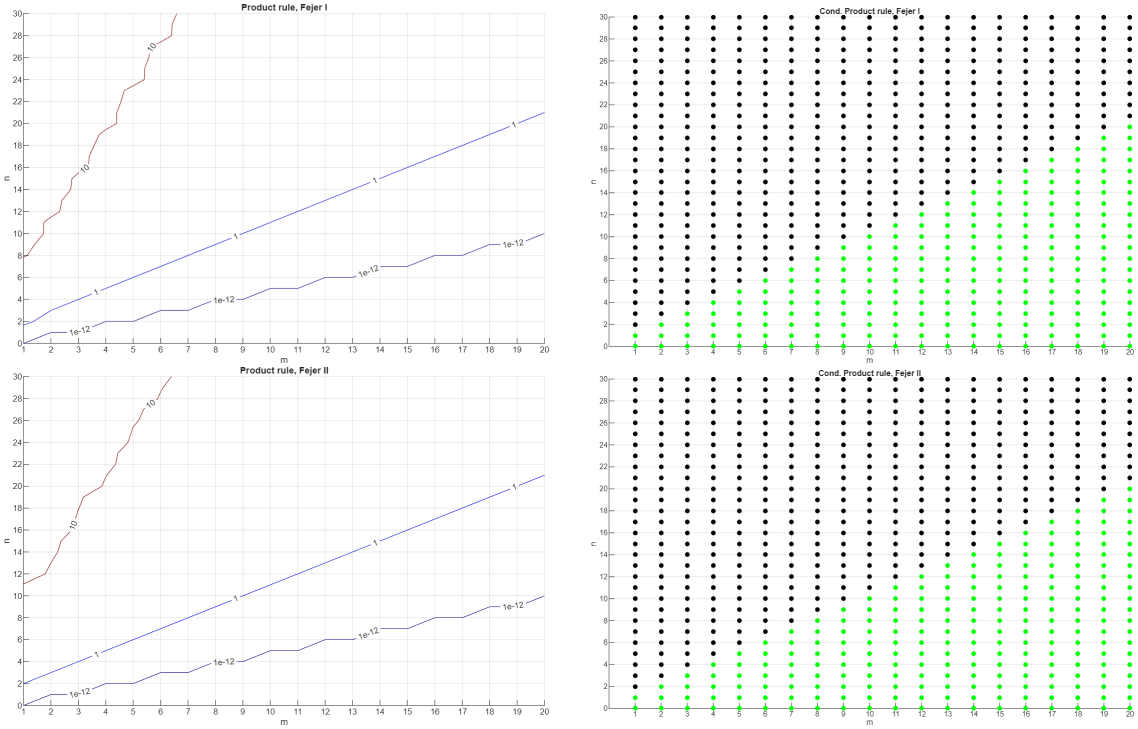

Figura 4.7: Curve di livello delle costante di Marcinkiewicz-Zygmund η in (4) per le formule prodotto di Fejér di tipo I (in alto a sinistra) e di Fejér di tipo II (in basso a sinistra). Condizionamento in norma 2 della matrice di Gram associata alle formule prodotto di Fejér di tipo I (in alto a destra) e di Fejér di tipo II (in basso a destra). Punti verdi: $\text{cond}_2(G) \in [1, 10)$. Punti neri: $\text{cond}_2(G) \in [10, +\infty)$.

Dalle figure (4.8) e (4.9) si può notare che l'iperinterpolazione rilassata fornisce risultati accettabili solo per valori piccoli di n, mentre tende a peggiorare all'aumentare del grado polinomiale.

L'approssimazione ai minimi quadrati, invece, produce errori vicini a quelli dell'iperinterpolazione classica, pur utilizzando una formula con grado algebrico di esattezza $ADE = 15$ invece che $ADE = 30$. Questo conferma che anche nel caso del quadrato l'approccio basato sulla matrice di Gram e sulla proiezione ai minimi quadrati risulta nettamente più efficace dell'iperinterpolazione *unfettered* a parità di formula.

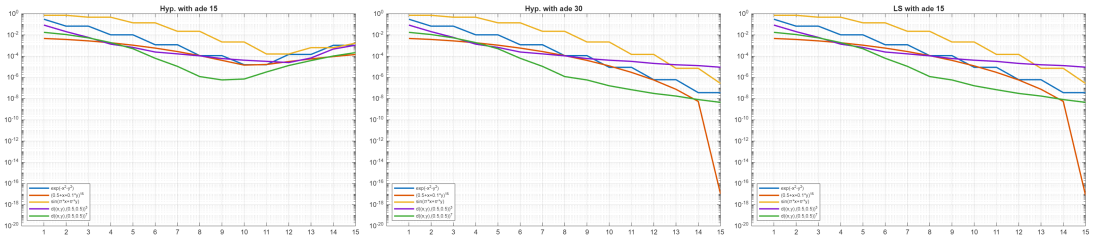

Figura 4.8: Confronti numerici sul quadrato unitario $[-1, 1]^2$. Errore di iperinterpolazione ai gradi $1, 2, \ldots, 15$ ottenuto con la formula prodotto di Fejér di tipo I con ADE=15 (a sinistra), con la formula prodotto di Fejér di tipo I con ADE=30 (al centro), ed errore ai minimi quadrati ottenuto usando la formula prodotto di Fejér di tipo I con ADE=15 (a destra).

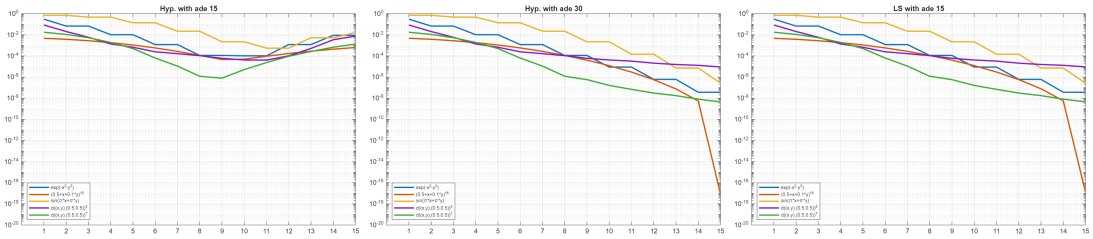

Figura 4.9: Confronti numerici sul quadrato unitario $[-1,1]^2$. Errore di iperinterpolazione ai gradi $1,2,\ldots,15$ ottenuto con la formula prodotto di Fejér di tipo II con ADE=15 (a sinistra), con la formula prodotto di Fejér di tipo II con ADE=30 (al centro), ed errore ai minimi quadrati ottenuto usando la formula prodotto di Fejér di tipo II con ADE=15 (a destra).

Bibliografia

[1] C. An, A. Sommariva e M. Vianello, *On the role of weak Marcinkiewicz–Zygmund constants in polynomial approximation by orthogonal bases*, Preprint, 2026.

[2] C. An e H.-N. Wu, *Bypassing the quadrature exactness assumption of hyperinterpolation on the sphere*, Journal of Complexity 80 (2024), p. 101789.

[3] C. An e H.-N. Wu, *On the quadrature exactness in hyperinterpolation*, BIT Numerical Mathematics (2022), pp. 1899–1919.

[4] I. H. Sloan, *Polynomial Interpolation and Hyperinterpolation over General Regions*, Journal of Approximation Theory 83 (1995), pp. 238–254.

[5] A. Sommariva, *Fast construction of Fejér and Clenshaw–Curtis rules for general weight functions*, Computers and Mathematics with Applications 65 (2013), pp. 682–693.

[6] J. Waldvogel, *Fast Construction of the Fejér and Clenshaw–Curtis Quadrature Rules*, BIT Numerical Mathematics 43.1 (2003), pp. 1–18.

Appendice

Vengono di seguito riportate le principali routines MATLAB utilizzate in questa tesi, con i relativi commenti ed eventuale documentazione.

Gli stessi codici sono disponibili, anche come programmi singoli, in forma open-source al link

`https://github.com/agnesepassi/costantiMZ`

Seguiranno due sezioni; nella prima vengono raccolte le principali routine utilizzate per l'intervallo, mentre la seconda riporta le routine del quadrato.

Funzioni MATLAB intervallo

```
function rule=cub_interval_gausslegendre(ade,a,b)

%--------------------------------------------------------------------------
% OBJECT:
%--------------------------------------------------------------------------
% Gauss-Jacobi rule on (-1,1) where "a", "b" are Jacobi weight function
% exponents.
% The rule has algebraic degree of precision "ade".
%--------------------------------------------------------------------------
% Tests.
%--------------------------------------------------------------------------
% Tested on Matlab R2024B, on a PC running Intel(R) N150 (800 MHz) with 16
% GB of RAM.
%--------------------------------------------------------------------------
% License:
%--------------------------------------------------------------------------
% Copyright (C) 2025 Laura Rinaldi, Alvise Sommariva, Marco Vianello.
%
% This program is free software; you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation; either version 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, write to the Free Software
% Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307 USA
%
% Authors:
%
% Laura Rinaldi    <laura.rinaldi@unipd.it>
% Alvise Sommariva <alvise@math.unipd.it>
% Marco Vianello   <marcov@math.unipd.it>
%
% Date: November 04, 2025
%--------------------------------------------------------------------------

if nargin < 3, a=0; b=0; end

N=ceil((ade+1)/2);
ab=r_jacobi(N,a,b);
rule=gauss(N,ab);




%--------------------------------------------------------------------------
% Subroutines
%--------------------------------------------------------------------------

function ab=r_jacobi(N,a,b)

% R_JACOBI Recurrence coefficients for monic Jacobi polynomials.
%
%    ab=R_JACOBI(n,a,b) generates the first n recurrence
%    coefficients for monic Jacobi polynomials with parameters
%    a and b. These are orthogonal on [-1,1] relative to the
%    weight function w(t)=(1-t)^a(1+t)^b. The n alpha-coefficients
%    are stored in the first column, the n beta-coefficients in
%    the second column, of the nx2 array ab. The call ab=
%    R_JACOBI(n,a) is the same as ab=R_JACOBI(n,a,a) and
%    ab=R_JACOBI(n) the same as ab=R_JACOBI(n,0,0).
%
%    Supplied by Dirk Laurie, 6-22-1998; edited by Walter
%    Gautschi, 4-4-2002.

if nargin<2, a=0; end;  if nargin<3, b=a; end
if((N<=0) || (a<=-1) ||(b<=-1))
    error('parameter(s) out of range')
end
nu=(b-a)/(a+b+2);
mu=2^(a+b+1)*gamma(a+1)*gamma(b+1)/gamma(a+b+2);
```

```
if N==1, ab=[nu mu]; return, end
N=N-1; n=1:N; nab=2*n+a+b;
A=[nu (b^2-a^2)*ones(1,N)./(nab.*(nab+2))];
n=2:N; nab=nab(n);
B1=4*(a+1)*(b+1)/((a+b+2)^2*(a+b+3));
B=4*(n+a).*(n+b).*n.*(n+a+b)./((nab.^2).*(nab+1).*(nab-1));
ab=[A' [mu; B1; B']];


function xw=gauss(N,ab)

%    Given a weight function w encoded by the nx2 array ab of the
%    first n recurrence coefficients for the associated orthogonal
%    polynomials, the first column of ab containing the n alpha-
%    coefficients and the second column the n beta-coefficients,
%    the call xw=GAUSS(n,ab) generates the nodes and weights xw of
%    the n-point Gauss quadrature rule for the weight function w.
%    The nodes, in increasing order, are stored in the first
%    column, the n corresponding weights in the second column, of
%    the nx2 array xw.
%    Supplied by Dirk Laurie, edited by Walter Gautschi.

NO=size(ab,1); if NO<N, error('input array ab too short'), end
J=zeros(N);
for n=1:N, J(n,n)=ab(n,1); end
for n=2:N
    J(n,n-1)=sqrt(ab(n,2));
    J(n-1,n)=J(n,n-1);
end
[V,D]=eig(J);
[D,I]=sort(diag(D));
V=V(:,I);
xw=[D ab(1,2)*V(1,:)'.^2];



function rule=cub_interval_clenshawcurtis(ade,card_choice)

%--------------------------------------------------------------------------
% Object
%--------------------------------------------------------------------------
% Clenshaw-Curtis quadrature rule on (-1,1) with ADE equal to "ade".
% The rule is w.r.t. Legendre weight function.
%--------------------------------------------------------------------------
% Input:
%--------------------------------------------------------------------------
% ade: algebraic degree of precision of the rule, i.e. number of points *1.
%      Tested with n <= 14 million nodes.
% card_choice: the Clenshaw-Curtis formula varies ade depending on its
%       cardinality "n" ("ade" is always odd). Fixing "ade" we choose as
%       cardinality "n" following "card_choice"
```

```
%          1: n=ade+1,
%          2: n=2*ceil((ade-1)/2)+1 (minimal).
%-------------------------------------------------------------------------
% Output:
%-------------------------------------------------------------------------
% rule: (n+1) x 2 matrix of nodes and weights.
%-------------------------------------------------------------------------
% Reference:
%-------------------------------------------------------------------------
% "Fast Construction of Fejer and Clenshaw-Curtis rules for general weight
% functions".
% A. Sommariva
%-------------------------------------------------------------------------
% Note:
%-------------------------------------------------------------------------
% If "n" is the cardinality of the rule, then ADE is 2*floor((n-1)/2)+1.
% Converse, a rule with fixed ADE must have n=ceil((ade-1)/2)+1 nodes.
% This has been checked numerically.
%
% The user may decide to use an interpolatory rule with ADE equal to "ade"
% and cardinality "n=ade+1" (see first lines of the actual code below).
%-------------------------------------------------------------------------
% License:
%-------------------------------------------------------------------------
% Copyright (C) 2025 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, write to the Free Software
% Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307 USA
%
% Authors:
%
% Alvise Sommariva <alvise@math.unipd.it>
%
% Date: December 21, 2025
%-------------------------------------------------------------------------

if nargin < 2
    card_choice=1; % 1: n=ade+1, 2: n=2*ceil((ade-1)/2)+1 (minimal).
end
```

```
if card_choice
    n=ade+1;
else
    n=2*ceil((ade-1)/2)+1; % example: ade=12 -> card=13, ade=13 -> card=13.
end

if nargin < 2
    moms=moms_cheb1type_legendre(n-1);
end

momsIIcc=compute_moments_IIwcc(n-3,moms);
theta=(1:n-2)'*pi/(n-1); xx=cos(theta);
w=((2*sin(theta)/(n-1)).*dst(momsIIcc))./(1-xx.^2);
w1=(2*sum(moms)-moms(1)-moms(end))/(2*(n-1));
wn=moms(1)-w1-sum(w);
x=[1;xx;-1]; w=[w1;w;wn];

rule=[x w];




%------------------------------------------------------------------------
% Subroutines
%------------------------------------------------------------------------

function [momsII,momsI]=compute_moments_IIwcc(n,momsI)

% COMPUTATION OF MOMENTS W.R.T. CHEBYSHEV SECOND TYPE FROM CHEBYSHEV FIRST
% TYPE.

momsI=momsI(1:n+3);
momsII=0.5*(momsI(1:end-2)-momsI(3:end));



function moms=moms_cheb1type_legendre(n)

moms=zeros(n+1,0);
m=0:2:n; m=m';
moms_even_degree=2./(1-m.^2);
moms(m+1,1)=moms_even_degree;



function rule=cub_interval_gausslegendre(ade,a,b)

%------------------------------------------------------------------------
% OBJECT:
%------------------------------------------------------------------------
```

```
% Gauss-Jacobi rule on (-1,1) where "a", "b" are Jacobi weight function
% exponents.
% The rule has algebraic degree of precision "ade".
%--------------------------------------------------------------------------
% Tests.
%--------------------------------------------------------------------------
% Tested on Matlab R2024B, on a PC running Intel(R) N150 (800 MHz) with 16
% GB of RAM.
%--------------------------------------------------------------------------
% License:
%--------------------------------------------------------------------------
% Copyright (C) 2025 Laura Rinaldi, Alvise Sommariva, Marco Vianello.
%
% This program is free software; you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation; either version 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, write to the Free Software
% Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307 USA
%
% Authors:
%
% Laura Rinaldi    <laura.rinaldi@unipd.it>
% Alvise Sommariva <alvise@math.unipd.it>
% Marco Vianello   <marcov@math.unipd.it>
%
% Date: November 04, 2025
%--------------------------------------------------------------------------

if nargin < 3, a=0; b=0; end

N=ceil((ade+1)/2);
ab=r_jacobi(N,a,b);
rule=gauss(N,ab);




%--------------------------------------------------------------------------
% Subroutines
%--------------------------------------------------------------------------
```

```
function ab=r_jacobi(N,a,b)

% R_JACOBI Recurrence coefficients for monic Jacobi polynomials.
%
%    ab=R_JACOBI(n,a,b) generates the first n recurrence
%    coefficients for monic Jacobi polynomials with parameters
%    a and b. These are orthogonal on [-1,1] relative to the
%    weight function w(t)=(1-t)^a(1+t)^b. The n alpha-coefficients
%    are stored in the first column, the n beta-coefficients in
%    the second column, of the nx2 array ab. The call ab=
%    R_JACOBI(n,a) is the same as ab=R_JACOBI(n,a,a) and
%    ab=R_JACOBI(n) the same as ab=R_JACOBI(n,0,0).
%
%    Supplied by Dirk Laurie, 6-22-1998; edited by Walter
%    Gautschi, 4-4-2002.

if nargin<2, a=0; end;  if nargin<3, b=a; end
if((N<=0) || (a<=-1) ||(b<=-1))
    error('parameter(s) out of range')
end
nu=(b-a)/(a+b+2);
mu=2^(a+b+1)*gamma(a+1)*gamma(b+1)/gamma(a+b+2);
if N==1, ab=[nu mu]; return, end
N=N-1; n=1:N; nab=2*n+a+b;
A=[nu (b^2-a^2)*ones(1,N)./(nab.*(nab+2))];
n=2:N; nab=nab(n);
B1=4*(a+1)*(b+1)/((a+b+2)^2*(a+b+3));
B=4*(n+a).*(n+b).*n.*(n+a+b)./((nab.^2).*(nab+1).*(nab-1));
ab=[A' [mu; B1; B']];


function xw=gauss(N,ab)

%    Given a weight function w encoded by the nx2 array ab of the
%    first n recurrence coefficients for the associated orthogonal
%    polynomials, the first column of ab containing the n alpha-
%    coefficients and the second column the n beta-coefficients,
%    the call xw=GAUSS(n,ab) generates the nodes and weights xw of
%    the n-point Gauss quadrature rule for the weight function w.
%    The nodes, in increasing order, are stored in the first
%    column, the n corresponding weights in the second column, of
%    the nx2 array xw.
%    Supplied by Dirk Laurie, edited by Walter Gautschi.

NO=size(ab,1); if NO<N, error('input array ab too short'), end
J=zeros(N);
for n=1:N, J(n,n)=ab(n,1); end
for n=2:N
    J(n,n-1)=sqrt(ab(n,2));
    J(n-1,n)=J(n,n-1);
end
```

```
[V,D]=eig(J);
[D,I]=sort(diag(D));
V=V(:,I);
xw=[D ab(1,2)*V(1,:)'.^2];
```

```
function demo_MZ_interval(mV,nV,pset_typeV,show_stats,save_fig,save_tab)

%-------------------------------------------------------------------------
% Object
%-------------------------------------------------------------------------
% Computation of Marcinkiewicz-Zygmund constants on the interval.
% It performs all the tests and provides results.
%-------------------------------------------------------------------------
% Reference paper:
%-------------------------------------------------------------------------
% "On the role of weak Marcinkiewicz-Zygmund constants in polynomial
% approximation by orthogonal bases"
% C. An, A. Sommariva and M. Vianello
%-------------------------------------------------------------------------
% License:
%-------------------------------------------------------------------------
% Copyright (C) 2025 Alvise Sommariva, Marco Vianello.
%
% This program is free software; you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation; either version 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, write to the Free Software
% Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307 USA
%
% Authors:
%
% Alvise Sommariva <alvise@math.unipd.it>
% Marco Vianello <marcov@math.unipd.it>
%
% Date: December 27, 2025
%-------------------------------------------------------------------------

domain='interval';

if nargin < 1, mV=1:20; end
```

```
if nargin < 2, nV=0:20; end
if nargin < 3, pset_typeV=1:2; end
if nargin < 4, show_stats=0; end % show temporary statistics
if nargin < 5, save_fig=0; end
if nargin < 6, save_tab=0; end

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

% Make all make_tests

table_data=cell(length(pset_typeV),1);
A_mat=cell(length(pset_typeV),1);
B_mat=cell(length(pset_typeV),1);
cond_mat=cell(length(pset_typeV),1);


for k=1:length(pset_typeV)
    pset_type=pset_typeV(k);

    if show_stats
        fprintf('\n \t ............... %2.0f .................. \n \n',...
            pset_type);
    end

    [table_data{k},A_mat{k},B_mat{k},cond_mat{k}]=make_test(mV,nV,...
        pset_type,show_stats,save_fig,save_tab,domain);

end

% Assembly summary table.

if show_stats
    assembly_latex_table(table_data,mV);
end









function [ table_data,A_mat,B_mat,cond_mat ] = make_test(mV,nV,pset_type,...
    show_stats,save_fig,save_tab,domain)

% ---------------------- Compute MZ constants -----------------------------

eta_mat=zeros(length(mV),length(nV));
A_mat=eta_mat;
B_mat=eta_mat;
```

```
cond_mat=eta_mat;
mV_mat=eta_mat;
nV_mat=eta_mat;

for i=1:length(mV)

    m=mV(i);
    rule=define_rule(pset_type,m);
    w=rule(:,end);

    for j=1:length(nV)
        n=nV(j);

        mV_mat(i,j)=m; nV_mat(i,j)=n;

        V=vandermonde_orthonormal(n,rule(:,1:end-1),pset_type);
        [eta_mat(i,j),G,~,A_mat(i,j),B_mat(i,j)]=compute_MZ_constant(V',w);
        cond_mat(i,j)=cond(G);

        if show_stats
            fprintf('\n \t m: %-2.0f n: %-2.0f eta: %1.2e A: %1.2e B: %1.2e cond: %1.2e',...
                m,n,eta_mat(i,j),A_mat(i,j),B_mat(i,j),cond_mat(i,j));
        end

    end

end


if show_stats
    fprintf('\n \n');
end

% ---------------------------- Statistics --------------------------------

table_data=print_MZ_statistics(mV,nV,eta_mat,show_stats);

% ---------------------------- Anomalies ---------------------------------

if show_stats
    print_MZ_anomalies(mV,nV,table_data,A_mat,B_mat,eta_mat);
    print_cond_anomalies(mV_mat,nV_mat,eta_mat,cond_mat);
end

% ........................... save MZ matrix ............................

if save_tab
    save_data(domain,pset_type,A_mat,B_mat,cond_mat);
end

% ------------------------- Plot results --------------------------------
```

```
plot_MZ_figures(mV,nV,eta_mat,A_mat,B_mat,cond_mat,domain,pset_type,...
    save_fig);

if show_stats
    fprintf('\n \n');
end









%--------------------------------------------------------------------------
% Subroutines
%--------------------------------------------------------------------------


function V=vandermonde_orthonormal(n,pts,pset_type)

if nargin < 3, pset_type=1; end

switch pset_type
    case {1,2}
        V=legpolys_orthn(n,pts);
end



function rule=define_rule(pset_type,deg)

switch pset_type

    case 1 % Gauss rule
        rule=cub_interval_gausslegendre(deg);

    case 2 % Clenshaw-Curtis
        rule=cub_interval_clenshawcurtis(deg);

    case 3 % Fejer type
        ftype=3;
        card=deg+1;
        rule=cub_interval_fejer(card,ftype);

end
```

```
function demo_FTEST_interval

%---------------------------------------------------------------------------
% Object
%---------------------------------------------------------------------------
% Testing function reconstruction by hyperinterpolation or alternative
% least- intervals methods, on unit- interval [-1,1], based on Gram matrix at
% degree "n" via formulas with "ADE=m" (in particular it may be m <= 2n).
%
% The methods used are
% L : hyperinterpolation via formula with "ADE=m".
% G : least- intervals via Gram matrix using f.la with "ADE=m".
% H : hyperinterpolationvia formula with "ADE=2*n".
%
%  ftype:
%
% 1. f= @(x) exp(-x.^2-y.^2-z.^2);
% 2. f= @(x) (0.5+x+0.1*y+0.4*z).^a;
% 3. f= @(x) sqrt( (x-0.5).^2 + (y-0.5).^2 + (z-0.5).^2 ).^3;
% 4. f= @(x) sqrt( (x-0.5).^2 + (y-0.5).^2 + (z-0.5).^2 ).^7;
% 5. f= @(x) sin(pi*x+pi*y+pi*z);
%---------------------------------------------------------------------------
% Reference paper:
%---------------------------------------------------------------------------
% "On the role of weak Marcinkiewicz-Zygmund constants in polynomial
% approximation by orthogonal bases"
% C. An, A. Sommariva and M. Vianello
%---------------------------------------------------------------------------
% License:
%---------------------------------------------------------------------------
% Copyright (C) 2025 Alvise Sommariva, Marco Vianello.
%
% This program is free software; you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation; either version 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, write to the Free Software
% Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307 USA
%
% Authors:
%
% Alvise Sommariva <alvise@math.unipd.it>
% Marco Vianello <marcov@math.unipd.it>
% Date: December 27, 2025
```

```
%--------------------------------------------------------------------------


clf;
warning off;

pset_typeL=5;             % 1. GL, 2. CC, 3. F1, 5. F2
pset_typeH=1;
pset_typeR=1;

% if pset_type*={1,2,3} then "m" is the ADE used by the rule
% if pset_type*={4} then one used a QMC rule based on "2^m" Halton points

m=15;                     % ADE rule in unfettered hyp. or Least-Squares
nH=1:m;                   % Hyperinterpolation degree
mCH=2*m;                  % ADE classical hyperinterpolation rule
mR=50;                    % ADE reference rule (L2 norms used for errs)

ftypeV=[1 2 5 3 4];

domain='interval';

save_fig=1;



% ......................... main routine below ............................

% Quadrature rule with "ADE=m" (hyperinterpolation up to degree "m").
ruleL=define_rule(pset_typeL,m);
ptsL=ruleL(:,1:end-1); wL=ruleL(:,end);
cardL=length(wL);

% Quadrature rule with "ADE=2*m" (hyperinterpolation up to degree "m").
ruleH=define_rule(pset_typeH,mCH);
ptsH=ruleH(:,1:end-1); wH=ruleH(:,end);
cardH=length(wH);

% Quadrature rule with "ADE=nR" (reference rule).
ruleR=define_rule(pset_typeR,mR);
ptsR=ruleR(:,1:end-1); wR=ruleR(:,end);
cardR=length(wR);

% Function evaluations

fLC=zeros(cardL,length(ftypeV));
fHC=zeros(cardH,length(ftypeV));
fRC=zeros(cardR,length(ftypeV));
norm2f=zeros(length(ftypeV),1);

fstr={'','','','',''};
```

```
for i=1:length(ftypeV)

    % analyse functions as batch
    ftype=ftypeV(i);
    [f,fstr{i}]=define_function(ftype,m); % m: ade, used to define polyn.
    fLC(:,i)=f(ptsL(:,1));
    fHC(:,i)=f(ptsH(:,1));
    fRC(:,i)=f(ptsR(:,1));
    norm2f(i)=wR'*(fRC(:,i)).^2;

end




% ............................ make tests ................................

err2L=zeros(length(nH),length(ftypeV));
err2H=err2L; err2G=err2L;

for k=1:length(nH)

    % define function
    n=nH(k);

    VL=vandermonde_orthonormal(n,ptsL);
    VH=vandermonde_orthonormal(n,ptsH);
    VR=vandermonde_orthonormal(n,ptsR);

    LL=size(VL,2);
    LH=size(VH,2);

    for i=1:length(ftypeV)

        % Hyper: low degree rule
        fL=fLC(:,i);
        coeffL_lowdim=(fL.*wL)'*VL; coeffL_lowdim=coeffL_lowdim';
        coeffL=zeros(LH,1); coeffL(1:LL,1)=coeffL_lowdim;

        % Hyper: high degree rule
        fH=fHC(:,i);
        coeffH=(fH.*wH)'*VH; coeffH=coeffH';

        % GRAM : low degree rule
        coeffG=hypGRAM(VL',fL,wL);

        % Evaluating errors
        fR=fRC(:,i);

        pL_XR=VR*coeffL;
```

```
        pH_XR=VR*coeffH;
        pG_XR=VR*coeffG;

        err2L(i,k)=sqrt(wR'*((fR-pL_XR).^2))/norm2f(i);  % HYP : low ade, low n
        err2H(i,k)=sqrt(wR'*((fR-pH_XR).^2))/norm2f(i);  % HYP : high ade, high n
        err2G(i,k)=sqrt(wR'*((fR-pG_XR).^2))/norm2f(i);  % GRAM: high ade, high n

        fprintf('\n \t n: %-2.0f f: %-2.0f err2L: %1.3e  err2H: %1.3e  err2G: %1.3e ', ...
            n,i,err2L(i,k),err2H(i,k),err2G(i,k));

    end



end


% ............................ make plots ................................

% Hyper.: ade=m

mlim=10^(floor(log10(max([min(min(err2G)) 10^(-20)]))));
Mlim=10^(ceil(log10(max(max(err2G)))));

figure(1)
if pset_typeL == 4
    title_str='QMC Hyp.';
else
    title_str=['Hyp. with ade ',num2str(m)];
end
plot_fig(ftypeV,pset_typeL,nH,err2L,mlim,Mlim,m,fstr,domain,'L',...
    save_fig,title_str);

% Hyper.: ade=2*m
figure(2)
if pset_typeH == 4
    title_str='QMC Hyp.';
else
    title_str=['Hyp. with ade ',num2str(mCH)];
end
plot_fig(ftypeV,pset_typeH,nH,err2H,mlim,Mlim,m,fstr,domain,'H',...
    save_fig,title_str);

% Gram:  ade=m
figure(3)
if pset_typeH == 4
    title_str='QMC LS';
else
    title_str=['LS with ade ',num2str(m)];
end
plot_fig(ftypeV,pset_typeL,nH,err2G,mlim,Mlim,m,fstr,domain,'G',...
```

```
    save_fig,title_str);
hold off;
```

```
%-------------------------------------------------------------------------
% General subroutines
%-------------------------------------------------------------------------

function [f,str]=define_function(ftype,a)

if nargin < 2, a=1; end

switch ftype
    case 1 % Gaussian: analytic
        str='exp(-x^2)';
        f= @(x) exp(-x.^2);

    case 2 % Polynomial
        str=['(0.5+x)^{',num2str(a),'}'];
        f= @(x) (0.5+x).^a;

    case 3 % Low regularity at "0.5".
        str='|x-0.5|^{3}';
        f= @(x) abs(x).^3;

    case 4 % Low regularity at "0.5".
        str='|x-0.5|^{7}';
        f= @(x) abs(x).^7;

    case 5 % Regular with oscillations
        str='sin(\pi*x)';
        f= @(x) sin(pi*x);

end
```

```
function save_figure(domain,mode_str,pset_type,m,save_fig)

if save_fig == 1
    filename=['figure_',domain,'_ftest_',mode_str,'_'...
        num2str(pset_type),'_',num2str(max(m))];
    file_fig=[filename '.fig'];
    savefig(file_fig);
    file_eps=[filename '.eps'];
    saveas(gca,file_eps,'epsc');
end
```

```
function plot_fig(ftypeV,pset_type,nH,err2L,mlim,Mlim,m,fstr,domain,...
    meth_str,save_fig,title_str)

lw=2;

for i=1:length(ftypeV)
    semilogy(nH,err2L(i,:),'-','LineWidth',lw);
    hold on;
    colororder("gem12");
end

ylim([mlim,Mlim]);

grid on;

xticks(1:nH(end));

title(title_str);
lgd=legend(fstr{1},fstr{2},fstr{3},fstr{4},fstr{5},'Location','southwest');
fontsize(lgd,7,'points')
save_figure(domain,meth_str,pset_type,m,save_fig);
hold off;
```

```
%--------------------------------------------------------------------------
% Specialized subroutines
%--------------------------------------------------------------------------
```

```
function V=vandermonde_orthonormal(n,pts)

V=legpolys_orthn(n,pts);
```

```
function rule=define_rule(pset_type,deg)

switch pset_type

    case 1 % Product rule basis
        rule=cub_interval_gausslegendre(deg,0,0);

    case 2 % Low cardinality rule
        rule=cub_interval_clenshawcurtis(deg);

    case 3 % Fejer I
        rule=cub_interval_fejer(deg+1,1);

    case 5 % Fejer II
        rule=cub_interval_fejer(deg+1,2);

end
```

Funzioni MATLAB quadrato

```
function rule=cub_square_prodrule(deg,alphabeta_math)

%--------------------------------------------------------------------------
% OBJECT:
%--------------------------------------------------------------------------
% Tensorial Gauss rule on the square [-1,1]^2.
% The tensorial Jacobi weight functions are described by "alphabeta_math"
% that is a matrix 2 x 2.
% In particular
%
% * alphabeta_math(1,:) is the Jacobi exponend in the x-direction,
% * alphabeta_math(2,:) is the Jacobi exponend in the y-direction.
%
% If "alphabeta_math" is not declared, the Legendre weight will be
% considered.
%--------------------------------------------------------------------------
% Related paper:
```

```
%-------------------------------------------------------------------------
% "Integration and differentiation by orthogonal moments"
% L. Rinaldi, A. Sommariva and M. Vianello
%-------------------------------------------------------------------------
% Tests.
%-------------------------------------------------------------------------
% Tested on Matlab R2024B, on a PC running Intel(R) N150 (800 MHz) with 16
% GB of RAM.
%-------------------------------------------------------------------------
% License:
%-------------------------------------------------------------------------
% Copyright (C) 2025 Laura Rinaldi, Alvise Sommariva, Marco Vianello.
%
% This program is free software; you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation; either version 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, write to the Free Software
% Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307 USA
%
% Authors:
%
% Laura Rinaldi    <laura.rinaldi@unipd.it>
% Alvise Sommariva <alvise@math.unipd.it>
% Marco Vianello   <marcov@math.unipd.it>
%
% Date: November 04, 2025
%-------------------------------------------------------------------------

if nargin < 2, alphabeta_math=[0 0; 0 0]; end % default value

card_gs=ceil((deg+1)/2);

xw=gauss_jacobi(card_gs,alphabeta_math(1,1),alphabeta_math(1,2));
yw=gauss_jacobi(card_gs,alphabeta_math(2,1),alphabeta_math(2,2));

[X,Y]=meshgrid(xw(:,1),yw(:,1));
[WX,WY]=meshgrid(xw(:,2),yw(:,2));

rule=[X(:) Y(:) (WX(:)).*(WY(:))];
```

```
%--------------------------------------------------------------------------
% Subroutines
%--------------------------------------------------------------------------

function xw=gauss_jacobi(N,a,b)

%--------------------------------------------------------------------------
% OBJECT:
%--------------------------------------------------------------------------
% Gauss-Jacobi rule on (-1,1) where "a", "b" are Jacobi weight function
% exponents.
%--------------------------------------------------------------------------
% Tests.
%--------------------------------------------------------------------------
% Tested on Matlab R2024B, on a PC running Intel(R) N150 (800 MHz) with 16
% GB of RAM.
%--------------------------------------------------------------------------
% License:
%--------------------------------------------------------------------------
% Copyright (C) 2025 Laura Rinaldi, Alvise Sommariva, Marco Vianello.
%
% This program is free software; you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation; either version 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, write to the Free Software
% Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307 USA
%
% Authors:
%
% Laura Rinaldi    <laura.rinaldi@unipd.it>
% Alvise Sommariva <alvise@math.unipd.it>
% Marco Vianello   <marcov@math.unipd.it>
%
% Date: November 04, 2025
%--------------------------------------------------------------------------

ab=r_jacobi(N,a,b);
xw=gauss(N,ab);
```

```
function ab=r_jacobi(N,a,b)

% R_JACOBI Recurrence coefficients for monic Jacobi polynomials.
%
%    ab=R_JACOBI(n,a,b) generates the first n recurrence
%    coefficients for monic Jacobi polynomials with parameters
%    a and b. These are orthogonal on [-1,1] relative to the
%    weight function w(t)=(1-t)^a(1+t)^b. The n alpha-coefficients
%    are stored in the first column, the n beta-coefficients in
%    the second column, of the nx2 array ab. The call ab=
%    R_JACOBI(n,a) is the same as ab=R_JACOBI(n,a,a) and
%    ab=R_JACOBI(n) the same as ab=R_JACOBI(n,0,0).
%
%    Supplied by Dirk Laurie, 6-22-1998; edited by Walter
%    Gautschi, 4-4-2002.

if nargin<2, a=0; end;  if nargin<3, b=a; end
if((N<=0) ||(a<=-1) ||(b<=-1))
    error('parameter(s) out of range')
end
nu=(b-a)/(a+b+2);
mu=2^(a+b+1)*gamma(a+1)*gamma(b+1)/gamma(a+b+2);
if N==1, ab=[nu mu]; return, end
N=N-1; n=1:N; nab=2*n+a+b;
A=[nu (b^2-a^2)*ones(1,N)./(nab.*(nab+2))];
n=2:N; nab=nab(n);
B1=4*(a+1)*(b+1)/((a+b+2)^2*(a+b+3));
B=4*(n+a).*(n+b).*n.*(n+a+b)./((nab.^2).*(nab+1).*(nab-1));
ab=[A' [mu; B1; B']];


function xw=gauss(N,ab)

%    Given a weight function w encoded by the nx2 array ab of the
%    first n recurrence coefficients for the associated orthogonal
%    polynomials, the first column of ab containing the n alpha-
%    coefficients and the second column the n beta-coefficients,
%    the call xw=GAUSS(n,ab) generates the nodes and weights xw of
%    the n-point Gauss quadrature rule for the weight function w.
%    The nodes, in increasing order, are stored in the first
%    column, the n corresponding weights in the second column, of
%    the nx2 array xw.
%    Supplied by Dirk Laurie, edited by Walter Gautschi.

N0=size(ab,1); if N0<N, error('input array ab too short'), end
J=zeros(N);
for n=1:N, J(n,n)=ab(n,1); end
for n=2:N
    J(n,n-1)=sqrt(ab(n,2));
    J(n-1,n)=J(n,n-1);
end
```

```
[V,D]=eig(J);
[D,I]=sort(diag(D));
V=V(:,I);
xw=[D ab(1,2)*V(1,:)'.^2];



function rule=cub_square_prodrule_fejer(card_rule,fejer_type)

%--------------------------------------------------------------------------
% OBJECT:
%--------------------------------------------------------------------------
% Tensorial Fejer type rule on the square [-1,1]^2.
% The variable "fejer_type" may assume values
%
% 1: Fejer I, 2: Fejer II, 3: Clenshaw-Curtis.
%
% If "fejer_type" is not declared, the Clenshaw-Curtis case will be
% considered.
%
% The variable "card_rule" is the cardinality of the 1D rule on which the
% rule on the square is based.
%--------------------------------------------------------------------------
% Related paper:
%--------------------------------------------------------------------------
% "Integration and differentiation by orthogonal moments"
% L. Rinaldi, A. Sommariva and M. Vianello
%--------------------------------------------------------------------------
% Tests.
%--------------------------------------------------------------------------
% Tested on Matlab R2024B, on a PC running Intel(R) N150 (800 MHz) with 16
% GB of RAM.
%--------------------------------------------------------------------------
% License:
%--------------------------------------------------------------------------
% Copyright (C) 2026 Agnese Passi, Laura Rinaldi, Alvise Sommariva,
%                    Marco Vianello.
%
% This program is free software; you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation; either version 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, write to the Free Software
```

```
% Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307 USA
%
% Authors:
% Agnese Passi
% Laura Rinaldi    <laura.rinaldi@unipd.it>
% Alvise Sommariva <alvise@math.unipd.it>
% Marco Vianello   <marcov@math.unipd.it>
%
% Date: February 4, 2026
%--------------------------------------------------------------------------

if nargin < 1, card_rule=10; end
if nargin < 2, fejer_type=3; end % default value

xw=cub_interval_fejer(card_rule,fejer_type);
yw=xw;

[X,Y]=meshgrid(xw(:,1),yw(:,1));
[WX,WY]=meshgrid(xw(:,2),yw(:,2));

rule=[X(:) Y(:) (WX(:)).*(WY(:))];



function demo_MZ_square(mV,nV,pset_typeV,fejer_type,show_stats,save_fig,...
    save_tab)

%--------------------------------------------------------------------------
% Object
%--------------------------------------------------------------------------
% Computation of Marcinkiewicz-Zygmund constants on the unit-square.
% It performs all the tests and provide results.
%--------------------------------------------------------------------------
% Reference paper:
%--------------------------------------------------------------------------
% "On the role of weak Marcinkiewicz-Zygmund constants in polynomial
% approximation by orthogonal bases"
% C. An, A. Sommariva and M. Vianello
%--------------------------------------------------------------------------
% License:
%--------------------------------------------------------------------------
% Copyright (C) 2025 Alvise Sommariva, Marco Vianello.
%
% This program is free software; you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation; either version 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, write to the Free Software
% Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307 USA
%
% Authors:
% Agnese Passi
% Alvise Sommariva <alvise@math.unipd.it>
% Marco Vianello <marcov@math.unipd.it>
% Date: February 4, 2026
%--------------------------------------------------------------------------

domain='square';

%--------------------------------------------------------------------------
% Note
%--------------------------------------------------------------------------
% In order to perform Fejer methods set below
% if nargin < 3, pset_typeV=0; end
%
% To modify the family set
% FEJER I:
% if nargin < 4, fejer_type=1; end
% FEJER II:
% if nargin < 4, fejer_type=2; end
% CLENSHAW CURTIS:
% if nargin < 4, fejer_type=3; end
%
% Take into account that for these rules the values "mV(k)" are the
% algebraic degree of precision of these 1D rules in each directions of the
% "k"-th rule, that is the cardinality of the 1D rule is "mV(k)+1".
%--------------------------------------------------------------------------

if nargin < 1, mV=1:20; end
if nargin < 2, nV=0:30; end
if nargin < 3, pset_typeV=0; end
if nargin < 4, fejer_type=3; end % 1:Fejer I,2:Fejer II,3:Clenshaw-Curtis.
if nargin < 5, show_stats=0; end % show temporary statistics
if nargin < 6, save_fig=0; end
if nargin < 7, save_tab=0; end



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

% Make all make_tests
for k=1:length(pset_typeV)
    pset_type=pset_typeV(k);

    if show_stats
```

```
        fprintf('\n \t ................ %2.0f .................. \n \n',...
            pset_type);
    end
    [table_data{k},A_mat{k},B_mat{k},cond_mat{k}]=make_test(mV,nV,...
        pset_type,fejer_type,show_stats,save_fig,save_tab,domain);

end,
% Assembly summary table.
if show_stats
    assembly_latex_table(table_data,mV);
end
```

```
function [table_data,A_mat,B_mat,cond_mat]=make_test(mV,nV,pset_type,...
    fejer_type,show_stats,save_fig,save_tab,domain)


% ---------------------- Compute MZ constants -----------------------------

eta_mat=zeros(length(mV),length(nV));
A_mat=eta_mat;
B_mat=eta_mat;
cond_mat=eta_mat;
mV_mat=eta_mat;
nV_mat=eta_mat;



for i=1:length(mV)

    m=mV(i);
    rule=define_rule(pset_type,m,fejer_type);
    w=rule(:,end);

    for j=1:length(nV)
        n=nV(j);

        mV_mat(i,j)=m; nV_mat(i,j)=n;

        V=vandermonde_orthonormal(n,rule(:,1:end-1),pset_type);
        [eta_mat(i,j),G,~,A_mat(i,j),B_mat(i,j)]=compute_MZ_constant(V',w);
        cond_mat(i,j)=cond(G);

        if show_stats
            fprintf('\n \t m: %-2.0f n: %-2.0f eta: %1.2e A: %1.2e B: %1.2e cond: %1.2e',...
```

```
                m,n,eta_mat(i,j),A_mat(i,j),B_mat(i,j),cond_mat(i,j));
        end

    end

end


if show_stats
    fprintf('\n \n');
end

% ---------------------------- Statistics --------------------------------

table_data=print_MZ_statistics(mV,nV,eta_mat,show_stats);

% ---------------------------- Anomalies ---------------------------------

if show_stats
    print_MZ_anomalies(mV,nV,table_data,A_mat,B_mat,eta_mat);
    print_cond_anomalies(mV_mat,nV_mat,eta_mat,cond_mat);
end

% ........................... save MZ matrix ............................

if save_tab
    save_data(domain,pset_type,A_mat,B_mat,cond_mat);
end

% ------------------------- Plot results --------------------------------

pset_type(1)=pset_type;
pset_type(2)=fejer_type;
plot_MZ_figures(mV,nV,eta_mat,A_mat,B_mat,cond_mat,domain,pset_type,...
    save_fig);

if show_stats
    fprintf('\n \n');
end
```

```
%--------------------------------------------------------------------------
% Subroutines
%--------------------------------------------------------------------------


function V=vandermonde_orthonormal(n,pts,pset_type)

if nargin < 3, pset_type=1; end

switch pset_type
    case 4
        V=dCHEBVAND_orthn(n,pts,[-1 -1; 1 1]);
    otherwise
        V=vandermonde_nlegendre_orthn(n,pts,[-1 -1; 1 1]);
end




function rule=define_rule(pset_type,n,fejer_type)

% Here "n" is the degree of precision of the rule based on interpolation
% properties.

switch pset_type

    case 0 % Product rule (Fejer):
        rule=cub_square_prodrule_fejer(n+1,fejer_type);

    case 1 % Product rule (Gaussian)
        rule=cub_square_prodrule(n,[0 0; 0 0]);

    case 2 % Cubature based on Padua Points
        rule = cub_square_padua(n,[-1 1 -1 1]);

    case 3 % Low cardinality rule
        rule=cub_square_lowcard(n);

    case 4 % MPX rule
        rule=cub_square_mpx(n);
        rule(:,3)=pi^2*rule(:,3);

    case 5 % QMC rule
        card=2^n;
        dbox=[-1 -1; 1 1];
```

```
        dim=2;
        rule=cub_square_QMC(card,dbox,dim);
end



function demo_FTEST_square

%-------------------------------------------------------------------------
% Object
%-------------------------------------------------------------------------
% Testing function reconstruction by hyperinterpolation or alternative
% least-squares methods, on unit-square [-1,1]^2, based on Gram matrix at
% degree "n" via formulas with "ADE=m" (in particular it may be m <= 2n).
%
% The methods used are
% L : hyperinterpolation via formula with "ADE=m".
% G : least-squares via Gram matrix using f.la with "ADE=m".
% H : hyperinterpolationvia formula with "ADE=2*n".
%
%  ftype:
%
% 1. f=@(x,y) exp(-x.^2-y.^2);
% 2. f=@(x,y) (0.5+x+0.1*y).^a;
% 3. f=@(x,y) sqrt( (x-0.5).^2 + (y-0.5).^2 ).^3;
% 4. f=@(x,y) sqrt( (x-0.5).^2 + (y-0.5).^2 ).^7;
% 5. f=@(x,y) sin(pi*x+pi*y);
%-------------------------------------------------------------------------
% Reference paper:
%-------------------------------------------------------------------------
% "On the role of weak Marcinkiewicz-Zygmund constants in polynomial
% approximation by orthogonal bases"
% C. An, A. Sommariva and M. Vianello
%-------------------------------------------------------------------------
% License:
%-------------------------------------------------------------------------
% Copyright (C) 2025 Alvise Sommariva, Marco Vianello.
%
% This program is free software; you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation; either version 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, write to the Free Software
```

```
% Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307 USA
%
% Authors:
%
% Alvise Sommariva <alvise@math.unipd.it>
% Marco Vianello <marcov@math.unipd.it>
% Date: December 27, 2025
%--------------------------------------------------------------------------

%--------------------------------------------------------------------------
% NOTE: in order to make tests with Fejer type rules set below
%
% pset_typeL=0;
% pset_typeH=0;
% pset_typeR=0;
%
% fejer_type=1;           % 1: Fejer I,
% fejer_type=2;           % 2: Fejer II,
% fejer_type=3;           % 3: Clenshaw-Curtis
%--------------------------------------------------------------------------

clf;

m=15;                   % ADE unfettered hyperinterpolation rule
nH=1:m;                 % Hyperinterpolation degree
mCH=2*m;                % ADE classical hyperinterpolation rule
mR=50;                  % degree of reference rule (L2 norms used for errs)

pset_typeL=0;            % 1. Product-rule,      2. Padua points
pset_typeH=0;                        % 3. Low cardinality,   4. MPX
pset_typeR=0;

fejer_type=2;           % 1: Fejer I, 2: Fejer II, 3: Clenshaw-Curtis

ftypeV=[1 2 5 3 4];

domain='square';

save_fig=1;



% ......................... main routine below ............................



% Quadrature rule with "ADE=m" (hyperinterpolation up to degree "m").
ruleL=define_rule([pset_typeL,fejer_type],m);
ptsL=ruleL(:,1:end-1); wL=ruleL(:,end);
cardL=length(wL);
```

```
% Quadrature rule with "ADE=2*m" (hyperinterpolation up to degree "m").
ruleH=define_rule([pset_typeH,fejer_type],mCH);
ptsH=ruleH(:,1:end-1); wH=ruleH(:,end);
cardH=length(wH);

% Quadrature rule with "ADE=nR" (reference rule).
ruleR=define_rule([pset_typeR,fejer_type],mR);
ptsR=ruleR(:,1:end-1); wR=ruleR(:,end);
cardR=length(wR);

% Function evaluations

fLC=zeros(cardL,length(ftypeV));
fHC=zeros(cardH,length(ftypeV));
fRC=zeros(cardR,length(ftypeV));
norm2f=zeros(length(ftypeV),1);

fstr={'','','','',''};

for i=1:length(ftypeV)

    % analyse functions as batch
    ftype=ftypeV(i);
    [f,fstr{i}]=define_function(ftype,m); % m: ade, used to define polyn.
    fLC(:,i)=f(ptsL(:,1),ptsL(:,2));
    fHC(:,i)=f(ptsH(:,1),ptsH(:,2));
    fRC(:,i)=f(ptsR(:,1),ptsR(:,2));
    norm2f(i)=wR'*(fRC(:,i)).^2;

end




% ............................ make tests ................................

err2L=zeros(length(nH),length(ftypeV));
err2H=err2L; err2G=err2L;

for k=1:length(nH)

    % define function
    n=nH(k);

    VL=vandermonde_orthonormal(n,ptsL);
    VH=vandermonde_orthonormal(n,ptsH);
    VR=vandermonde_orthonormal(n,ptsR);

    LL=size(VL,2);
    LH=size(VH,2);
```

```
    for i=1:length(ftypeV)

        % Hyper: low degree rule
        fL=fLC(:,i);
        coeffL_lowdim=(fL.*wL)'*VL; coeffL_lowdim=coeffL_lowdim';
        coeffL=zeros(LH,1); coeffL(1:LL,1)=coeffL_lowdim;

        % Hyper: high degree rule
        fH=fHC(:,i);
        coeffH=(fH.*wH)'*VH; coeffH=coeffH';

        % GRAM : low degree rule
        coeffG=hypGRAM(VL',fL,wL);

        % Evaluating errors
        fR=fRC(:,i);

        pL_XR=VR*coeffL;
        pH_XR=VR*coeffH;
        pG_XR=VR*coeffG;

        err2L(i,k)=sqrt(wR'*((fR-pL_XR).^2))/norm2f(i);  % HYP : low ade, low n
        err2H(i,k)=sqrt(wR'*((fR-pH_XR).^2))/norm2f(i);  % HYP : high ade, high n
        err2G(i,k)=sqrt(wR'*((fR-pG_XR).^2))/norm2f(i);  % GRAM: high ade, high n

        fprintf('\n \t n: %-2.0f f: %-2.0f err2L: %1.3e  err2H: %1.3e  err2G: %1.3e ', ...
            n,i,err2L(i,k),err2H(i,k),err2G(i,k));

    end
end


% ............................ make plots ................................

% Hyper.: ade=m

mlim=10^(floor(log10(max([min(min(err2G)) 10^(-20)]))));
Mlim=10^(ceil(log10(max(max(err2G)))));

figure(1)
title_str=['Hyp. with ade ',num2str(m)];
plot_fig(ftypeV,pset_typeL,nH,err2L,mlim,Mlim,m,fstr,domain,'L',...
    save_fig,title_str);

% Hyper.: ade=2*m
figure(2)
title_str=['Hyp. with ade ',num2str(mCH)];
plot_fig(ftypeV,pset_typeH,nH,err2H,mlim,Mlim,m,fstr,domain,'H',...
    save_fig,title_str);

% Gram:  ade=m
```

```
figure(3)
title_str=['LS with ade ',num2str(m)];
plot_fig(ftypeV,pset_typeL,nH,err2G,mlim,Mlim,m,fstr,domain,'G',...
    save_fig,title_str);
hold off;
```

```
%--------------------------------------------------------------------------
% Subroutines
%--------------------------------------------------------------------------

function [f,str]=define_function(ftype,a)

if nargin < 2, a=1; end

switch ftype
    case 1 % Gaussian: analytic
        str='exp(-x^2-y^2)';
        f=@(x,y) exp(-x.^2-y.^2);

    case 2 % Polynomial
        str=['(0.5+x+0.1*y)^{',num2str(a),'}'];
        f=@(x,y) (0.5+x+0.1*y).^a;

    case 3 % Low regularity at "0.5".
        str='d((x,y),(0.5,0.5))^{3}';
        f=@(x,y) sqrt( (x-0.5).^2 + (y-0.5).^2 ).^3;

    case 4 % Low regularity at "0.5".
        str='d((x,y),(0.5,0.5))^{7}';
        f=@(x,y) sqrt( (x-0.5).^2 + (y-0.5).^2 ).^7;

    case 5 % Regular with oscillations
        str='sin(\pi*x+\pi*y)';
        f=@(x,y) sin(pi*x+pi*y);

end
```

```
function save_figure(domain,mode_str,pset_type,m,save_fig)

if save_fig == 1
    filename=['figure_',domain,'_ftest_',mode_str,'_'...
        num2str(pset_type),'_',num2str(max(m))];
    file_fig=[filename '.fig'];
    savefig(file_fig);
    file_eps=[filename '.eps'];
    saveas(gca,file_eps,'epsc');
end
```

```
function plot_fig(ftypeV,pset_type,nH,err2L,mlim,Mlim,m,fstr,domain,...
    meth_str,save_fig,title_str)

lw=2;

for i=1:length(ftypeV)
    semilogy(nH,err2L(i,:),'-','LineWidth',lw);
    hold on;
    colororder("gem12");
end

ylim([mlim,Mlim]);

grid on;

xticks(1:nH(end));

title(title_str);
lgd=legend(fstr{1},fstr{2},fstr{3},fstr{4},fstr{5},'Location','southwest');
fontsize(lgd,7,'points')
save_figure(domain,meth_str,pset_type,m,save_fig);
hold off;
```

```
%--------------------------------------------------------------------------
% Specialized subroutines
%--------------------------------------------------------------------------

function V=vandermonde_orthonormal(n,pts,pset_type)

if nargin < 3, pset_type=1; end

switch pset_type
    case 4
        V=dCHEBVAND_orthn(n,pts,[-1 -1; 1 1]);
    otherwise
        V=vandermonde_nlegendre_orthn(n,pts,[-1 -1; 1 1]);
end









function rule=define_rule(pset_type,deg)

fejer_type=pset_type(end);
pset_type=pset_type(1);

switch pset_type
    case 0 % Product rule Fejer
        rule=cub_square_prodrule_fejer(deg+1,fejer_type);

    case 1 % Product rule basis
        rule=cub_square_prodrule(deg,[0 0; 0 0]);

    case 2 % Cubature based on Padua Points
        rule = cub_square_padua(deg,[-1 1 -1 1]);

    case 3 % Low cardinality rule
        rule=cub_square_lowcard(deg);

    case 4 % MPX rule
        rule=cub_square_mpx(deg);
        rule(:,3)=pi^2*rule(:,3);
end
```