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

Cubatura adattiva su poligoni sferici

Relatore:
Prof. Alvise Sommariva

Laureanda: Nicole Mietto
Matricola: 2004482

Correlatore:
Prof. Giacomo Elefante

Anno Accademico 2023/2024

11/12/2024

Indice

Introduzione

L'obiettivo di questa tesi è quello di proporre un algoritmo di quadratura adattiva su poligoni sferici $\mathcal{P}$ appartenenti alla sfera unitaria $\mathbb{S}^2$. In altri termini, data una integranda continua $f\colon \mathcal{P} \subseteq \mathbb{S}^2 \to \mathbb{R}$, il suo scopo é di approssimare numericamente

$$I = \int_{\mathcal{P}} f(\mathcal{P}) d\mathcal{P}$$

a meno di una quantitá determinata dalle tolleranze assolute e relative fissate dall'utente.

L'idea consiste nel suddividere inizialmente il dominio $\mathcal{P}$ in triangoli sferici, fornendo una prima approssimazione dell'integrale richiesto I, utilizzando formule di quadratura su questi domini piú semplici e delle stime d'errore.

L'idea é di raffinare opportunamente questa triangolazione iniziale, fino a quando le stime d'errore permettono di stabilire che si é ottenuto un adeguato valore numerico dell'integrale richiesto.

Nel primo capitolo introdurremo delle formule

$$\int_{\mathcal{T}} f(\mathcal{T}) d\mathcal{T} \approx \sum_{i=1}^{N} w_i f(T_i)$$

aventi pesi w_i positivi e nodi T_i appartenenti al fissato triangolo sferico $\mathcal{T}$, con grado di precisione fissato.

Nel secondo capitolo mostreremo come definire l'algoritmo adattativo che permette di calcolare I con un errore dipendente dalle tolleranze assolute e relative fissate dall'utente.

Concluderemo questo lavoro con una sezione numerica in cui mostriamo la bontá dell'approccio seguito, su domini che sono un'approssimazione dell'Australia, di una cardioide sulla sfera e di un decagono sferico.

Nell'appendice forniamo i listati dei codici Matlab che sono stati sviluppati per tale procedura adattativa.

Capitolo 1

Formule di quadratura per poligoni sferici

In questo capitolo tratteremo il calcolo di formule di quadratura algebrica su poligoni sferici $\mathcal{P}$. Come anticipato nell'introduzione, $\mathcal{P}$ é partizionabile in triangoli sferici. Quindi, precedentemente, ci focalizzeremo sulla determinazione di formule di quadratura su triangoli sferici $\mathcal{T}$. Le informazioni esposte si basano su [4].

A sua volta, per individuare tali formule, un ingrediente essenziale é la determinazione di formule di quadratura su settori ellittici, basate sul calcolo di formule gaussiane di natura *subperiodica* in [1].

Di seguito mostriamo come utilizzarle per determinare le formule richieste su $\mathcal{T}$ e successivamente su $\mathcal{P}$.

1.1 Blending lineare di archi di ellisse e relative formule di quadratura

Come anticipato, mostriamo come ottenere una formula di quadratura prodotto per la regione ottenuta tramite blending lineare di due archi di ellisse. Quindi, al paragrafo successivo, focalizzeremo l'attenzione al caso particolare di un settore ellittico. Per maggiori dettagli si consideri [1].

Definiamo dapprima il dominio in cui si intende approssimare l'integrale tramite formule di quadratura. Come precedentemente accennato, esso é ottenuto tramite *blending lineare* di due archi di ellisse.

Innanzitutto si parametrizzano due archi di ellisse rispettivamente come

$$\begin{aligned} P(\theta) &= A_1 \cos(\theta) + B_1 \sin(\theta) + C_1, \\ Q(\theta) &= A_2 \cos(\theta) + B_2 \sin(\theta) + C_2, \end{aligned}$$

in cui $\theta \in [\alpha, \beta]$, $0 \leq \beta - \alpha \leq 2\pi$ e

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

Quindi definiamo con $\mathcal{S}$ la combinazione convessa dei due archi, ovvero la seguente regione ottenuta tramite blending lineare di P e Q

$$\mathcal{S} = \{(x, y) = U(t, \theta) = tP(\theta) + (1 - t)Q(\theta),\ (t, \theta) \in [0, 1] \times [\alpha, \beta]\}.$$

Per determinare una formula di quadratura su tali regioni $\mathcal{S}$, useremo una formula di tipo prodotto basata su formule di Gauss-Legendre e di tipo *trigonometrico*.

A tal proposito indicheremo con

$$\mathbb{T}_n([-w, w]) = \text{span}\{1, \cos(k\theta), \sin(k\theta), \quad 1 \leq k \leq n, \quad \theta \in [-w, w]\},$$

lo spazio dei polinomi trigonometrici di grado al piú n nell'intervallo $[-w, w]$, dove $0 < w \leq \pi$.
La seguente proposizione illustra come ottenere i nodi e i pesi di una formula di quadratura Gaussiana trigonometrica in $[\alpha, \beta]$ a partire dai nodi e dai pesi di una formula di quadratura Gaussiana algebrica in $[-1, 1]$ relativi alla funzione peso $w(x)$, esplicitata nell'enunciato.

Proposizione 1.1 *Siano $\{(\xi_j, \lambda_j)\}_{1 \leq j \leq n+1}$ i nodi e i pesi positivi della formula di quadratura Gaussiana algebrica relativi alla funzione peso:*

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

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

$$\int_\alpha^\beta f(\theta) d\theta = \sum_{j=1}^{n+1} \lambda_j f(\theta_j + \mu), \quad \forall f \in \mathbb{T}_n([\alpha, \beta]), \quad \mu = \frac{\alpha + \beta}{2}, \qquad (1.1)$$

dove

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

Prima di procedere con la prossima proposizione sono necessarie alcune considerazioni su $JU(t,\theta)$, il determinante della matrice Jacobiana della parametrizzazione $U(t,\theta)$.

Assumendo che la mappa U sia iniettiva in $(0,1)\times(\alpha,\beta)$ si ha che $JU(t,\theta)$ non cambia segno in tale prodotto di intervalli.

Eseguendo alcuni calcoli si ottiene

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

dove

$$u(\theta) = u_0 + u_1\cos(\theta) + u_2\sin(\theta)$$

con

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

e

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

con

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

Possiamo ora enunciare la seguente proposizione che fornisce una formula di quadratura prodotto per $\mathcal{S}$

Proposizione 1.2 *Si consideri il dominio*

$$\mathcal{S} = \{(x,y) = U(t,\theta), \quad (t,\theta)\in[0,1]\times[\alpha,\beta], 0<\beta-\alpha\le 2\pi\},$$

dove la trasformazione $U(t,\theta)$ é iniettiva in $(t,\theta)\in(0,1)\times(\alpha,\beta)$. Allora vale la seguente formula gaussiana prodotto con $n^2/2+O(n)$ nodi

$$\iint_{\mathcal{S}} f(x,y)\,dx\,dy = I_n(f) = \sum_{j=1}^{n+k+1}\sum_{i=1}^{\lceil\frac{n+h+1}{2}\rceil} W_{ij}f(x_{ij},y_{ij}), \quad \forall f\in\mathbb{P}_n^2, \quad (1.2)$$

dove con $\mathbb{P}_n^2$ *si denota lo spazio dei polinomi bivariati di grado totale* n*, con* $h = 0$ *se* $u_i = 0$*,* $i = 0, 1, 2$*, e* $h = 1$ *altrimenti, mentre* $k = 0$ *se* $u_1 = u_2 = 0$ *e* $v_i = 0$*,* $i = 1, \ldots, 4$*,* $k = 1$ *se* $v_3 = v_4 = 0$ *e almeno uno tra* u_1, u_2, v_1, v_2 *é diverso da zero, e* $k = 2$ *se* $v_3 \neq 0$ *o* $v_4 \neq 0$*. In* (1.2) *abbiamo*

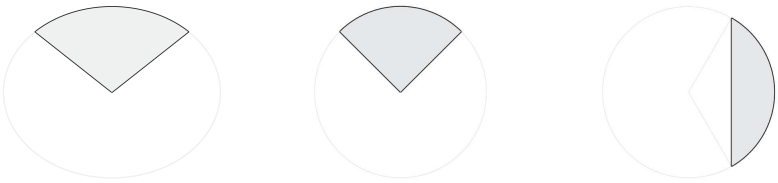
$$(x_{ij}, y_{ij}) = U(t_i^{GL}, \theta_j + \mu), \quad 0 < W_{ij} = |\det JU(t_i^{GL}, \theta_j + \mu)|\omega_i^{GL}\lambda_j,$$

dove $\{(\theta_j+\mu, \lambda_j)\}$ *sono i nodi e i pesi della formula Gaussiana trigonometrica* (1.1) *con grado di precisione* $n + k$ *su* $[\alpha, \beta]$*, e* $\{(t_i^{GL}, w_i^{GL})\}$ *i nodi e i pesi della formula di Gauss-Legendre con grado di precisione* $n + h$ *su* $[0, 1]$*.*

1.2 Settori ellittici ed altri esempi di blending

In questo paragrafo si illustrano esempi di regioni ottenute tramite blending lineare di archi di ellisse. Ci si focalizza, in particolare, nel caso di settori ellittici. Tali regioni saranno di fondamentale importanza per la determinazione di formule di quadratura per poligoni sferici.

La seguente figura mostra tre tipi di regione ottenute tramite blending: un settore ellittico, un settore circolare e un segmento circolare.

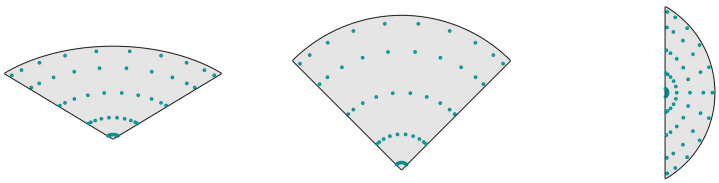


(a) Figura 1a (b) Figura 1b (c) Figura 1c

Figura 1.1: Di seguito si esplicitano solamente i valori $A_1, B_1, C_1, A_2, B_2, C_2$ diversi dal vettore nullo; inoltre si evidenziano l'ellisse o la circonferenza di cui fanno parte i settori mostrati in grigio. In particolare raffiguriamo
(a) settore ellittico con $A_2 = (0.5, 0.0)$, $B_2 = (0.0, 0.3)$ e $\theta \in [\frac{\pi}{4}, \frac{3\pi}{4}]$.
(b) settore circolare con $A_2 = (0.5, 0.0)$, $B_2 = (0.0, 0.5)$ e $\theta \in [\frac{\pi}{4}, \frac{3\pi}{4}]$.
(c) segmento circolare con $C_1 = (0.5, 0.0)$, $A_2 = (0.5, 0.0)$, $B_2 = (0.0, 0.5)$ e $\theta \in [-\frac{\pi}{3}, \frac{\pi}{3}]$.

Utilizzando la proposizione 1.1 e 1.2 si possono ottenere i nodi delle formule di quadratura prodotto sulle tre regioni mostrate sopra.

Si osserva, inoltre, che i nodi e i pesi della formula di quadratura prodotto del paragrafo precedente dipendono dalla scelta di $P(\theta)$ e $Q(\theta)$. Infatti, un esempio presentato in [1], mostra i diversi modi di campionamento dei nodi delle formule nel caso di un segmento circolare al variare di $P(\theta)$ e $Q(\theta)$.



(a) Figura 2a (b) Figura 2b (c) Figura 2c

Figura 1.2: Le figure illustrano i nodi delle formule di quadratura con grado di precisione $\delta = 8$ relativi alle tre regioni appena mostrate, mantenendo il rispettivo ordine.

Di seguito si rappresenta tale esempio assumendo che l'arco del segmento circolare corrisponda all'intervallo angolare $[-\frac{\pi}{3}, \frac{\pi}{3}]$ relativo ad un raggio pari a 0.5 e che il punto medio della corda che unisce le due estremitá dell'arco sia $(0.25, 0)$. Variando $P(\theta)$ e $Q(\theta)$ si possono utilizzare piú parametrizzazioni del tipo mostrato sopra per denotare la medesima regione di interesse.

Elenchiamo qui di seguito alcuni esempi:

- $P(\theta) = (R\cos(\theta), R\sin(\theta))$ e $Q(\theta) = (R\cos(\theta), -R\sin(\theta))$, $\theta \in [0, \frac{\pi}{3}]$;
- $P(\theta) = (R\cos(\beta), R\sin(\theta))$ e $Q(\theta) = (R\cos(\theta), R\sin(\theta))$, $\theta \in [-\frac{\pi}{3}, \frac{\pi}{3}]$;
- $P(\theta) = (0.25, 0)$ e $Q(\theta) = (R\cos(\theta), R\sin(\theta))$, $\theta \in [-\frac{\pi}{3}, \frac{\pi}{3}]$.

La Figura 1.3 rappresenta un esempio di nodi di una formula di quadratura con grado di esattezza 8 nei tre casi considerati.

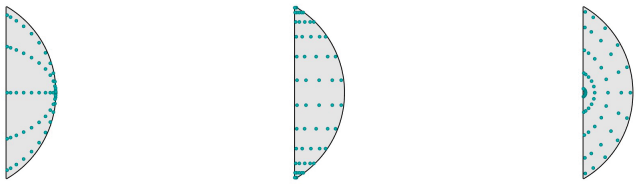


Figura 1.3: Le figure illustrano i nodi delle formule di quadratura con grado di precisione $\delta = 8$ relativi alle tre parametrizzazioni elencate nel rispettivo ordine.

Infine, ci soffermiamo sul caso generale di un settore ellittico. Utilizzando le considerazioni precedenti, un settore ellittico si puo' ottenere come blending lineare tra l'origine ed un arco di ellisse centrato in $(0, 0)$.

Le due curve che delimitano una tale regione sono quindi:

$$P^*(\theta) = (0, 0), \quad Q^*(\theta) = A_2\cos(\theta) + B_2\sin(\theta). \tag{1.3}$$

Con la stessa notazione dell'enunciato 1.2, si ha $u_1 = u_2 = 0$ e $v_1 = v_2 = v_3 = v_4 = 0$ e quindi per la proposizione $h = 1$ e $k = 0$.

Da $|\det JU(t_i^{GL}, \theta_j + \mu)| = |t_i^{GL} u_0 + v_0|$, si ricava

$$\iint_{\mathcal{S}} f(x,y)\,dx\,dy = I_n(f) = \sum_{j=1}^{n+1} \sum_{i=1}^{\lceil \frac{n+2}{2} \rceil} |t_i^{GL} u_0 + v_0| w_i^{GL} \lambda_j f(U(t_i^{GL}, \theta_j + \mu)), \tag{1.4}$$

per ogni $f \in \mathbb{P}_n^2$.

1.3 Quadratura su triangoli sferici

Dati tre punti distinti $V_1, \ldots, V_3 \in \mathbb{S}^2$ e posto $V_0 = V_3$ un *triangolo sferico* con vertici $\{V_k\}_{k=1,\ldots,3}$ é la regione $\mathcal{T} \subset \mathbb{S}^2$ la cui frontiera é determinata dagli archi $\{\gamma_k\}_{k=0,\ldots,2}$, dove γ_k é geodetica che unisce, tenendo conto dell'orientamento, V_k con V_{k+1}, $k = 0, 1, 2$.

Talvolta si dice che ogni arco orientato γ_k sta un *grande cerchio*, ovvero nell'intersezione tra un certo piano contenente l'origine e la sfera.

Avendo quale proposito il calcolo di formule di quadratura con un certo grado di precisione su un triangolo sferico $\mathcal{T}$, non si perde di generalità qualora si assuma che $\mathcal{T}$ abbia il proprio baricentro che coincida con il polo nord della sfera ovvero $\mathcal{T} = ABC$ con $(A+B+C)/\|A+B+C\|_2 = (0,0,1)$.

Nel caso in cui il triangolo sferico $\mathcal{T}$ non soddisfi la condizione riguardante il baricentro, basta considerare una rotazione R che determini un triangolo $\mathcal{T}^*$ con tali proprietá e una volta determinati i nodi di quadratura $\{P_k^*\}$, considerare l'insieme di nodi $\{P_k\}$ in cui $P_k = R^{-1} P_k^*$, senza cambiare i corrispettivi pesi.

In questo lavoro consideriamo il caso di triangoli sferici completamente contenuti nell'emisfero superiore che, quindi, non intersechino l'equatore. Questa richiesta non risulta restrittiva in quanto ogni triangolo puó essere suddiviso in triangoli sferici, ognuno contenuto strettamente in una semisfera, ottenendo, in caso, una formula composta.

Sia ora $g(x,y) = \sqrt{1 - x^2 - y^2}$ e

$$\phi(x,y) = (x, y, g(x,y)) = (x, y, \sqrt{1 - x^2 - y^2}) \tag{1.5}$$

la mappa che proietta ogni punto del triangolo sferico $\mathcal{T}$ sul piano $z = 0$.

Il dominio piano $\Omega := \phi(\mathcal{T})$ corrisponde a un poligono curvilineo con lati che sono archi di ellisse.

L'integrale di superficie di una integranda $f\colon \mathcal{T} \to \mathbb{R}$, che supporremo continua, é dato da

$$\int_{\mathcal{T}} f(x,y,z)d\sigma = \int_{\Omega} f(x,y,g(x,y))\frac{1}{g(x,y)}dxdy \tag{1.6}$$

Si nota che:

$$\Omega = \bigcup_{i=1}^{3} \mathcal{S}_i \tag{1.7}$$

con $\mathcal{S}_i$ settori di ellisse ottenuti congiungendo $(0,0,0)$ con la proiezione dei vertici di $\mathcal{T}$ in $\{z=0\}$. In particolare gli $\mathcal{S}_i$ sono dati dalla proiezione sul piano $\{z=0\}$ di settori circolari con centro in $(0,0,0)$ ed arco coincidente con uno dei lati curvilinei di $\mathcal{T}$.

Ció suggerisce che le formule si possano derivare da altre giá note per i settori di ellisse e circolari, come si è visto nel paragrafo precedente.

Perció l'integrale precedentemente esplicitato si puó riscrivere come somma degli integrali nei singoli settori.

Si ottiene quindi

$$\int_{\mathcal{T}} f(x,y,z)d\sigma = \sum_{i=1}^{3} \int_{\mathcal{S}_i} f(x,y,g(x,y))\frac{1}{g(x,y)}dxdy \tag{1.8}$$

Determinati i nodi $\{(x_{i,j},y_{i,j})\}_j$ e i pesi $\{w_{i,j}\}_j$ di una formula di quadratura in $\mathcal{S}_i$, $\{(x_{i,j},y_{i,j},g(x_{i,j},y_{i,j})\}_{i,j}$ e $\{w_{i,j}/g(x_{i,j},y_{i,j})\}_{i,j}$ sono rispettivamente i nodi e i pesi positivi della formula di quadratura in $\mathcal{T}$.

Analizziamo ora la funzione integranda su un settore $\mathcal{S}_i$.

Sia $\mathbb{P}^3_n$ lo spazio dei polinomi trivariati di grado totale minore o uguale a n.

Se $f \in \mathbb{P}^3_n$ allora $f(x,y,g(x,y)) \in \mathbb{P}^3_n(\mathbb{S}^2)$, ossia lo spazio $\mathbb{P}^3_n$ ristretto alla superficie sferica.

Si nota facilmente che $F = \frac{f(x,y,g(x,y))}{g(x,y)}$ in generale non é un polinomio. Si consideri, infatti, un insieme di generatori di $\mathbb{P}^3_n(\mathbb{S}^2)$, ossia dove $z = g(x,y)$:

$$f(x,y,g(x,y)) = x^\alpha y^\beta g(x,y)^\gamma \quad \text{t.c. } \alpha+\beta+\gamma \leq n$$

Poniamo

$$F(x,y) := x^\alpha y^\beta g(x,y)^{\gamma-1}$$

Se

- γ é *dispari*, si ha $F(x,y) = x^\alpha y^\beta (1-x^2-y^2)^{\frac{\gamma-1}{2}}$, perció $F \in \mathbb{P}^2_{n-1}$

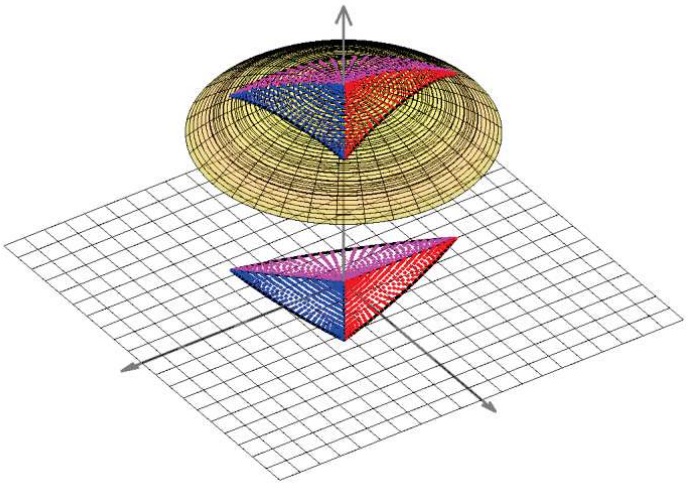


Figura 1.4: I nodi delle formule di quadratura per i tre settori ellittici sul piano $z = 0$ e i corrispondenti nodi sul triangolo sferico.

- γ é *pari*, si ha $F(x,y) = \frac{1}{\sqrt{1-x^2-y^2}} x^\alpha y^\beta (1-x^2-y^2)^{\frac{\gamma}{2}}$, perció $F \in \frac{1}{g}\mathbb{P}_n^2$.

Questo fatto comporta quindi che le integrande

$$F(x,y) = \frac{f(x,y,g(x,y))}{g(x,y)}$$

in generale non sempre sono polinomi bivariati, qualora f sia un polinomio trivariato.

Nel paragrafo successivo si analizza come ottenere una formula di quadratura per triangoli sferici con grado di esattezza (numerico) n utilizzando delle formule per settori ellittici con grado di precisione m.

1.4 Quasi esattezza delle formule in $\mathbb{P}_n(\mathbb{S}^2)$

Prima di procedere nella trattazione, ricordiamo cosa si intende con grado di precisione di una formula algebrica.

Definizione 1.1 *Una formula*

$$\int_\Omega f(x)w(x)d\Omega \approx \sum_{i=1}^{N} w_i f(x_i),$$

con w funzione peso in Ω e x_i e w_i rispettivamente i nodi e i pesi della formula, ha grado di esattezza *almeno n se e solo se é esatta per tutti i polinomi di grado totale inferiore o uguale a n ovvero*

$$\int_\Omega p_n(x)w(x)d\Omega = \sum_{i=1}^{N} w_i p_n(x_i), \quad p_n \in \mathbb{P}_n.$$

Inoltre ha grado di esattezza esattamente n se la formula é esatta per ogni $p_n \in \mathbb{P}_n$ ed esiste $p_{n+1} \in \mathbb{P}_{n+1}$ tale per cui non vale l'uguaglianza.

Nel paragrafo precedente si era giunti alla riscrittura del problema dell'integrazione su un triangolo sferico con la formula (1.8). Volendo ora integrare su dei settori ellittici $\mathcal{S}_i$, notiamo che, utilizzando delle formule di quadratura con grado di esattezza n per questi domini planari, non possiamo ottenere propriamente delle formule con grado di esattezza n per $f \in \mathbb{P}_n(\mathbb{S}_2)$.

Questo avviene perché se $f \in \mathbb{P}_n(\mathbb{S}_2)$ in generale $f(x, y, g(x, y))/g(x, y)$, la funzione integranda per i domini $\mathcal{S}_i$, non è un polinomio, come visto al termine del paragrafo precedente. Perció, utilizzando una formula con grado di esattezza n per $\mathcal{S}_i$ non si ottiene una formula con lo stesso grado di precisione per $\mathcal{T}$.

Per sopperire all'esigenza di trovare una formula con grado di esattezza n, si puó optare nell'approssimare $1/g(x, y)$, a meno di una certa tolleranza ϵ, con con un polinomio di grado m. Quindi, dato ϵ, si intende trovare p_ϵ di grado $m(\epsilon)$ tale che $|p_\epsilon(x, y) - 1/g(x, y)| \leq \epsilon\, 1/|g(x, y)|$ per ogni $(x, y) \in \mathcal{S}_i$. In questo modo si deduce dalla disuguaglianza precedente che fp_ϵ approssima $f(x, y, g(x, y))/g(x, y)$ a meno di una certa tolleranza ϵ.

Dunque, considerando una formula di quadratura su $\mathcal{S}_i$ con nodi $\{(x_i, y_i)\}$ e pesi positivi $\{w_i\}$ e grado di precisione $m(\epsilon)+n$, essa é *quasi* esatta di grado n per la funzione integranda $f(x, y, g(x, y))/g(x, y)$ con $f \equiv p \in \mathbb{P}_n^3$.

A tal proposito, siano

- $\tilde{p}(x, y) := p(x, y, g(x, y))$, $p \in \mathbb{P}_n(\mathbb{S}_2)$,
- $I_i(h) = \int_{\mathcal{S}_i} h(x, y)dxdy$, $h \in \mathcal{C}(\mathcal{S}_i)$
- $S_i(h) = \sum_{j=1}^{N} w_j h(x_j, y_j)$ dove (x_j, y_j) é il generico nodo della formula di quadratura sul settore sferico $\mathcal{S}_i$ sopra menzionata, con grado $m(\epsilon)+n$,
- $|p_\epsilon(x, y) - 1/g(x, y)| \leq \epsilon\, 1/|g(x, y)|$ per ogni $(x, y) \in \mathcal{S}_i$.

Ne consegue facilmente che per $h = \tilde{p}/g$ abbiamo

$$\begin{aligned} |I_i(h) - S_i(h)| &= |I_i(\tilde{p}/g) - I_i(\tilde{p}\, p_\epsilon) + I_i(\tilde{p}\, p_\epsilon) - S_i(\tilde{p}/g)| \\ &= |I_i(\tilde{p}/g) - I_i(\tilde{p}\, p_\epsilon) + S_i(\tilde{p}\, p_\epsilon) - S_i(\tilde{p}/g)| \\ &\leq |I_i(\tilde{p}/g) - I_i(\tilde{p}\, p_\epsilon)| + |S_i(\tilde{p}\, p_\epsilon) - S_i(\tilde{p}/g)| \\ &\leq I_i(|\tilde{p}/g - \tilde{p}\, p_\epsilon|) + S_i(|\tilde{p}\, p_\epsilon - \tilde{p}/g|) \\ &\leq \epsilon(I_i(|h|) + S_i(|h|)). \end{aligned} \tag{1.9}$$

Essendo $h = \frac{\tilde{p}}{g}$ continua in $\mathcal{S}_i$, esiste finito $M_i = \max_{(x,y)\in\mathcal{S}_i} |h(x,y)|$. Denotata con $\mu(\mathcal{S}_i)$ la misura di Lebesgue del settore ellittico $\mathcal{S}_i$ concludiamo che in virtú del *quasi* grado di esattezza della formula

$$|I_i(h) - S_i(h)| \leq \epsilon M_i \left(\mu(\mathcal{S}_i) + \sum_{j=1}^{N} w_j \right) \approx 2\epsilon M_i\ \mu(\mathcal{S}_i).$$

Da questa si ricava immediatamente l'errore della formula di quadratura sul triangolo sferico.

Nota 1.1 *Si osservi che da (1.9) se* $0 \neq I_i(|h|) \approx S_i(|h|)$ *allora* $|I_i(h) - S_i(h)|/|I_i(h)|$ *é al piú (circa)* 2ϵ *e quindi possiamo avere una stima dell'errore relativo di quadratura compiuto in ogni settore ellittico.*

Ritornando alla formula a cui si era giunti alla fine del paragrafo precedente 1.8, si ha che dati $\{(x_i, y_i)\}_i$ e pesi positivi $\{w_i\}_i$ di una formula di quadratura di grado di precisione $n + m(\epsilon)$ per l'integrazione sul dominio Ω in 1.7, si ottiene una formula di quasi esattezza n sul dominio $\mathcal{T}$ con nodi $\{(x_i, y_i, g(x_i, y_i)\}_i$ e pesi positivi $\{w_i/g(x_i, y_i)\}_i$.

Approfondiamo ora come si ricava $m(\epsilon)$ a partire dal dominio Ω. Si ricorda che $g(x, y) = \sqrt{1 - (x^2 + y^2)}$ e si nota che:

$$0 \leq x^2 + y^2 \leq \rho = \max\{\|\hat{A}\|_2^2, \|\hat{B}\|_2^2, \|\hat{C}\|_2^2\} < 1, \quad (x, y) \in \Omega,$$

con $\hat{A}$, $\hat{B}$, $\hat{C}$ le proiezioni dei vertici di $\mathcal{T}$ sul piano $z = 0$.

Per concludere, studiamo $g(t) = 1/\sqrt{1-t}$ per $t \in [0, \rho]$. In particolare, denotiamo con $v(\epsilon, \rho)$ il grado del polinomio necessario per approssimare intorno alla precisione di macchina $g(t)$ su tutto l'intervallo $[0, \rho]$. Una volta determinato il grado $v(\epsilon, \rho)$, basta considerare $m(\epsilon) = 2v(\epsilon, \rho)$ poiché il polinomio trovato in precedenza deve essere composto con la trasformazione $t = x^2 + y^2$.

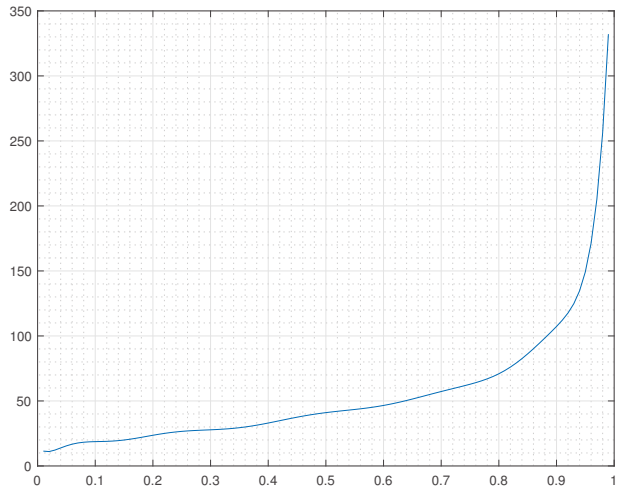


Figura 1.5: grafico del grado di $m(\epsilon)$ in funzione di ρ

Stimiamo il grado del polinomio usando un approccio numerico.

Un primo approccio consiste nell'utilizzo del pacchetto Chebfun di Matlab, che permette di approssimare alla precisione di macchina una funzione f in input con un polinomio interpolante in un certo numero di nodi di Chebyshev. Noto il grado di tale polinomio, si ottiene una formula *numericamente* esatta al grado di precisione richiesto.

Data una tolleranza ϵ, si intende verificare quale grado dell'interpolante di Chebyshev p_{cheb} verifichi $|p_{cheb}(t) - 1/\sqrt{1-t}| \leq \epsilon 1/\sqrt{1-t}, \quad t \in [0, \rho]$ per un opportuno campionamento nell'intervallo. Una volta trovato il minimo n che verifica la condizione precedente, si pone $m(\epsilon) = 2n$.

Infine, evidenziamo che il numero totale di nodi della formula é circa $\frac{3}{2}(n+m(\epsilon))^2$, il quale puó essere molto grande se ρ tende ad 1. Non conviene, peró, avere un numero di nodi elevato in una formula di quadratura a causa del possibile costo computazionale eccessivo e degli errori di arrotondamento. Si puó, quindi, porre rimedio utilizzando delle formule di compressione per trasformare la formula di quadratura precedente in un'altra, sempre con pesi positivi ma numero di nodi piú contenuto.

Questo é possibile, ad esempio, eseguendo la compressione di Caratheodory-Tchakaloff. La formula ottenuta tramite compressione ha come nodi un sottoinsieme dell'insieme di nodi della formula precedente la cui cardinalitá non

supera $\dim(\mathbb{P}_n^3(\mathcal{T})) = \dim(\mathbb{P}_n^3(\mathbb{S}^2)) = (n+1)^2$.

1.5 Formule di quadratura per poligoni sferici

In questa sezione presentiamo quanto noto in letteratura per quanto riguarda la determinazione di formule di quadratura su poligoni sferici $\mathcal{P}$ che abbiano, almeno *numericamente*, un grado di esattezza "δ" precisato dall'utente.

Dati n punti distinti $V_1, \ldots, V_n \in \mathbb{S}^2$ e posto $V_n = V_0$, un poligono sferico é la regione $\mathcal{P} \subset \mathbb{S}^2$ la cui frontiera $\delta\mathcal{A}$ é determinata dagli archi $\{\gamma_k\}_{k=0,\ldots,n-1}$, dove γ_k é geodetica che unisce, tenendo conto dell'orientamento, V_k con V_{k+1}, $k = 0, \ldots, n-1$.

In ambito bivariato, una strategia per determinare tali formule su una regione poligonale piana $\mathcal{P}$ concerne nel

- suddividerla in triangoli $\{\mathcal{T}_i\}_{i=1,\ldots,m}$, meglio se in un numero minimo,
- calcolare una formula di quadratura con grado di esattezza "δ" su ognuno dei triangoli $\mathcal{T}_i$ della suddivisione, cosí da avere una formula composta su $\mathcal{P}$.

Similmente, per determinare delle formule di quadratura su dei poligoni sferici $\mathcal{P}$ conviene per prima cosa determinare una suddivisione in triangoli sferici $\mathcal{T}_i$, ovvero

$$\mathcal{P} = \bigcup_{i=1}^{N} \mathcal{T}_i$$

con $\text{interno}(\mathcal{T}_i) \cap \text{interno}(\mathcal{T}_j) = \emptyset$ qualora $i \neq j$.

Per effettuare la triangolazione, si procede come segue, supponendo il poligono contenunto in una caps di altezza minore di 1 e centro C.

1. si calcola il centro C della cap sopramenzionata e si determina una rotazione R cosí da mapparlo al Polo Nord;
2. si determinano i vertici ruotati $\tilde{V}_i = R(V_i)$, $i = 1, \ldots, n$;
3. si mappano i punti $\tilde{V}_i$ sul piano equatoriale mediante la proiezione Π, ottenendo $\bar{V}_i = \Pi(\tilde{V}_i)$, $i = 1, \ldots, n$;
4. mediante le routines di `polyshape` si calcola una triangolazione minima in $n-2$ triangoli T_j, ovvero un set di punti $\bar{U}_i$ di $\Pi(R(\mathcal{P}))$ che comprende i vertici $\bar{V}_i$ e le loro connessioni cosí da determinare T_j, $j = 1, \ldots, n-2$;

5. mediante le inverse di Π e R si determinano i triangoli sferici $\mathcal{T}_j$, $j = 1, \ldots, n-2$ che compongono la triangolazione del poligono sferico.

Quindi, si ricava dall' additivitá dell operatore di integrazione

$$\int_{\mathcal{P}} f \, d\mathcal{P} = \sum_{i=1}^{N} \int_{\mathcal{T}_i} f \, d\mathcal{T}_i.$$

Per quanto visto precedentemente sappiamo, quindi, fornire delle formule di quadratura di quasi esattezza δ per triangoli sferici completamente contenuti, a meno di una certa rotazione, nell'emisfero superiore di $\mathbb{S}^2$.

Capitolo 2

Algoritmo di quadratura adattiva su poligoni sferici

In questo breve capitolo, data una funzione f introduciamo un algoritmo adattivo per l'approssimazione dell'integrale definito di f su un poligono sferico.

Routines di questo tipo esistono per varie classiche regioni. In Matlab, ad esempio, `integral` calcola l'integrale qualora il dominio sia un intervallo, mentre `integral2` permette di ottenere un simile risultato su un quadrato e `integral3` su un cubo.

I relativi codici sono basati sulle routines di L. Shampine descritte in [3] e [2], in cui l'autore sfrutta la vettorializzazione di Matlab, anche mostrando come si possano utilizzare per domini come il disco, riscrivendo il problema in coordinate polari.

Un'ottima alternativa é prevista nel caso univariato dalla routine `sum` dell'ambiente Chebfun, che permette l'integrazione multivariata anche sul disco mediante `diskfun`, sul quadrato con `Chebfun2` e perfino nella sfera con `spherefun`.

Routines di questo tipo non sono al momento presenti su poligoni piani e regioni della sfera quali poligoni sferici.

A tal proposito, sia $f\colon \mathcal{P} \to \mathbb{R}$ un'integranda continua, con $\mathcal{P}$ un poligono sferico contenuto in una cap con altezza strettamente minore di 1. Sia

$$I = \int_{\mathcal{P}} f(\mathcal{P}) d\mathcal{P}.$$

L'intento di questo capitolo é approssimare numericamente I a meno di una tolleranza assoluta `atol` e una relativa `rtol` fissata dall'utente. L'algorit-

mo proposto é di tipo adattivo e fondamentalmente suddivide nuovamente il dominio con triangolazione sferica nei punti in cui l'approssimazione dell'integrale risulta peggiore.

Esso si divide in due fasi:

1. triangolazione sferica minimale del dominio di partenza $\mathcal{P}$,
2. quadratura adattiva.

Il procedimento termina una volta raggiunta la tolleranza indicata o se le iterazioni svolte durante la fase di quadratura adattiva superano un numero massimo fissato; in questo caso viene mostrato un messaggio di mancata convergenza.

La suddivisione minima in triangoli sferici

$$\mathcal{P} = \bigcup_{i=1}^{N} \mathcal{T}_i, \quad \text{con interno}(\mathcal{T}_i) \cap \text{interno}(\mathcal{T}_j) = \emptyset \quad \forall i \neq j$$

si ottiene come discusso al capitolo precedente.

Conclusasi, quindi, la prima parte dell'algoritmo, si prosegue con la fase di quadratura adattiva. Inizialmente, si calcola numericamente l'integrale su $\mathcal{P}$ sfruttando le formule di quadratura con grado di esattezza numerica δ, viste per i triangoli sferici nel primo capitolo. Nell'implementazione alla fine del documento si utilizza $\delta = 4$. Denotando con $I_{\mathcal{T}_i}(f) = \int_{\mathcal{T}_i} f$ e con $S_{\mathcal{T}_i}(f)$ la formula di quadratura applicata a $\mathcal{T}_i$ si ha che

$$I_{\mathcal{P}}(f) = \int_{\mathcal{P}} f = \sum_{i=1}^{N} I_{\mathcal{T}_i}(f) \approx \sum_{i=1}^{N} S_{\mathcal{T}_i}(f) := S^L_{\mathcal{P}}(f),$$

dove con la variabile $S^L_{\mathcal{P}}(f)$ si vuole indicare un'approssimazione di $I_{\mathcal{P}}(f)$ ad un livello inferiore di raffinamento del dominio. Successivamente si suddivide ulteriormente il dominio triangolando ogni $\mathcal{T}_i$ con quattro triangoli sferici.

Si supponga V_1, V_2, V_3 siano i vertici di $\mathcal{T}_i$ e si denoti con $M_{i,j}$ il punto medio del lato V_iV_j. La divisione di $\mathcal{T}_i$ é costituita da:

- $\mathcal{T}_{i,1}$ con vertici $V_1, M_{1,2}, M_{3,1}$
- $\mathcal{T}_{i,2}$ con vertici $V_2, M_{2,3}, M_{1,2}$
- $\mathcal{T}_{i,3}$ con vertici $V_3, M_{3,1}, M_{2,3}$
- $\mathcal{T}_{i,4}$ con vertici $M_{1,2}, M_{2,3}, M_{3,1}$

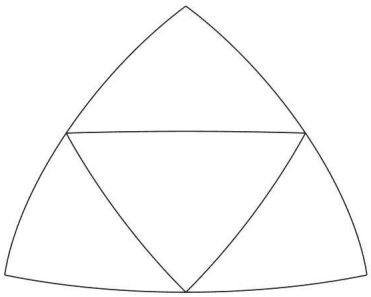


Figura 2.1: Esempio di suddivisione di un triangolo sferico di vertici $V_1 = (-0.5000, -0.5000, 0.7071), V_2 = (0.0000, 0.7071, 0.7071), V_3 = (0.5000, -0.5000, 0.7071)$ in quattro triangoli sferici usando i punti medi. Vista proiettata sul piano XY.

Il numero di triangoli sferici costituenti la divisione del dominio si é quindi quadruplicato.

É opportuno introdurre la seguente notazione:

- $\diamond$ $S^L_{\mathcal{T}_i}(f)$ per indicare l'approssimazione di $I_{\mathcal{T}_i}(f)$ utilizzando le formule di quadratura con grado di esattezza $\delta = 4$ in $\mathcal{T}_i$;
- $\diamond$ $S^H_{\mathcal{T}_i}(f) := \sum_{j=1}^{4} S^L_{\mathcal{T}_{i,j}}(f) \approx I_{\mathcal{T}_i}(f)$, dove la formula di quadratura viene applicata singolarmente a domini piú ristretti e quindi ci si aspetta una migliore stima di $I_{\mathcal{T}_i}(f)$.

I dati acquisiti si possono organizzare in due insiemi: uno riguardante i triangoli sferici del livello inferiore L di raffinamento del dominio $\mathcal{T}_i$, e l'altro quello superiore H con $\mathcal{T}_{i,j}$. I contenuti sono i seguenti:

- L: una `lista L` contenente $\mathcal{T}_i$, ossia i triangoli generanti, e i valori $S^L_{\mathcal{T}_i}(f), S^H_{\mathcal{T}_i}(f) \quad i = 1, \ldots, N$
- H: una `lista H` contenente $\mathcal{T}_{i,j}$, ossia i triangoli generati tramite la suddivisione, e i valori $S^L_{\mathcal{T}_{i,j}}(f) \quad i = 1, \ldots, N$ e $j = 1, \ldots, 4$

Valutando ora l'approssimazione numerica di $I_{\mathcal{P}}(f)$ con la divisione del dominio in $4N$ triangoli sferici si ha

$$I_{\mathcal{P}}(f) = \sum_{i=1}^{N} \sum_{j=1}^{4} I_{\mathcal{T}_{i,j}}(f) \approx \sum_{i=1}^{N} \sum_{j=1}^{4} S^L_{\mathcal{T}_{i,j}}(f) = \sum_{i=1}^{N} S^H_{\mathcal{T}_i}(f) := S^H_{\mathcal{P}}(f).$$

Segue, poi, la seguente verifica

$$|S^L_{\mathcal{P}}(f) - S^H_{\mathcal{P}}(f)| \leq \max(\texttt{atol}, \texttt{rtol}|S^H_{\mathcal{P}}(f)|), \tag{2.1}$$

dove con `atol` e `rtol` ci si riferisce rispettivamente ad una tolleranza assoluta e relativa fissata dall'utente. Se la valutazione risulta positiva significa che, presumibilmente, iterando nuovamente la divisione del dominio non si rivelano significativi cambiamenti nell'approssimazione numerica dell'integrale. In questo caso, l'algoritmo termina producendo, come output, il risultato $S^H_{\mathcal{P}}(f)$. Alternativamente, se non si verifica la condizione si itera la seguente procedura:

1. Si determina il triangolo sferico $\mathcal{T}^* = \mathcal{T}_i$ nel quale la stima dell'errore $|S^L_{\mathcal{T}_i}(f) - S^H_{\mathcal{T}_i}(f)|$ é maggiore.

2. Si aggiorna la `lista L`: $\mathcal{T}^*$ viene rimosso dalla lista e sostituito con $\mathcal{T}^*_1 := \mathcal{T}_{i,1}, \mathcal{T}^*_2 := \mathcal{T}_{i,2}, \mathcal{T}^*_3 := \mathcal{T}_{i,3}, \mathcal{T}^*_4 := \mathcal{T}_{i,4}$ per i quali era giá stato calcolato $S^L_{\mathcal{T}^*_i}(f) = S^L_{\mathcal{T}_{i,j}}(f)$.

3. Si aggiorna la `lista H`: vengono rimossi $\mathcal{T}^*_i$, $i = 1, \ldots, 4$, e ciascuno di essi viene sostituito con quattro triangoli sferici $\mathcal{T}^*_{i,j}$, $i = 1, \ldots, 4$, $j = 1, \ldots, 4$ generati nel modo visto sopra.

4. Si determina, come visto precedentemente, $S^H_{\mathcal{T}^*_i}(f)$, $i = 1, \ldots, 4$.

A questo punto si hanno tutti i dati necessari per poter calcolare i nuovi valori $S^L_{\mathcal{P}}(f)$ e $S^H_{\mathcal{P}}(f)$ e si puó testare nuovamente (2.1) e procedere come in precedenza.

Nel caso in cui la condizione (2.1) non sia soddisfatta dopo aver prodotto un numero considerevole di triangoli sferici, l'algoritmo fallisce e produce, dopo un avviso di mancata convergenza, $S^L_{\mathcal{P}}(f)$ ed una stima dell'errore di approssimazione di $I_{\mathcal{P}}(f)$.

Capitolo 3

Test numerici

In quest'ultimo capitolo, testiamo l'algoritmo di cubatura adattiva su tre diversi poligoni sferici $\mathcal{P}$.

In particolare, su ognuno di loro approssimiamo numericamente l'integrale di tre funzioni integrande continue in $\mathcal{P}$. Per diversificare, consideriamo funzioni che hanno diversi gradi di regolaritá in modo tale da evidenziare il cambiamento nella suddivisione in triangoli sferici del dominio.

Di seguito notiamo che, nei punti di maggiore irregolaritá di una funzione, ad esempio quelli di non derivabilitá, la suddivisione in triangoli sferici aumenta negli intorni di tali zone. A supporto degli esempi riportiamo delle immagini e delle tabelle contenenti dati per il confronto sulle medesime regioni.

Quali poligoni sferici consideriamo

- una approssimazione dell'Australia senza isole minori e Tasmania, tracciata utilizzando un campionamento discreto di coordinate,
- una approssimazione di una cardioide *sferica*,
- un decagono sferico.

Le funzioni test che prendiamo in esame sono le seguenti

- $f_1(x, y, z) := \exp(-h(x, y, z))$;
- $f_2(x, y, z) := \exp(-h(x, y, z)) \sin^2(10y + 20z) \cos^2(10x + 20z)$;
- $f_3(x, y, z) := ((x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2)^{1/2}$;
- $f_4(x, y, z) := ((x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2)^{1/4}$;

dove (x_0, y_0, z_0) indica il baricentro del poligono sferico su cui si sta valutando la funzione, allora $h(x, y, z) := (x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2$ e (x_i, y_i, z_i) corrispondono alle coordinate di uno dei vertici V_i del poligono sferico.

Le funzioni f_1 e f_2 sono analitiche con f_2 oscillante mentre f_3 e f_4 non sono derivabili rispettivamente nel baricentro e in $V_i \in \partial\mathcal{P}$.

Si prospetta, quindi, una suddivisione piú fitta del dominio nell'intorno di tali punti.

A seguire, in ogni paragrafo trattiamo un diverso dominio valutando per ognuno di essi f_1, f_2, f_3,f_4. Dato un dominio $\mathcal{P}$ e una funzione f_i evidenziamo

- ◇ il numero di vertici di $\mathcal{P}$,
- ◇ il numero di vertici della triangolazione minimale di $\mathcal{P}$,
- ◇ il numero di nodi di una formula di quadratura (non adattiva) con grado di esattezza pari a 40 su $\mathcal{P}$,
- ◇ I_{num} l'integrale numerico ottenuto su $\mathcal{P}$ con la formula di quadratura accennata.

In seguito viene riportata una tabella in cui confrontiamo i risultati ottenuti dall'algoritmo con diverse tolleranze (assumendo che la tolleranza assoluta coincida con quella relativa). In tale tabella esplicitiamo

- `tol`: la tolleranza assoluta,
- `it.`: il n. di iterazioni dell'algoritmo di cubatura adattiva,
- `triang.`: il n. totale di triangoli in cui si é suddiviso il dominio dopo aver terminato l'algoritmo adattivo,
- $\texttt{I}_{\text{a}}$: l'approssimazione dell'integrale di f_i su $\mathcal{P}$ fornito dall'algoritmo,
- `time`: il tempo di esecuzione dell'algoritmo,
- `AE`: l'errore assoluto compiuto dall'algoritmo,
- `RE`: l'errore relativo compiuto dall'algoritmo.

I tests sono stati eseguiti su un computer con processore *12th Gen Intel(R) Core(TM) i5-1235U 1.30 GHz* e 16GB di RAM. La versione di Matlab utilizzata é R2023b.

3.1 Australia

Il dominio ha, quale frontiera, un'approssimazione del profilo costiero australiano utilizzando un campionamento discreto di punti.

In particolare,

- ha 169 vertici e la triangolazione minimale si compone di 167 triangoli sferici;
- il baricentro del dominio é il punto $(-0.63, 0.67, -0.39)$;
- il vertice in cui si trova la singolaritá di f_4 é $(-0.56, 0.64, -0.52)$.

Nella Figura 3.2 mostriamo le suddivisioni ottenute per f_1, f_2, f_3, f_4 qualora si richiedano tolleranze assolute `atol`=10^{-12} e relative `rtol`=10^{-12}. Risulta particolarmente interessante l'addensarsi delle triangolazioni nei pressi delle singolaritá di f_3 (baricentro) e di f_4 (vertice scelto).

Nei grafici delle triangolazioni adattive é presente una colorazione aggiuntiva che varia a seconda dei valori che f_i assume su di esso. Per ogni immagine interessata, a destra é presente una barra dei colori come legenda. Una variazione di colori piú ravvicinata nello spazio é sintomo di forti oscillazioni dei valori di una funzione, come nel caso della funzione oscillante f_2.

Di seguito le tabelle riportano i dati raccolti rispettivamente per f_1, f_2, f_3 e f_4 valutando in ognuna la stessa funzione con tolleranze diverse, ovvero `atol`=`rtol` pari a 10^{-6}, 10^{-9}, 10^{-12}.

Osserviamo come, al crescere delle tolleranze richieste, aumenti pure il numero di iterazioni e che, in generale, il tempo richiesto si assesta nell'ordine del secondo.

Valutando f_1 l'algoritmo converge istantaneamente fornendo il risultato alla precisione desiderata in 1 iterazione. L'accuratezza é confermata dagli errori assoluti e relativi con l'integrale di riferimento dell'ordine della precisione di macchina.

In f_2 il numero di iterazioni eseguite é contenuto. In particolare, imponendo una tol= 10^{-12} all'algoritmo adattivo, si ottiene I_{num} a meno della precisione di macchina in 29 iterazioni.

In f_3 sia in 3 che in 21 iterazioni, rispettivamente ralative a una tol= $10^{-9}, 10^{-12}$, l'errore con I_{num} si assesta all'ordine di 10^{-9} per AE e 10^{-8} per RE.

Infine, per f_4, fissata tol$= 10^{-12}$, con 56 iterazioni si ha che I_{a} approssima I_{num} con AE dell'ordine di 10^{-11}; appena un ordine di grandezza inferiore a quello atteso (10^{-12}).

Osserviamo che, in ogni caso, l'algoritmo adattivo converge, in quanto fornisce un risultato con un numero di triangolazioni inferiore a 1000 (massimo numero imposto nell'algoritmo adattivo). In generale, alla diminuzione della tolleranza corrisponde un aumento della precisione di I_{a}.

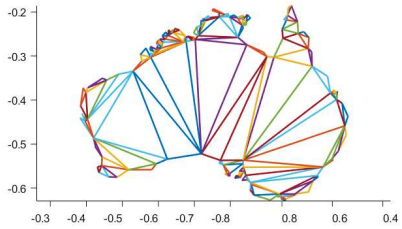
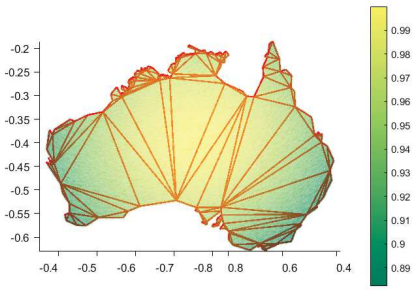
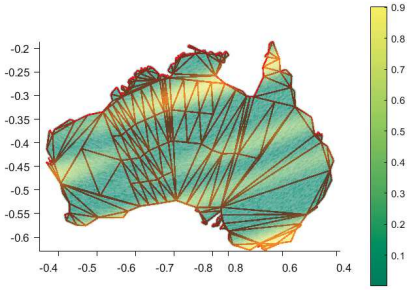


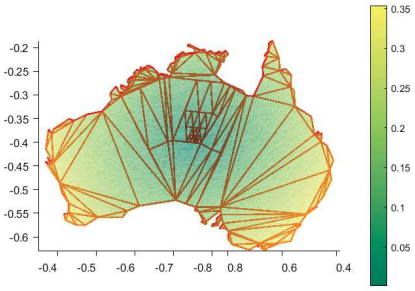
Figura 3.1: La triangolazione minima (proiezione su un piano).



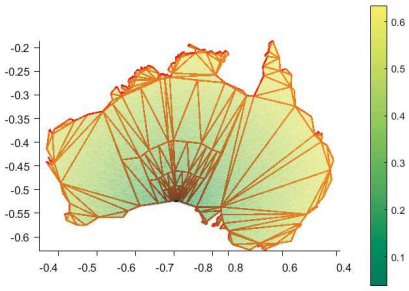
(a) La triangolazione per f_1.



(b) La triangolazione per f_2.



(c) La triangolazione per f_3.



(d) La triangolazione per f_4.

Figura 3.2: In (a), (b),(c) e (d) le triangolazioni eseguite dall'algoritmo adattivo rispettivamente per f_1, f_2, f_3, f_4 per `atol`=`rtol`=10^{-12} (vista planare: view(215,0)).

$I_{\text{num}} = 1.806254323593869e-01$, nodi: 579891						
tol	it.	triang.	I_{a}	time	AE	RE
1e-06	1	167	1.806254323593869e-01	0.73s	5.6e-17	3.1e-16
1e-09	1	167	1.806254323593869e-01	0.78s	5.6e-17	3.1e-16
1e-12	1	167	1.806254323593869e-01	0.70s	5.6e-17	3.1e-16

Tabella 3.1: Indicazione dei parametri rilevanti per i tests sulla funzione f_1.

$I_{\text{num}} = 3.590674580040355e-02$, nodi: 579891						
tol	it.	triang.	I_{a}	time	AE	RE
1e-06	1	167	3.590674560647433e-02	0.82s	1.9e-10	5.4e-09
1e-09	2	170	3.590674521310398e-02	0.69s	5.9e-10	1.6e-08
1e-12	29	251	3.590674580040278e-02	1.12s	7.8e-16	2.7e-14

Tabella 3.2: Indicazione dei parametri rilevanti per i tests sulla funzione f_2.

$I_{\text{num}} = 3.398364433363321e-02$, nodi: 579891						
tol	it.	triang.	I_{a}	time	AE	RE
1e-06	1	167	3.398374406036552e-02	0.83s	9.9e-08	2.9e-06
1e-09	3	173	3.398364250523284e-02	0.85s	1.8e-09	5.4e-08
1e-12	21	227	3.398364191934793e-02	1.02s	2.4e-09	7.1e-08

Tabella 3.3: Indicazione dei parametri rilevanti per i tests sulla funzione f_3.

$I_{\text{num}} = 8.358278208982420e-02$, nodi: 579891						
tol	it.	triang.	I_{a}	time	AE	RE
1e-06	1	167	8.358279590076553e-02	0.85s	1.4e-08	1.6e-07
1e-09	15	209	8.358278322284372e-02	0.91s	1.1e-09	1.36e-08
1e-12	56	332	8.358278202908213e-02	1.40s	6.1e-11	7.27e-10

Tabella 3.4: Indicazione dei parametri rilevanti per i tests sulla funzione f_4.

3.2 Cardioide poligonale

Il dominio considerato si costruisce campionando un certo numero di vertici della cardioide planare di riferimento. A partire da tali vertici, si determina il poligono sferico di interesse utilizzando il sollevamento sulla sfera di questi ultimi.

L'equazione parametrica della cardioide piana corrispondente é

$$\begin{cases} x(\theta) &= \cos(\theta)(1-\cos(\theta)) \\ y(\theta) &= \sin(\theta)(1-\cos(\theta)) \end{cases}$$

con $\theta \in [0, 2\pi]$.

Siano $\{\theta_i\}_{i=1,\dots,33}$ dei punti equispaziati nell'intervallo $[0, 2\pi]$. Siano

$$V_i = \{(x(\theta_i), y(\theta_i), \sqrt{1 - x(\theta_i)^2 - y(\theta_i)^2})\}_{i=1,\dots,32}$$

i vertici del poligono sferico in considerazione, che in qualche senso rappresenta una approssimazione di una cardioide sferica.

Riassumendo alcune caratteristiche del dominio utili alla comprensione delle immagini e delle tabelle

- ha 32 vertici e la triangolazione minimale si compone di 30 triangoli sferici;

- il baricentro del dominio é il punto $(-0.29, 0, 0.96)$;
- il vertice in cui si trova la singolaritá di f_4 é $(0, 0, 1)$.

Nella Figura 3.4 mostriamo le suddivisioni ottenute per f_1, f_2, f_3, f_4 qualora si richiedano tolleranze assolute `atol`$=10^{-12}$ e relative `rtol`$=10^{-12}$. Osserviamo comportamenti simili a quanto visto per il dominio precedente in corrispondenza del nuovo baricentro e del vertice in cui si trova la singolaritá di f_4.

Analogamente all'esempio precedente, f_1 e f_2, essendo analitiche, approssimano I_{a} a meno dell'errore atteso con tol. Notiamo peró che f_2, la funzione oscillante, necessita 693 triangolazioni per terminare l'algoritmo adattivo nel caso di tol$= 10^{-12}$. Tale numero supera addirittura in n. di triangolazioni necessarie a terminare l'algoritmo con tol$= 10^{-12}$ per le funzioni con la singolaritá di tipo radice.

Nel caso di f_3 l'algoritmo esegue un numero contenuto di iterazioni e l'errore con I_{num} si stabilizza all'ordine di 10^{-7} per tutte le tolleranze testate.

Infine, f_4 per tol$= 10^{-12}$ esegue 200 iterazioni con AE e RE dell'ordine di 10^{-9}.

In ognuno dei casi l'algoritmo adattivo converge.

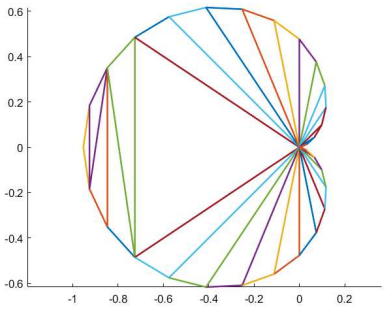
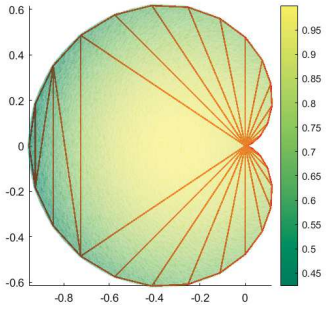
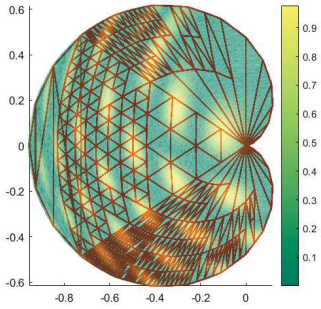


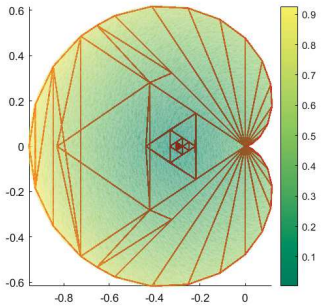
Figura 3.3: La triangolazione minima (proiezione su un piano).



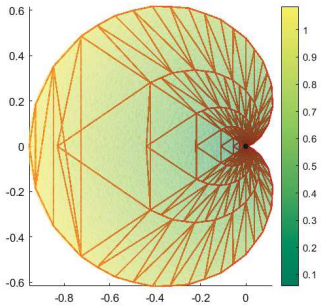
(a) La triangolazione per f_1.



(b) La triangolazione per f_2.



(c) La triangolazione per f_3.



(d) La triangolazione per f_4.

Figura 3.4: In (a), (b),(c) e (d) le triangolazioni eseguite dall'algoritmo adattivo rispettivamente per f_1,f_2, f_3, f_4 per `atol`=`rtol`=10^{-12} (vista planare: view(0,90)).

$I_{\text{num}} = 1.077532507688124e+00$, nodi: 140250						
tol	it.	triang.	I_{a}	time	AE	RE
1e-06	1	30	1.077532507688125e+00	0.25s	1.33e-15	1.24e-15
1e-09	1	30	1.077532507688125e+00	0.16s	1.33e-15	1.24e-15
1e-12	1	30	1.077532507688125e+00	0.16s	1.33e-15	1.24e-15

Tabella 3.5: Indicazione dei parametri rilevanti per i tests sulla funzione f_1.

$I_{\text{num}} = 2.559015739903048e-01$, nodi: 140250						
tol	it.	triang.	I_{a}	time	AE	RE
1e-06	2	33	2.559018535051488e-01	0.25s	2.8e-07	1.09e-06
1e-09	31	120	2.559015742012746e-01	0.81s	2.11e-10	8.24e-10
1e-12	222	693	2.559015739902026e-01	3.4s	1.02e-13	3.99e-13

Tabella 3.6: Indicazione dei parametri rilevanti per i tests sulla funzione f_2.

$I_{\text{num}} = 6.722865750040778e-01$, nodi: 140250						
tol	it.	triang.	I_{a}	time	AE	RE
1e-06	2	33	6.722868857780782e-01	0.2s	3.11e-07	4.62e-07
1e-09	5	42	6.722868847338693e-01	0.22s	3.1e-07	4.61e-07
1e-12	15	72	6.722868848325659e-01	0.4s	3.1e-07	4.61e-07

Tabella 3.7: Indicazione dei parametri rilevanti per i tests sulla funzione f_3.

$I_{\text{num}} = 1.071848894876717e+00$, nodi: 140250						
tol	it.	triang.	I_{a}	time	AE	RE
1e-06	1	30	1.071849030898455e+00	0.22s	1.36e-07	1.27e-07
1e-09	85	282	1.071848894699621e+00	1.83s	1.77e-10	1.65e-10
1e-12	200	627	1.071848893302702e+00	2.85s	1.57e-09	1.47e-09

Tabella 3.8: Indicazione dei parametri rilevanti per i tests sulla funzione f_4.

3.3 Decagono sferico

Il dominio scelto é concavo. I suoi vertici sono

- $V_1 = (0.4045, 0.2939, 0.8660)$
- $V_2 = (0.2676, 0.8236, 0.5000)$
- $V_3 = (-0.1545, 0.4755, 0.8660)$
- $V_4 = (-0.7006, 0.5090, 0.5000)$;
- $V_5 = (-0.5000, 0.0000, 0.8660)$;
- $V_6 = (-0.7006, -0.5090, 0.5000)$
- $V_7 = (-0.1545, -0.4755, 0.8660)$

- $V_8 = (0.2676, -0.8236, 0.5000)$
- $V_9 = (0.4045, -0.2939, 0.8660)$;
- $V_{10} = (0.8660, -0.0000, 0.5000)$

Alcuni particolari importanti per la comprensione delle immagini

- ha 10 vertici e la triangolazione minimale si compone di 8 triangoli sferici;
- il baricentro del dominio é il punto $(0, 0, 1)$;
- il vertice in cui si trova la singolaritá di f_4 é V_1.

Nella Figura 3.6 mostriamo le suddivisioni ottenute per f_1, f_2, f_3, f_4 qualora si richiedano tolleranze assolute `atol`=10^{-12} e relative `rtol`=10^{-12}. Osserviamo comportamenti simili a quanto visto per il dominio precedente in corrispondenza del nuovo baricentro e del vertice in cui si trova la singolaritá di f_4.

Come nel caso della cardioide, i tests su f_1 e f_2 approssimano I_{num} a meno di tol. Inoltre, la funzione f_2, per la sua natura oscillante in tutto il dominio, necessita un maggior numero di iterazioni rispetto a tutte le altre.

Per quanto riguarda le singolaritá di tipo radice, f_3 e f_4 stabilizzano il proprio errore a partire rispettivamente da tol= 10^{-6} all'ordine di 10^{-7} e da 10^{-9} all'ordine di 10^{-10}.

Tutte le valutazioni dell'algoritmo adattivo sono convergenti.

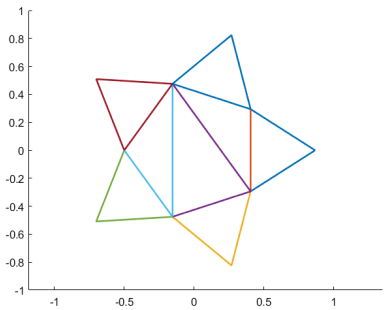
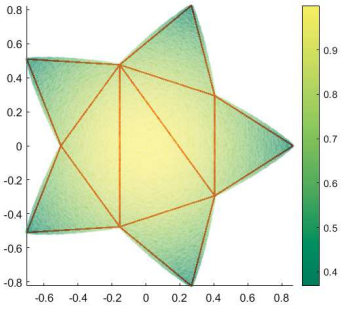
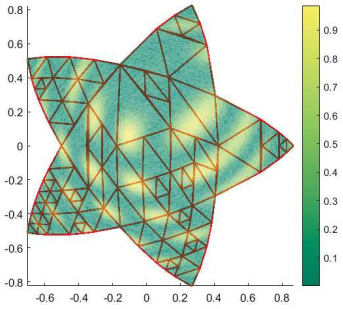


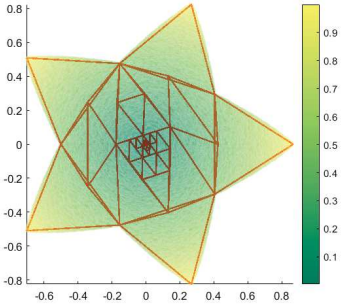
Figura 3.5: La triangolazione minima su un decagono sferico (proiezione su un piano).



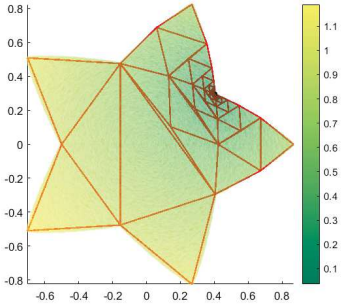
(a) La triangolazione per f_1.



(b) La triangolazione per f_2.



(c) La triangolazione per f_3.



(d) La triangolazione per f_4.

Figura 3.6: In (a), (b),(c) e (d) le triangolazioni eseguite dall'algoritmo adattivo rispettivamente per f_1, f_2, f_3, f_4 per `atol`=`rtol`=10^{-12} (vista planare: view(0,90)).

$I_{\text{num}} = 1.224085992888560e+00$, nodi: 42972						
tol	it.	triang.	I_{a}	time	AE	RE
1e-06	1	8	1.224085992888558e+00	0.05s	1.78e-15	1.45e-15
1e-09	1	8	1.224085992888558e+00	0.07s	1.78e-15	1.45e-15
1e-12	1	8	1.224085992888558e+00	0.04s	1.78e-15	1.45e-15

Tabella 3.9: Indicazione dei parametri rilevanti per i tests sulla funzione f_1.

$I_{\text{num}} = 3.169038982467544e-01$, nodi: 42972						
tol	it.	triang.	I_{a}	time	AE	RE
1e-06	2	11	3.169041607683646e-01	0.1s	2.63e-07	8.28e-07
1e-09	7	26	3.169038984741208e-01	0.20s	2.27e-10	7.17e-10
1e-12	44	137	3.169038982466765e-01	0.97s	7.78e-14	2.46e-13

Tabella 3.10: Indicazione dei parametri rilevanti per i tests sulla funzione f_2.

$I_{\text{num}} = 8.144739939248191e-01$, nodi: 42972						
tol	it.	triang.	I_{a}	time	AE	RE
1e-06	2	11	8.144745055120493e-01	0.07s	5.12e-07	6.28e-07
1e-09	8	29	8.144738644762369e-01	0.19s	1.29e-07	1.59e-07
1e-12	20	65	8.144738652096978e-01	0.36s	1.29e-07	1.58e-07

Tabella 3.11: Indicazione dei parametri rilevanti per i tests sulla funzione f_3.

$I_{\text{num}} = 1.273072680943058e+00$, nodi: 42972						
tol	it.	triang.	I_{a}	time	AE	RE
1e-06	1	8	1.273072697663152e+00	0.05s	1.67e-08	1.31e-08
1e-09	10	35	1.273072681165690e+00	0.21s	2.23e-10	1.75e-10
1e-12	26	83	1.273072680809076e+00	0.46s	1.34e-10	1.05e-10

Tabella 3.12: Indicazione dei parametri rilevanti per i tests sulla funzione f_4.

Bibliografia

[1] G. Da Fies, A. Sommariva e M. Vianello. *Algebraic cubature by linear blending of elliptical arcs*, Applied Numerical Mathematics 74 (2013), pp. 49–61.

[2] L.F. Shampine. *MATLAB Program for Quadrature in 2D*, Applied Mathematics and Computation 202 issue 1 (2008), pp. 266–274.

[3] L.F. Shampine. *Vectorized adaptive quadrature in MATLAB*, Journal of Computational and Applied Mathematics 221 issue 2 (2008), pp. 131–140.

[4] A. Sommariva e M. Vianello. *Near-algebraic Tchakaloff-like quadrature on spherical triangles*, Applied Mathematics Letters 120 (2021), p. 107282.

Appendice A

Codici

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

`https://github.com/NicoleMietto/adaptivecub_sphplg`

Demo:

```
function demo_sphplg

%This demo, given a list of 3 tolerances, a domain (example)
%and a function shows 5 graphs and a table containing the main
%datas.
%figure 1: minimal domain's triangulation.
%figure 2,3,4: domain's adaptive triangulation due to
%respectively first, second and third tolerance in the list.
%figure 5: algebric cubature formula's (not adaptive) nodes
%with 40 as degree of exactness.

example=3;
function_type=4;
deg_rule=40;

[vertices,domain_str,V_i]=define_domain(example);
[f,fs]=define_function(function_type,vertices,V_i);

vertices=vertices./(vecnorm(vertices'))';
atol_list=[10^(-6); 10^(-9); 10^(-12)];

clf;

iter_list=[];
```

```
triang=[];
Ia=[];
AE=[];
RE=[];
tempo=[];

 for i=1:3
    atol=atol_list(i);
    rtol=atol;

    tic;
    [I,Ierr,flag,Ihigh,iters,tri_vertices,tri_conn_list,
        L1_vertices]=...
    adaptive_cub_sphpgon(vertices,f,atol,rtol);
    t_2=toc;

    [XW,tri_vertices,tri_conn_list]=cub_sphpgon(deg_rule,
        vertices);

    fX=feval(f,XW(:,1),XW(:,2),XW(:,3));
    Inum=(XW(:,4))'*fX;

    iter_list=[iter_list; iters];
    triang=[triang; length(L1_vertices)];
    Ia_string=sprintf('%1.15e', I);
    Ia=[Ia; Ia_string];
    AE=[AE; abs(I-Inum)];
    RE=[RE;abs(I-Inum)/Inum];
    tempo=[tempo; t_2];

    %prints some results
    if i==1
        fprintf('\n \t ......................');
        fprintf('\n \t '); disp(fs);
        fprintf('\n \t '); disp(domain_str);
        fprintf('\n \t ..... geometry .....');
        fprintf('\n \t vertices : %8.0f',size(tri_vertices,1));
        fprintf('\n \t triangle : %8.0f',size(tri_conn_list,1))
            ;
        fprintf('\n \t ...... rule .......');
        fprintf('\n \t pts      : %8.0f',size(XW,1));
        fprintf('\n \t In: %-1.15e',Inum);
        fprintf('\n \t ********************');
    end
    fprintf('\n \t --------------------');
    fprintf('\n \t TOL      : %1.15e',atol);
    fprintf('\n \t ..... adaptive .....');
    fprintf('\n \t iters    : %8.0f',iters);
    fprintf('\n \t triangles: %8.0f',length(L1_vertices));
```

```
fprintf('\n \t ..... errors ......');
fprintf('\n \t Ia: %-1.15e',I);
fprintf('\n \t AE: %-1.15e',I-Inum);
fprintf('\n \t RE: %-1.15e \n \n',(I-Inum)/Inum);
fprintf('\n \t --------------------');

%To create the coloured background
Neval = 150000;
N = 100000;
xeval = PtsInSphPol(Neval,'S',vertices);
p = haltonset(2);
Xsq = net(p,N);
Xsq = [2*Xsq(:,1)-1,2*pi*Xsq(:,2)];
X = [sqrt(1-Xsq(:,1).^2).*cos(Xsq(:,2)), sqrt(1-Xsq(:,1)
    .^2).*sin(Xsq(:,2)), Xsq(:,1)];
F = f(xeval(:,1),xeval(:,2),xeval(:,3));
%
    ------------------------------------------------------------------


if i==1
   figure(1)
   hold on;
   for k=1:size(tri_conn_list,1)

    con_L=tri_conn_list(k,:);
    tri_L=tri_vertices(con_L',:);
    tri_L(end+1,:)=tri_L(1,:);
    plot3(tri_L(:,1),tri_L(:,2),tri_L(:,3),'LineWidth',1.5)
        ;
   end
   if function_type==3
        plot3(V_i(1),V_i(2),V_i(3),'ko','MarkerFaceColor','
            k', ...
        'MarkerSize',4);
   end
   axis equal;
   if example==1
        view(215, 0);
   else
        view(0, 90);
   end
end
hold off;

figure(i+1)

scatter3(xeval(:,1),xeval(:,2),xeval(:,3),15,F,'o','filled'
    ,'MarkerFaceAlpha',0.1,'MarkerEdgeAlpha',0.1)
```

```
    colormap summer
    colorbar
    axis equal;
    grid off;
    hold on;

    for k=1:length(L1_vertices)
      tri_L=L1_vertices{k};
      tri_L(end+1,:)=tri_L(1,:);
      plot3(tri_L(:,1),tri_L(:,2),tri_L(:,3),'r-','LineWidth'
          ,1.5);
      hold on;
    end
    if function_type==4
      plot3(V_i(1),V_i(2),V_i(3),'ko','MarkerFaceColor','k',
          ...
      'MarkerSize',4);
    end

    if example==1
            view(215, 0);
    else
            view(0, 90);
    end
    hold off;

end

figure(5)
    plot3(vertices(:,1),vertices(:,2),vertices(:,3),'mo',...
    'MarkerFaceColor','b', ...
    'MarkerSize',2);
    hold on;
    plot3(XW(:,1),XW(:,2),XW(:,3),'mo',...
    'MarkerFaceColor','m', ...
    'MarkerSize',2);
    axis equal;
    if example==1
            view(215, 0);
    else
            view(0, 90);
    end
    hold off;

T = table(atol_list, iter_list, triang, Ia, tempo, AE, RE, '
    VariableNames', ...
    {'tol', 'iteraz.','triang.','Ia','time','AE','RE'});
display(T)
```

```
end



function [vertices,domain_str,V_i]=define_domain(example)

%
    ------------------------------------------------------------------------

% Object:
% This routine, defines the domain to analyse.
%
    ------------------------------------------------------------------------

% Input:
% example: determines the spherical polygon domain to be
% analysed. From 1 to 3.
%
    ------------------------------------------------------------------------

% Output:
% vertices: it is a matrix whose rows are the vertices of the
% spherical triangles.
% domain_str: string that stores the geometry of the region.
% V_i: f_4's singularity vertice.
%
    ------------------------------------------------------------------------



switch example

    case 1

        % Australia island (no Tasmania), vertices in degrees.
        % Column 1: Longitude. Column 2: Latitude.
        % Planar domain in polyshape form.
        australia_pshape=coastline_australia(0);
        vertices_degs=australia_pshape.Vertices;

        % Australia vertices in radians.
        longitudes=deg2rad(vertices_degs(:,1));
        latitudes=deg2rad(vertices_degs(:,2));

        % Australia vertices in cartesian coordinates.
        [XX,YY,ZZ] = sph2cart(longitudes,latitudes,1);
        vertices=[XX YY ZZ];
```

```
        V_i=[XX(33) YY(33) ZZ(33)];

        domain_str='Australia as island (170 vertices, via
            polyshape)';

    case 2
        spec_settings=32;
        th=linspace(0,2*pi,spec_settings+1);
        polygon_sides=[cos(th').*(1-cos(th')) sin(th').*(1-cos(
            th'))];
        polygon_sides=polygon_sides(1:end-1,:);
        XV=polygon_sides(:,1); XV=XV/2.1;
        YV=polygon_sides(:,2); YV=YV/2.1;
        ZV=sqrt(1-XV.^2-YV.^2);
        vertices=[XV YV ZV];
        V_i=[0 0 1];

        domain_str='Polygonal cardioid at North-Pole';

    case 3
        vertices = [[0.4045    0.2939    0.8660];
                    [0.2676    0.8236    0.5000];
                    [-0.1545    0.4755    0.8660];
                    [-0.7006    0.5090    0.5000];
                    [-0.5000    0.0000    0.8660];
                    [-0.7006   -0.5090    0.5000];
                    [-0.1545   -0.4755    0.8660];
                    [0.2676   -0.8236    0.5000];
                    [0.4045   -0.2939    0.8660];
                    [0.8660   -0.0000    0.5000]];

        domain_str='decagono';
        V_i=vertices(1,:);
end
end

function [f,fs]=define_function(function_type,vertices,V_i)

%
    ------------------------------------------------------------------------

% Object:
% This routine, defines a function "f" to approximate and a
% string "fs" for possible messages to the user.
%
    ------------------------------------------------------------------------

% Input:
% function_type: determines the function to study.
```

```
% vertices: domain's vertices
% V_i: f_4's singularity
%
    ------------------------------------------------------------------------

% Output:
% f: defines a function "f" to approximate;
% fs: string with the content of the function "f".
%
    ------------------------------------------------------------------------

% Reference paper:
% A. Sommariva and M. Vianello
% Numerical hyperinterpolation overspherical triangles
%
    ------------------------------------------------------------------------


% ... centroid computation ...
ok_vertices=find( isnan(vertices(:,1))== 0 & ...
isnan(vertices(:,2)) == 0 );
vertices=vertices(ok_vertices,:);
centroid=sum(vertices,1); centroid=centroid/norm(centroid);
x0=centroid(1); y0=centroid(2); z0=centroid(3);

switch function_type

    case 1
        % ... function ...
        g=@(x,y,z) (x-x0).^2 + (y-y0).^2 + (z-z0).^2;
        f=@(x,y,z) exp( -g(x,y,z));
        fs=['exp(-g(x,y,z))' ...
            ' g=@(x,y,z) (x-x0).^2+(y-y0).^2+(z-z0).^2'];

        % ... output string ...
        x0str=num2str(x0,'%1.3e');
        y0str=num2str(y0,'%1.3e');
        z0str=num2str(z0,'%1.3e');
        fs=['exp(-g(x,y,z))' ...
            ' g(x,y,z) = (x-x0).^2+(y-y0).^2+(z-z0).^2'];
        fs=strcat(fs,'  centroid=(',x0str,',',y0str,',',z0str,'
           )');

    case 2

        % ... function ...
        g=@(x,y,z) (x-x0).^2 + (y-y0).^2 + (z-z0).^2;
        f=@(x,y,z) exp( -g(x,y,z)).*(sin(10.*y+20*z)).^2.*cos
           (10.*x+20*z).^2;
```

```
            fs=['exp(-g(x,y,z)).*(sin(10.*x)).^2.*cos(10.*x).^2,'
                ...
                ' g=@(x,y,z) (x-x0).^2+(y-y0).^2+(z-z0).^2'];

            % ... output string ...
            x0str=num2str(x0,'%1.3e');
            y0str=num2str(y0,'%1.3e');
            z0str=num2str(z0,'%1.3e');
            fs=['exp(-g(x,y,z)).*(sin(10.*x)).^2.*cos(10.*x).^2'
                ...
                ' g(x,y,z) = (x-x0).^2+(y-y0).^2+(z-z0).^2'];
            fs=strcat(fs,'  centroid=(',x0str,',',y0str,',',z0str,'
                )');

        case 3 % centroid distance like

            % ... function ...
            f=@(x,y,z) ((x-x0).^2 + (y-y0).^2 + (z-z0).^2).^(1/2);

            % ... output string ...
            x0str=num2str(x0,'%1.3e');
            y0str=num2str(y0,'%1.3e');
            z0str=num2str(z0,'%1.3e');
            fs='((x-x0).^2 + (y-y0).^2 + (z-z0).^2).^(1/2), ';
            fs=strcat(fs,'  centroid=(',x0str,',',y0str,',',z0str,'
                )');

        case 4

            % ... function ...
            f=@(x,y,z) ((x-V_i(1)).^2 + (y-V_i(2)).^2 + (z-V_i(3))
                .^2).^(1/4);

            % ... output string ...
            x0str=num2str(V_i(1),'%1.3e');
            y0str=num2str(V_i(2),'%1.3e');
            z0str=num2str(V_i(3),'%1.3e');
            fs='((x-V_i(1)).^2 + (y-V_i(2)).^2 + (z-V_i(3)).^2)
                .^(1/4), ';
            fs=strcat(fs,'  V_i=(',x0str,',',y0str,',',z0str,')');


    end  % end: "define_function"

    end
```

Algoritmo adattivo:

```
function [I,Ierr,flag,Ihigh,iters,tri_vertices,tri_conn_list,
    L1_vertices]=...
     adaptive_cub_sphpgon(vertices,f,atol,rtol)

%
    ---------------------------------------------------------------------------

% Input:
% vertices : it is a M x 3 matrix, where the k-th row
%            represents the cartesian coordinates of the k-th
%            vertex of the spherical polygon (counterclockwise
%            order);
% f        : function to integrate over the spherical triangle
%            defined by vertices;
% atol     : absolute cubature error tolerance.
% rtol     : relative cubature error tolerance.
%
    ---------------------------------------------------------------------------

% Output:
% I      : approximation of integral of "f" over the spherical
%          triangle defined by vertices
%
    ---------------------------------------------------------------------------

% Subroutines:
% 1. area_spherical_triangle;
% 2. center_spherical_triangle;
% 3. generate_sphtri_sons.
%
    ---------------------------------------------------------------------------

% Information:
% Authors: A. Sommariva and M. Vianello.
% Date: June 7, 2021.
% Modified: November 12, 2024.
%
    ---------------------------------------------------------------------------

%% Copyright (C) 2021-
%% Alvise Sommariva, Marco Vianello.
%%
%% This program is free software; you can redistribute it and/
%% or modify it under the terms of the GNU General Public
%% License as published by the Free Software Foundation; either
%% version 2 of the License, or (at your option) any later
```

```
%% version.
%%
%% This program is distributed in the hope that it will be
%% useful, but WITHOUT ANY WARRANTY; without even the implied
%% warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
%% PURPOSE.  See the GNU General Public License for more
%% details.
%%
%% You should have received a copy of the GNU General Public
%% License along with this program; if not, write to the Free
%% Software Foundation, Inc., 59 Temple Place, Suite 330,
%% Boston, MA  02111-1307 USA
%%
%% Authors:
%% Alvise Sommariva, Marco Vianello.
%
    ------------------------------------------------------------------------


fprintf('\n adaptive_cub_sphpgon');



% ......... Troubleshooting and settings .........

if nargin < 3, atol=10^(-6); end
if nargin < 4, rtol=10^(-6); end
tri_meth=2; % Triangulation method.

% max number of triangles in which the domain is partitioned
max_triangles=1000;

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

% ..... Procedure start up, data stored in structures .....

%
    ........................................................................


% Main strategy:
% First we triangulate the spherical polygon, providing the
% vertices of the triangulation and its connectivity list
% (description of each spherical triangle via its vertices).
%
% Below, we make 2 lists of sph. triangle data:
% 1. LIST 1: all the triangles provide contribution to the
%            integration result and for each triangle an error
%            estimate is available;
```

```
% 2. LIST 2: all the triangles are "sons" triangles of some
%            triangle in LIST 1; integrals are computed in each
%            of them but no error estimate is available.
%
% We store the relevant data of each spherical triangle (LIST 1
% set).
%
% * L1_vertices : cell, whose k-th element are the vertices of
%                 the k-th sph. triangle stored as a 3 x 3
%                 matrix, whose j-th row are the cartesian
%                 coordinates  of the j-th vertex of the k-th
%                 sph. triangle (ordered counterclockwise);
% * L1_areas    : vector, whose k-component represent the area
%                 of the k-th sph. triangle;
% * L1_integrals: vector, whose k-component represent the
%                 approximation of the integral of "f" on the
%                 k-th sph. triangle;
% * L1_errors   : vector, whose k-component represent the
%                 approximation of the error of integral on the
%                 k-th sph. triangle.
%
% If the error estimate is not satisfied, the code find the
% worst triangle, subdvides it, and updates the lists.
%
    ..........................................................................


% fprintf('\n \t Adaptive code: triangulation');
% ......... Troubleshooting and settings .........

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

% ..... Procedure start up, data stored in structures.....

%
    ..........................................................................

% Main strategy:
% First we triangulate the spherical polygon, providing the
% vertices of the triangulation and its connectivity list
% (description of each spherical triangle via its vertices).
% Next we determine a rule for each spherical triangle.
%
    ..........................................................................


%[tri_vertices,tri_conn_list]=triangulate_sphpgon(vertices);
```

```
subdom_index=find(isnan(vertices(:,1)) == 1);

% .... Regularize if some vertex coordinates are NaNs ....

%
    ......................................................................

% NaN components are usually introduced in vertices components
% to separate disconnected domains.
% We take this into account separating the analysis of the k-th
% subdomain taking data from vertices index "first_index(k)"
% to "last_index(k)".
%
    ......................................................................

first_index=1;
if length(subdom_index) == 0
    first_index=1;
    last_index=size(vertices,1);
else
    if subdom_index(end) == size(vertices,1)
        subdom_index=subdom_index(1:end-1);
    end

    first_index=[1 subdom_index+1];
    last_index=[subdom_index-1 size(vertices,1)];
end



% Computation of cubature rule over spherical polygon.
XW=[];
tri_vertices=[]; tri_conn_list=[];
for k=1:length(first_index)

    vertices_sub=vertices(first_index:last_index,:);
    [tri_verticesL,tri_conn_listL]=triangulate_sphpgon_tg(
        vertices_sub);
    tri_vertices=[tri_vertices; tri_verticesL];
    tri_conn_list=[tri_conn_list; tri_conn_listL];
end


L1_vertices={}; L1_integrals=[]; L1_errors=[]; L2_data={};

% fprintf('\n \t Adaptive code: start up');
for k=1:size(tri_conn_list,1)
    vertices=tri_vertices((tri_conn_list(k,:))',:);
```

```
    [L1_vertices,L1_integrals,L1_errors,L2_data,Itemp_low,
        Itemp_high]=...
        start_up(vertices,f,L1_vertices,L1_integrals,L1_errors,
            L2_data);
end

I=sum(L1_integrals);

I0=I;


% .......... successive refinement ..........

iters=1;


% fprintf('\n \t Adaptive code: refining');
while  sum(L1_errors) > max([atol rtol*I])

    % fprintf('\n \t %6.0f',iters);

    N=length(L1_integrals);

    % too many triangles: exit with errors
    if N > max_triangles
        I=sum(L1_integrals); Ierr=sum(L1_errors); flag=1;
        if nargout > 3, Ihigh=Ihigh_computation(L2_data); end
        return;
    end

    [Ierr_max,kmax]=max(L1_errors);

    % Erase "kmax" sph. triangle from LIST 1
    k_ok=setdiff(1:N,kmax);

    if length(k_ok) > 0
        L1_vertices={L1_vertices{k_ok}}; L1_integrals=
            L1_integrals(k_ok);
        L1_errors=L1_errors(k_ok);
    else
        L1_vertices={}; L1_integrals=[]; L1_errors=[];
    end

    % Move sub sph. triangles relative to "k-max" from LIST 2
    % to LIST 1.

    L2_data_kmax=L2_data{kmax}; % cell with 4 data structs

    if length(k_ok) > 0
```

```
        % erase cell relative to sub triangles of kmax
        L2_data={L2_data{k_ok}};
    else
        L2_data={};
    end
    % triangle, from LIST 2

    %
        ....................................................................

    % Generate sons of each son of L2_data_kmax and compute
    % approximate integration errors for each son of
    % L2_data_kmax.
    % 1. Each son of L2_data_kmax is moved to LIST 1.
    % 2. The son of each son of L2_data_kmax is moved to
    %    LIST 2.
    %
        ....................................................................


    for j=1:4
        L2_data_temp=L2_data_kmax{j}; % struct data
        L2_data_temp_sons=generate_sphtri_sons(L2_data_temp.
            vertices,f);

        % "L2_data_kmax_j_sons" is a cell with 4 structs data.
        for jj=1:4
            L2_data_temp_sons_integral(jj)=L2_data_temp_sons{jj
                }.integral;
        end
        Itemp_low=L2_data_temp.integral;
        Itemp_high=sum(L2_data_temp_sons_integral);
        Itemp_err=abs(Itemp_high-Itemp_low);

        % update LIST 1 with j-th sub triangle "generated" by
        % kmax.
        L1_vertices{end+1}=L2_data_temp.vertices;
        L1_integrals(end+1)=L2_data_temp.integral;
        L1_errors(end+1)=Itemp_err;

        % update LIST 2 with sons of j-th sub triangle
        % "generated" by kmax.
        L2_data{end+1}=L2_data_temp_sons;
    end

    iters=iters+1;
    I=sum(L1_integrals);

end
```

```
% ............ providing results ............

I=sum(L1_integrals); Ierr=sum(L1_errors); flag=0;
if nargout > 3, Ihigh=Ihigh_computation(L2_data); end

fprintf('\n \t I0: %1.15e',I0);
fprintf('\n \t I1: %1.15e',I);
```

```
function [L1_vertices,L1_integrals,L1_errors,L2_data,Itemp_low
    ,...
    Itemp_high]=start_up(vertices,f,L1_vertices,L1_integrals,
        L1_errors,...
    L2_data)

%
    ------------------------------------------------------------------------

% Object:
% Given a sph.triangle with vertices "vertices", a cell with
% previous vertices of previous triangles, a vector
% "L1_integrals" with integrals of "f" in the previous
% triangles and a vector "L1_errors" of the error
% estimates of the integrals of "f" in the previous triangles,
% it updates these cells and vectors with vertices of the
% new triangle, the integral of "f" in the new triangle, and
% its error estimate.
% Data about the sons of the new triangle is put in the cell
% "L2_data".
%
    ------------------------------------------------------------------------
```

```
% vertices
L1_vertices{end+1}=vertices;

% integrals
P1=(vertices(1,:))'; P2=(vertices(2,:))'; P3=(vertices(3,:))';

xyzw= cub_sphtri(4,P1,P2,P3);
Itemp_low=(xyzw(:,4))'*feval(f,xyzw(:,1),xyzw(:,2),xyzw(:,3));

% approximation of integral on triangle (few evals)
L1_integrals=[L1_integrals; Itemp_low];

%
    .....................................................................

% Note:
% We subdivide each sph. triangle in the list LIST 1 in 4
% triangles, that we call "sons".
%
% Each initial planar triangle defining the generical
% sph. triangle, has 3 midpoints. In view of the midpoints, we
% get 4 triangles.
%
% Of each son triangle we make a structure:
%
% * L2_data.vertices : matrix 3 x 3, whose l-th row are the
%                      cartesian coordinates of the l-th vertex
%                      of the j-th sub sph. triangle;
% * L2_data.area     : scalar representing the area of the j-th
%                      son sph. triangle;
% * L2_data.integral : scalar representing the integral of "f"
%                      on the j-th son sph. triangle.
%
    .....................................................................


L2_data_temp=generate_sphtri_sons(vertices,f);
L2_data{end+1}=L2_data_temp;

% ... Evaluate absolute error ...
for j=1:4, L2_data_integrals(j)=L2_data_temp{j}.integral; end

% better approximation of integral on triangle (more evals)
Itemp_high=sum(L2_data_integrals);

L1_errors=[L1_errors; abs(Itemp_high-Itemp_low)];
```

```
function Ihigh=Ihigh_computation(L2_data)

%
    ------------------------------------------------------------------------

% Object:
% Approximation of the integral in view of approximations in
% LIST 2.
%
    ------------------------------------------------------------------------

% Input:
% L2_data: struct defining LIST 2 sph. triangles vertices,
%          areas and integral approximations.
%
    ------------------------------------------------------------------------

% Output:
% Ihigh:   approximation of integral of "f" over the initial
%          spherical triangle.
%
    ------------------------------------------------------------------------


M=length(L2_data);
Ihigh=0;
for j=1:M
    L2_data_temp=L2_data{j};
    for jj=1:4
        L2_data_temp_sons_integral(jj)=L2_data_temp{jj}.
            integral;
    end
    Ihigh=Ihigh+sum(L2_data_temp_sons_integral);
end
```

```
function L2_data=generate_sphtri_sons(vertices,f)

%
    ------------------------------------------------------------------------

% Input:
% vertices : it is a 3 x 3 matrix, where the k-th row
%            represents the cartesian coordinates of the
%            k-th vertex (counterclockwise);
% f        : function to integrate over the spherical triangle
%            defined by vertices;
%
    ------------------------------------------------------------------------

% Output:
% L2_data :
%
    ------------------------------------------------------------------------


OA=(vertices(1,:)); OB=(vertices(2,:)); OC=(vertices(3,:));
R=norm(OA);

% ......... compute midpoints .........

OAB_mid=(OA+OB)/2; OAB_mid=R*OAB_mid/norm(OAB_mid);
OAC_mid=(OA+OC)/2; OAC_mid=R*OAC_mid/norm(OAC_mid);
OBC_mid=(OB+OC)/2; OBC_mid=R*OBC_mid/norm(OBC_mid);

% ........ triangle data ........

vertices=[OA; OAB_mid; OAC_mid];

L2_data{1}=make_L2_data(vertices,f);
L2_data{2}=make_L2_data([OAB_mid; OB; OBC_mid],f);
L2_data{3}=make_L2_data([OBC_mid; OC; OAC_mid],f);
L2_data{4}=make_L2_data([OAB_mid; OBC_mid; OAC_mid],f);




function L2_dataL=make_L2_data(vertices,f)
```

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

% Input:
% vertices : it is a 3 x 3 matrix, where the k-th row
%            represents the cartesian coordinates of the
%            k-th vertex (counterclockwise);
% f        : function to integrate over the spherical triangle
%            defined by vertices.
%
    ------------------------------------------------------------------------

% Output:
% L2_dataL : determine from the spherical triangle defined by
%            vertices, a struct data, with form:
%            * L2_dataL.vertices,
%            * L2_dataL.area,
%            * L2_dataL.integral.
%
    ------------------------------------------------------------------------


% vertices
L2_dataL.vertices=vertices;
P1=(vertices(1,:))'; P2=(vertices(2,:))'; P3=(vertices(3,:))';
% xyzw = sphtriquad(4,5,P1,P2,P3,0);
xyzw= cub_sphtri(4,P1,P2,P3);
L2_dataL.integral=(xyzw(:,4))'*feval(f,xyzw(:,1),xyzw(:,2),xyzw
    (:,3));









function [tri_vertices,tri_conn_list]=triangulate_sphpgon_tg(
    vertices)

%
    ------------------------------------------------------------------------

% Object:
% In this routine we determine a triangulation of the spherical
% polygon with M vertices described in cartesian coordinates
% the M x 3 matrix "vertices".
```

```
% The points of the triangulation are stored in "tri_vertices",
% while the K x 3 connectivity matrix is described in
% "tri_conn_list".
% In particular the k-th row of "tri_conn_list" consists of the
% indices of vertices of the k-th spherical triangle, w.r.t.
% "tri_vertices".
%
    ------------------------------------------------------------------------



% .......... compute barycenter vertices ..........

R=norm(vertices(1,:)); vertices=vertices/R;
CC=mean(vertices); CC=CC/norm(CC);

% ........ rotation matrix to the north pole ........

[az,el,r] = cart2sph(CC(1),CC(2),CC(3));
phi=az; theta=pi/2-el;
cp=cos(phi); sp=sin(phi); ct=cos(theta); st=sin(theta);
R1=[ct 0 -st; 0 1 0; st 0 ct]; R2=[cp sp 0; -sp cp 0; 0 0 1];
rotmat=R1*R2; inv_rotmat=rotmat';

% ........... rotate vertices ...........

vertices_NP=(rotmat*vertices')';

% ... map radially vertices to plane tangent to north pole ...

XX_NP=vertices_NP(:,1); YY_NP=vertices_NP(:,2); ZZ_NP=
    vertices_NP(:,3);
rat=1./(1+ZZ_NP);

XX_NPm=rat.*XX_NP; YY_NPm=rat.*YY_NP; ZZ_NPm=ones(size(XX_NPm))
    ;

% ... polyshape of projected polygon to North Pole ...

PG = polyshape(XX_NPm,YY_NPm);
PG = simplify(PG);

% ..... triangulation of projected polygon to North Pole .....

tri = triangulation(PG);

tri_vertices_NPm=PG.Vertices;

phi = @(x,y) [2*x./(1+x.^2+y.^2), 2*y./(1+x.^2+y.^2), (1-x.^2-y
```

```
    .^2)./(1+x.^2+y.^2)];
tri_vertices_NPm=phi(tri_vertices_NPm(:,1),tri_vertices_NPm
    (:,2));

% ........... output data ...........

%tri_vertices_NP=tri_vertices_NPm./rads;
tri_vertices=R*(inv_rotmat*tri_vertices_NPm')';
tri_conn_list=tri.ConnectivityList;
```