UNIVERSITÀ DEGLI STUDI DI PADOVA

FACOLTÀ DI SCIENZE MATEMATICHE FISICHE E NATURALI
CORSO DI LAUREA TRIENNALE IN MATEMATICA

TESI DI LAUREA

CONFRONTO DI METODI PER L'INTEGRAZIONE NUMERICA DI GAUSSIANE BIVARIATE SU POLIGONI

Relatore: Ch.mo Prof. Marco Vianello

Correlatore: Ch.mo Prof. Alvise Sommariva

Laureanda:
Sara Ballan
591905

Anno accademico: 2010-2011

Indice

Capitolo 1

Introduzione

In questo lavoro si considera il problema di integrare una gaussiana in due variabili su un poligono semplice (cfr. [15]) e limitato $\Omega \in \mathbb{R}^2$. Questo problema compare nella letteratura statistica, si veda ad esempio la Tesi di Master di S. Meyer (cfr. [5]) dell'Università di Monaco, dove nel capitolo 3 vengono utilizzati e confrontati a tal fine sei diversi metodi di cubatura numerica su poligoni.

La funzione gaussiana bivariata ha la seguente forma (cfr. [7],[12]):

$$f(x,y) = \frac{1}{2\pi\sigma_x\sigma_y\sqrt{1-\rho^2}} \exp\left\{-\frac{1}{2(1-\rho^2)}\left[\frac{(x-\mu_x)^2}{\sigma_x^2} + \frac{(y-\mu_y)^2}{\sigma_y^2} - \frac{2\rho(x-\mu_x)(y-\mu_y)}{\sigma_x\sigma_y}\right]\right\} \quad (1.1)$$

e i suoi parametri sono

(μ_x, μ_y) vettore delle medie marginali,

$\left(\sigma_x^2, \sigma_y^2\right)$ vettore delle varianze marginali,

ρ il coefficiente di correlazione.

In ambito probabilistico essa rappresenta la densità di un vettore aleatorio normale (o gaussiano) $X \sim N(\mu, \Sigma)$, con $\mu = \begin{pmatrix} \mu_x \\ \mu_y \end{pmatrix}$, la media, $\Sigma = \begin{pmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{pmatrix}$, la matrice di covarianza, e in quanto tale viene impiegata in un enorme numero di applicazioni pratiche. Nasce da qui l'interesse nel trovare un algoritmo efficiente per approssimare il suo integrale su un dato poligono nel minor tempo possibile. A tale scopo si è deciso di utilizzare uno dei più popolari linguaggi di programmazione, Matlab (cfr. [4]). La letteratura riguardante in generale il problema dell'integrazione numerica su poligoni non è molto vasta, nonostante il fatto che questi siano al centro della geometria computazionale.

Un passo fondamentale nella creazione del codice è costituito dal teorema di Green (cfr. [1]), che in una delle sue formulazioni di base può essere scritto come

$$\int\int_{\Omega} f(x,y)dxdy = \oint_{\partial\Omega} F(x,y)dy, \quad F(x,y) = \int f(x,y)dx, \tag{1.2}$$

dove f è di classe C^1 in $\Omega \subset \mathbb{R}^2$, un dominio chiuso che ha come bordo una curva di Jordan (in questo contesto f è la gaussiana bivariata in (1.1), Ω è il poligono con i vertici dei lati descritti in senso antiorario). Questo risultato fornisce un importante strumento per la cubatura numerica, dal momento che trasforma un problema 2-dimensionale in 1-dimensionale. Per conoscere la x-primitiva di f, richiesta nell'utilizzo pratico del teorema, si è deciso di ricorrere ad un integratore simbolico [16], ottenendo così:

$$F(x,y) = \frac{1}{2}\frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(y-\mu_y)^2}{2\sigma_y^2}\right) \operatorname{erf}\left(\frac{\sigma_x\rho(\mu_y - y) + \sigma_y(x-\mu_x)}{\sqrt{2(1-\rho^2)}\sigma_x\sigma y}\right). \tag{1.3}$$

Servendosi infine di 'quadgk' (cfr. [4]), una particolare funzione del Matlab per la quadratura numerica, è stato implementato il programma Matlab 'MySoftware' (il codice è presente nel capitolo 5). Questo richiede all'utente di inserire in input solamente i parametri della gaussiana, i vertici del poligono Ω e la tolleranza con la quale si vuole venga calcolato l'integrale; il valore approssimato di quest'ultimo viene invece restituito in output.

Questo approccio permette di affrontare il problema dell'integrazione sul poligono, conoscendo solo i suoi vertici ordinati in senso antiorario, senza dovergli applicare un metodo di triangolarizzazione. Si ha quindi un vantaggio notevole, considerando che non risulta banale interfacciare triangolatori efficienti (che lavorino anche su poligoni non convessi), tipicamente implementati in C/C++, con un buon integratore Matlab su triangoli.

In seconda istanza si è affrontato il medesimo problema, utilizzando altri tre metodi di cubatura su poligoni noti in letteratura, per poi valutare l'efficienza di ciascuno di questi e confrontare i risultati numerici ottenuti.

Il primo è la funzione 'polygauss' [11], che consiste in una formula di cubatura 'gaussiana', anch'essa costruita a partire dal teorema di Green (cfr. [1]).

Il secondo è un integratore automatico del Matlab, la funzione 'dblquad' (cfr. [4]), con la quale si riesce a trattare domini non standard (come ad esempio i poligoni), valutando il prodotto dell'integranda assegnata per la funzione caratteristica del dominio, su un rettangolo che lo contiene.

Infine il terzo algoritmo, 'twoD' [9], è un integratore bidimensionale adattativo in parte simile a 'dblquad', con delle particolarità che lo rendono spesso più efficiente del precedente.

Nel secondo capitolo sono discussi 'MySoftware' (cap. 5) e 'polygauss' [11]. Per quanto riguarda il primo, ne vengono illustrate le caratteristiche

principali, i procedimenti e i passaggi teorici con cui si è arrivati alla sua stesura. Per il secondo si richiama invece la dimostrazione delle proprietà teoriche della formula di cubatura.

I rimanenti due algoritmi sono descritti e confrontati tra loro nel terzo capitolo.

Infine nel quarto capitolo si analizzano in vari esempi i risultati numerici ottenuti approssimando l'integrale della gaussiana bivariata. Si vede che 'MySoftware' (cap. 5) è l'algoritmo più efficiente: a differenza degli altri tre riesce sempre a restituire in output il risultato con un errore assoluto minore della tolleranza richiesta, con tempi di esecuzione che variano dai centesimi, ad al più i decimi di secondo.

Capitolo 2

MySoftware e polygauss

2.1 Il teorema di Green

Questi primi due codici hanno come punto di partenza comune il teorema di Green (cfr. [1]), il quale viene spesso richiamato esplicitamente nel contesto della cubatura numerica per la sua potenzialità nel trasformare un problema 2-dimensionale in uno 1-dimensionale: esso esprime un integrale doppio su una regione piana R come integrale di linea lungo una curva chiusa, C, che costituisce la frontiera di R.

Teorema 1 (Teorema di Green per regioni piane limitate da curve di Jordan generalmente regolari)**.** *Siano P e Q campi scalari derivabili con continuità su un insieme aperto S del piano xy. Sia C una curva di Jordan, generalmente regolare, e sia R l'unione di C col suo interno. Supponiamo che R sia un sottoinsieme di S. Allora vale l'identità*

$$\int\int_R \left(\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y}\right) dxdy = \oint_C (Pdx + Qdy) \tag{2.1}$$

dove l'integrale di linea è fatto lungo la curva C, in senso antiorario.

Si può notare che sono richieste due tipi di ipotesi per la validità dell'uguaglianza (2.1): la condizione di derivabilità con continuità delle funzioni P, Q sull'aperto S, che servono per assicurare l'esistenza degli integrali, e le condizioni di natura geometrica che sono imposte alla regione R e alla curva C, frontiera di R.

C deve essere una curva *chiusa*, *semplice*, *rettificabile*. Supponendo che essa sia descritta da una funzione α, definita su un intervallo $[a, b]$, a valori vettoriali, continua, se $\alpha(a) = \alpha(b)$, allora la curva C si dice *chiusa*. Poi una curva chiusa tale che $\alpha(t_1) \neq \alpha(t_2)$, per ogni coppia di valori $t_1 \neq t_2$ nell'intervallo semiaperto $(a, b]$, si chiama curva chiusa *semplice*. Ciò significa che, eccetto per gli estremi dell'intervallo $[a, b]$, valori distinti di t individuano punti distinti sulla curva. Infine il termine 'rettificabile' significa che C ha lunghezza finita.

Le curve chiuse semplici che stanno in un piano sono usualmente chiamate *curve di Jordan*. Ogni curva di Jordan, C, divide il piano in due aperti connessi disgiunti che hanno la curva C come loro frontiera comune. Una di queste regioni è limitata ed è chiamata la *regione interna* a C; l'altra è illimitata ed è chiamata la *regione esterna* a C.

Il teorema di Green è valido ogniqualvolta C è una curva di Jordan rettificabile e la regione R è l'unione di C e del suo interno.

Corollario 1. *Dato un dominio chiuso $\Omega \subset \boldsymbol{R}^2$ la cui frontiera è una curva di Jordan e una funzione f di classe C^1 in Ω si ha*

$$\int\int_{\Omega} f(x,y)dxdy = \oint_{\partial\Omega} F(x,y)dy \tag{2.2}$$

dove $F(x,y) = \int f(x,y)dx$.

La formula d'integrazione (2.2) è quella utilizzata in questo contesto, dove si assume in particolare che Ω sia un poligono limitato e semplice (cfr. [15]) con bordo $\partial\Omega$ formato da una sequenza di vertici ordinati in senso antiorario. Si capisce che tale formula, nel suo utilizzo pratico, richiede la conoscenza di $F(x,y)$, una fissata x-primitiva di f. I codici descritti affrontano tale ostacolo in maniere diverse: nel primo viene utilizzato un integratore simbolico, il 'Wolfram Integrator' [16], nel secondo si ricorre ad una formula di quadratura di Gauss-Legendre (cfr. [3]). Differenti sono poi anche i due approcci per integrare $F(x,y)$ su $\partial\Omega$, il bordo del poligono: nel primo caso si adopera 'quadgk' (cfr. [4]) una funzione predefinita del Matlab, mentre nel secondo viene applicata nuovamente una formula di quadratura di Gauss-Legendre.

2.2 MySoftware

2.2.1 *Il metodo*

Tale codice richiede in input (si veda cap. 5):

- i parametri della gaussiana ossia σ_x^2, σ_y^2, μ_x, μ_y, ρ;
- A, la matrice $n \times 2$ contenente gli n vertici del poligono numerati in senso antiorario (l'i-esima riga corrisponde ad ascissa ed ordinata dell'i-esimo vertice);
- *tol*, la tolleranza per l'errore assoluto con il quale si vuole che venga calcolato l'integrale.

Mentre in output ritorna l'integrale di $f(x,y)$, la funzione gaussiana in due variabili in (1.1).
Calcolando la x-primitiva di f, $F(x,y) = \int f(x,y)dx$, con l'integratore simbolico (cfr. [16]), si ottiene:

$$F(x,y) = \frac{1}{2}\frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(y-\mu_y)^2}{2\sigma_y^2}\right) \operatorname{erf}\left(\frac{\sigma_x\rho(\mu_y - y) + \sigma_y(x-\mu_x)}{\sqrt{2(1-\rho^2)}\sigma_x\sigma y}\right). \tag{2.3}$$

La primitiva di f è perciò il prodotto della costante $\frac{1}{2}$, per una gaussiana in una variabile di parametri (μ_y,σ_y^2) ed una funzione erf (cfr. [13]). Quest'ultima è la cosiddetta 'funzione errore', olomorfa intera e definita da

$$\operatorname{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z \exp\left(-t^2\right) dt.$$

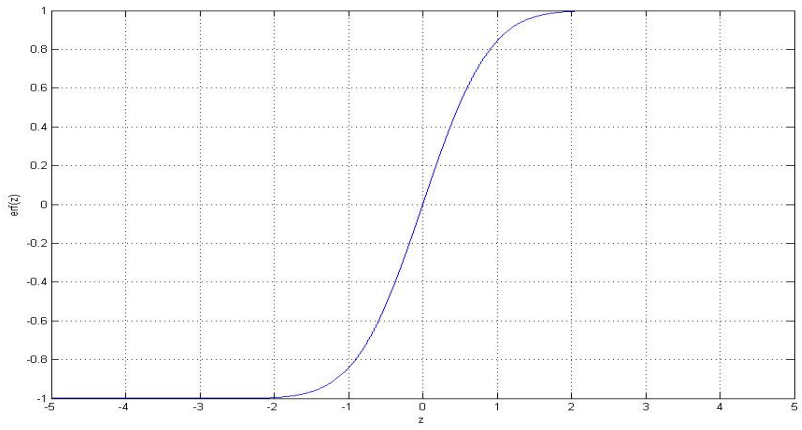


Figura 2.1: Funzione erf.

Si nota che variando a piacere i parametri della f, l'unica cosa che cambia è l'argomento della funzione erf. Prendendo ad esempio $\sigma_x^2 = \sigma_y^2$, $\mu_x \neq \mu_y$, $\rho = 0$ si ottiene:

$$F(x,y) = \frac{1}{2}\frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(y-\mu_y)^2}{2\sigma_y^2}\right) \operatorname{erf}\left(\frac{x-\mu x}{\sqrt{2}\sigma_x}\right).$$

Oppure con $\sigma_x^2 = \sigma_y^2$, $\mu_x = \mu_y$, $\rho \neq 0$ si ha:

$$F(x,y) = \frac{1}{2}\frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(y-\mu_y)^2}{2\sigma_y^2}\right) \operatorname{erf}\left(\frac{(\rho-1)\mu_x - y\rho + x}{\sqrt{2(1-\rho^2)}\sigma_x}\right).$$

Questo porta a concludere che la x-primitiva di f, a meno di una costante moltiplicativa e della funzione erf (in modulo minore o uguale ad 1) è una gaussiana nella sola variabile y, con parametri μ_y (la media marginale in y di f) e σ_y^2 (la varianza marginale in y di f). Come è noto una funzione di questo tipo tende a 0 per $y \to \pm\infty$ ed ha integrale pari ad 1 su $\mathbb{R}$. La figura 2.2 illustra in particolare quali sono le percentuali cumulate sottostanti una tale curva, in corrispondenza ai multipli della varianza (cfr. [14]):

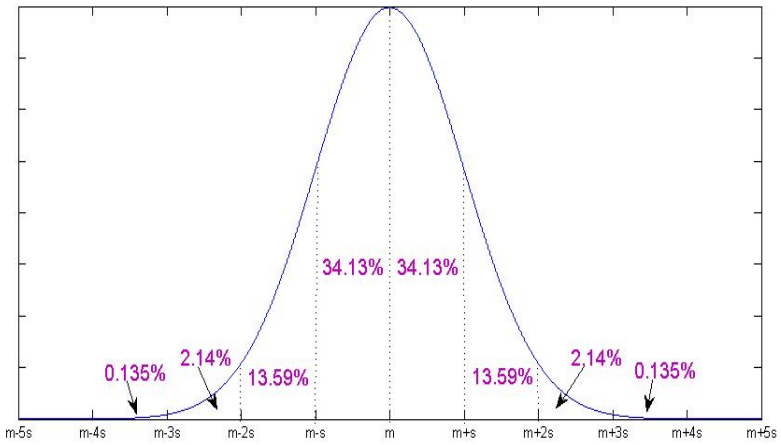


Figura 2.2: Gaussiana 1-dimensionale di media m e varianza s.

Osservandola viene l'idea di calcolare approssimativamente l'integrale della $F(x,y)$ in (2.3) su $\partial\Omega$ sfruttando la parità ed il decadimento del fattore gaussiano in essa contenuto. In base alla tolleranza (*tol*) con la quale si vuole venga restituito il risultato, si considera la x-primitiva solamente dove questa, in modulo, è superiore ad un certo valore (chiamato $\epsilon > 0$), il quale dipenderà allora da *tol*, e la si trascura nel rimanente intervallo.
Come mostra la figura 2.3 imponendo

$$\begin{aligned} |F(x,y)| &= \frac{1}{2}\frac{1}{\sqrt{2\pi\sigma_y^2}}\exp\left(-\frac{(y-\mu_y)^2}{2\sigma_y^2}\right)\underbrace{\left|\operatorname{erf}\left(\frac{\sigma_x\rho(\mu_y-y)+\sigma_y(x-\mu_x)}{\sqrt{2(1-\rho^2)}\sigma_x\sigma y}\right)\right|}_{\leq 1} \\ &\leq \frac{1}{2}\frac{1}{\sqrt{2\pi\sigma_y^2}}\exp\left(-\frac{(y-\mu_y)^2}{2\sigma_y^2}\right) \leq \epsilon \end{aligned} \tag{2.4}$$

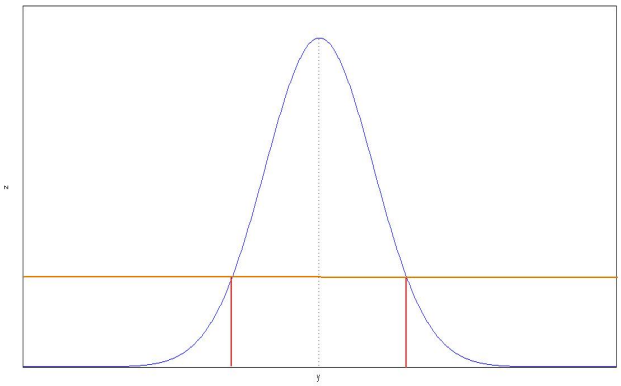


Figura 2.3: Gaussiana in una variabile.

e invertendo tale disuguaglianza si ottiene un vincolo del tipo $|y - \mu_y| \geq h(\sigma_y^2)$, dove h è un'opportuna funzione, che verrà specificata più avanti in (2.14), di variabile σ_y^2 e dipendente dal parametro *tol*. In altre parole il fattore gaussiano della x-primitiva diventa piccolo in modulo, cioè inferiore od uguale ad ϵ, quando la y sta al di fuori della striscia $\{y \in \mathbb{R} : |F(x,y)| > \epsilon\}$ centrata nella media μ_y, di semiampiezza $h(\sigma_y^2)$. Inizialmente si è così pensato di fare la seguente approssimazione:

$$\oint_{\partial\Omega} F(x,y)dy \approx \int_{\partial\Omega \cap \{y \in \mathbb{R}: |F(x,y)| > \epsilon\}} F(x,y)dy \tag{2.5}$$

si integra quindi $F(x,y)$ in (2.3) non sull'intero bordo del poligono, ma solamente sulla porzione di quest'ultimo dove F è 'sufficientemente grande', nello specifico $|F(x,y)| > \epsilon$. Da qui nasce la necessità di determinare l'ampiezza $2h(\sigma_y^2)$ della striscia $\{y \in \mathbb{R} : |F(x,y)| > \epsilon\}$ e quali siano le catene o sequenze di lati consecutivi di $\partial\Omega$ che la intersecano. La prima parte del programma svolge questi due compiti: nelle prime righe vi è il calcolo esplicito di $h(\sigma_y^2)$ (riportato e illustrato nella successiva sessione dedicata alla stima dell'errore), poi si controlla che effettivamente il poligono non sia tutto incluso nella striscia (in caso contrario questo procedimento è inutile e conviene di certo integrare su tutto $\partial\Omega$). A questo punto vengono chiamate le due funzioni 'VertexHorizontal' e 'VertexNotHorizontal' (i codici sono riportati al cap. 5), le quali ricevono in input:

μ_y , la media del fattore gaussiano in (2.3);

$h(\sigma_y^2)$, la semiampiezza della striscia;

A , la matrice a due colonne dei vertici del poligono (la quale deve essere modificata in modo tale da avere l'n+1-esima entrata uguale alla prima; si vuole cioè che l'n+1-esimo vertice coincida con il primo).

Mentre in output restituiscono:

***B*egin** , un vettore contenente gli indici di riga della matrice A che corrispondono ai vertici di inizio delle catene;

***F*inal** , un vettore contenente gli indici di riga della matrice A che corrispondono ai vertici finali delle catene.

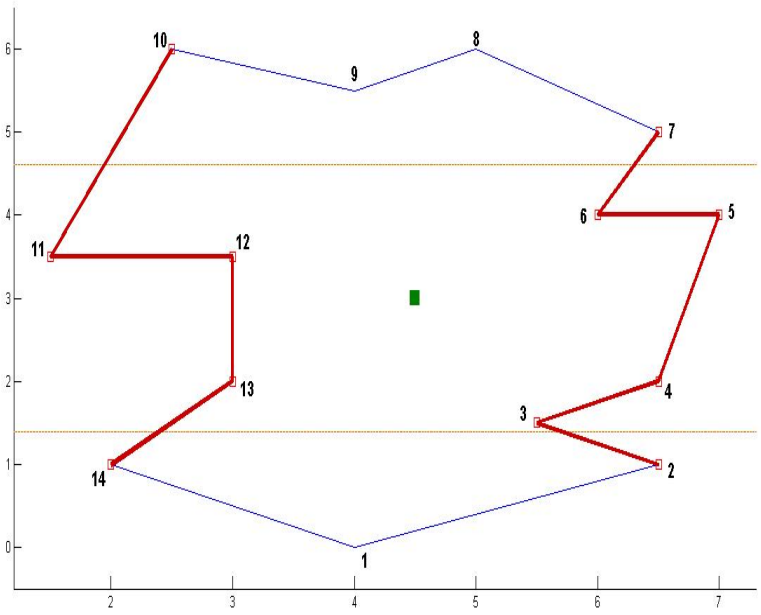
'VertexHorizontal' considera inclusi in queste sequenze di lati, anche quelli orizzontali (paralleli all'asse x), dove chiaramente l'integrale di $F(x, y)$ (2.3) in dy risulta nullo.

'VertexNotHorizontal' invece li esclude , facendo terminare la catena appena ne incontra uno: il vertice di inizio del lato orizzontale sarà quindi quello finale della catena in questione, mentre il vertice di fine dello stesso lato costituirà l'inizio della catena successiva. Il vantaggio di questa seconda scelta sarebbe quello di velocizzare poi l'integrazione di $F(x, y)$ sui lati compresi nella striscia (ma come si vedrà all'interno dell'ultimo capitolo, nella maggior parte degli esempi sorgono dei problemi tecnici).

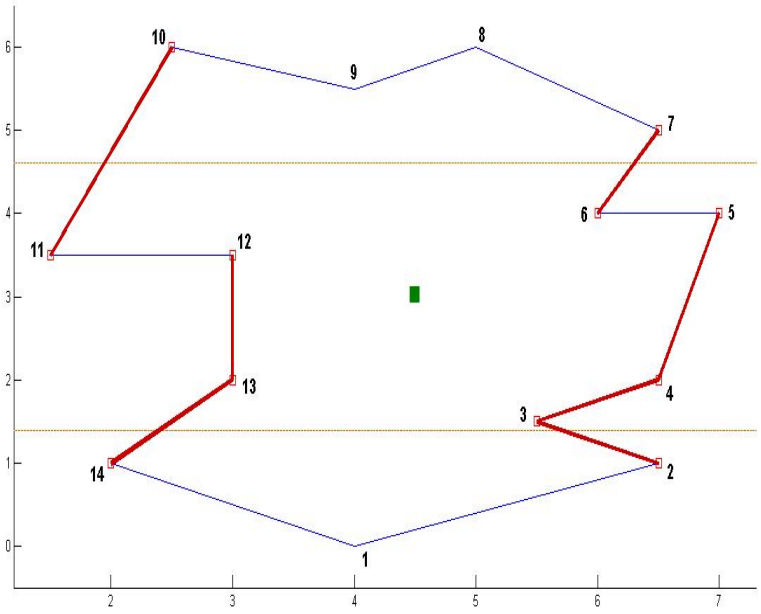
Il seguente esempio mostra come agiscono le due funzioni sul bordo di un poligono di 14 lati, con vertici inseriti e numerati in senso antiorario. I dati sono i seguenti:

$$\mathcal{A} = \begin{pmatrix} 4 & 0 \\ 6.5 & 1 \\ 5.5 & 1.5 \\ 6.5 & 2 \\ 7 & 4 \\ 6 & 4 \\ 6.5 & 5 \\ 5 & 6 \\ 4 & 5.5 \\ 2.5 & 6 \\ 1.5 & 3.5 \\ 3 & 3.5 \\ 3 & 2 \\ 2 & 1 \end{pmatrix}$$

$\sigma_x^2 = 0.1$, $\sigma_y^2 = 0.09$, $\mu_x = 4.5$, $\mu_y = 3$, $\rho = 0.5$, $tol =$ `1E-07`.



(a) *'VertexHorizontal'*.



(b) *'VertexNotHorizontal'*.

Figura 2.4: Esempio di calcolo delle catene.

Nella figura 2.4 in blu è disegnato il bordo del poligono, in marrone chiaro le due rette che delimitano la striscia, in verde il punto di media. I lati evidenziati in rosso invece compongono le catene: queste nel caso (a) (in cui si applica 'VertexHorizontal') sono 2, mentre nel (b) (dove si applica 'VertexNotHorizontal') sono 4. Infatti 'VertexHorizontal' restituisce in output:

$$\mathcal{B}egin = \begin{pmatrix} 2 \\ 10 \end{pmatrix}$$

$$\mathcal{F}inal = \begin{pmatrix} 7 \\ 4 \end{pmatrix}$$

mentre 'VertexNotHorizontal':

$$\mathcal{B}egin = \begin{pmatrix} 2 \\ 6 \\ 10 \\ 12 \end{pmatrix}$$

$$\mathcal{F}inal = \begin{pmatrix} 5 \\ 7 \\ 11 \\ 14 \end{pmatrix}.$$

Si nota facilmente che in un caso come questo i lati agli estremi delle catene, non sono completamente contenuti nella striscia $\{y \in \mathbb{R} : |F(x,y)| > \epsilon\}$. Quindi in realtà con questo metodo l'approssimazione che viene fatta non è quella inizialmente ipotizzata in (2.5) ma la seguente:

$$\oint_{\partial\Omega} F(x,y)dy \approx \int_{\bigcup_k C_k} F(x,y)dy \tag{2.6}$$

dove $C_k = \bigcup_j [V_{k,j}, V_{k,j+1}]$ è la catena k-esima e $[V_{k,j}, V_{k,j+1}]$ è il suo lato j-esimo, di vertici $V_{k,j}$,$V_{k,j+1}$. Nella pratica si va cioè ad integrare non sulla porzione di bordo $\partial\Omega$ contenuta esattamente nella striscia, ma sulla sequenza di lati che entrano, escono o sono contenuti nella stessa. D'altra parte a livello computazionale non si avrebbero vantaggi rilevanti a calcolare i punti di intersezione determinati dai lati agli estremi delle catene con le rette $y = \sigma_y^2 + h(\sigma_y^2)$ e $y = \sigma_y^2 - h(\sigma_y^2)$ che delimitano la striscia, considerando cioè solo catene propriamente contenute in essa.

Per l'integrazione si ricorre alla funzione predefinita del Matlab 'quadgk' (cfr. [4]), la quale utilizza una formula di quadratura adattativa di Gauss-Kronrod (cfr. [3]). Essa può essere chiamata nel modo più semplice come segue:

$$\texttt{Q} = \texttt{quadgk}(\texttt{FUN}, \texttt{A}, \texttt{B})$$

dove

Y=FUN(X) è la funzione integranda, accetta come argomento un vettore X e ritorna un vettore risultante Y, contenente i valori di FUN valutata in ogni elemento di X;

A,B sono gli estremi di integrazione;

1E-10 è tolleranza di default per l'errore assoluto.

La formulazione più completa, presente in questo codice è la seguente:

$$\texttt{Q} = \texttt{quadgk}(\texttt{FUN}, \texttt{A}, \texttt{B}, \texttt{PARAM1}, \texttt{VAL1}, \texttt{PARAM2}, \texttt{VAL2}, ...).$$

In questo caso si aggiungono specifici valori dei parametri opzionali:

AbsTol tolleranza errore assoluto,

RelTol tolleranza errore relativo,

Waypoints vettore dei punti (intermedi) di integrazione.

Tutte le componenti di Waypoints, assieme anche ad A e B, sono numeri complessi e l'integrazione viene così compiuta su un cammino o meglio una spezzata nel piano complesso (che in questo caso sarà una catena o l'intero bordo del poligono, interpretando i vertici come punti nel piano complesso e prendendo poi la parte immaginaria del risultato, cfr. (2.8)): da A al primo punto, poi dal primo al secondo e così via, fino al segmento che congiunge l'ultimo punto con B.

In 'MySoftware' (vedere cap. 5) la funzione 'quadgk' viene applicata in tre modi diversi per la valutazione di $F(x, y)$ in (2.3): la prima sulle catene comprendenti i lati orizzontali (cioè quelle che si ottengono con 'VertexHorizontal'), la seconda sulle catene senza tali lati (cioè quelle che si ricevono in output da 'VertexNotHorizontal') e la terza su tutto il bordo del poligono $\partial\Omega$. In tutti e tre i casi i punti intermedi di integrazione contenuti nel vettore Waypoints sono i vertici del poligono: ascissa e ordinata di ciascun vertice vengono trasformate rispettivemante nella parte reale e immaginaria di un numero complesso. Gli estremi A e B diventano di volta in volta i vertici iniziale e finale di ciasuna catena di lati, mentre nel terzo caso coincidono entrambi con il primo, o $n+1$-esimo, vertice del poligono.

Sia $C_k = \bigcup_j [V_{k,j}, V_{k,j+1}]$ la k-esima catena e $[V_{k,j}, V_{k,j+1}]$ il suo lato j-esimo, di vertici $V_{k,j} = (\alpha_{k,j}, \beta_{k,j})$ e $V_{k,j+1} = (\alpha_{k,j+1}, \beta_{k,j+1})$. Allora un punto del piano cartesiano che appartiene a tale lato

$$\begin{aligned} P_{k,j}(t) &= (x_{k,j}(t), y_{k,j}(t)) \in [V_{k,j}, V_{k,j+1}] \\ &= \left(\frac{\Delta\alpha_{k,j}}{2} t + \frac{\alpha_{k,j} + \alpha_{k,j+1}}{2}, \frac{\Delta\beta_{k,j}}{2} t + \frac{\beta_{k,j} + \beta_{k,j+1}}{2} \right), \end{aligned} \tag{2.7}$$

per $t \in [-1,1]$, visto come punto nel piano complesso diventa: $z_{k,j}(t) = x_{k,j}(t) + iy_{k,j}(t)$, $t \in [-1,1]$. Di conseguenza l'integrale di $F(x+iy) = F(z)$ su C_k risulta:

$$\begin{aligned}
\int_{C_k} F(z)dz &= \sum_j \left(\int_{[V_{k,j}, V_{k,j+1}]} F(z)dz \right) \\
&= \sum_j \left(\int_{-1}^{1} F(z_{k,j}(t)) z'_{k,j}(t)dt \right) \\
&= \sum_j \left(\int_{-1}^{1} F(x_{k,j}(t) + iy_{k,j}(t))(x'_{k,j}(t) + iy'_{k,j}(t))dt \right) \\
&= \sum_j \left(\int_{-1}^{1} F(x_{k,j}(t)) x'_{k,j}(t)dt \right) + i \underbrace{\sum_j \left(\int_{-1}^{1} F(y_{k,j}(t)) y'_{k,j}(t)dt \right)}_{= \int_{C_k} F(x,y)dy}.
\end{aligned} \tag{2.8}$$

Essendo $\int_{C_k} F(z)dz$ la quantità che calcola 'quadgk' (cfr. [4]), quando viene chiamata come sopra descritto, per integrare $F(x,y)$ in (2.3) sulla catena C_k, si capisce che si deve considerare solo la parte immaginaria del numero complesso `Q`, dato in output dalla funzione. Questo ragionamento vale non solo per le catene, ma anche naturalmente per quando si valuta F sull'intero bordo $\partial\Omega$.

Tornando poi ai parametri opzionali da dare in input a 'quadgk' (cfr. [4]), la tolleranza per l'errore relativo viene imposta sempre nulla, mentre l'altra cambia. Se *tol*, passata in input a 'MySoftware' (cap. 5), è la tolleranza per l'errore assoluto con il quale si vuole venga calcolato l'integrale, allora quando si integra solo sulle catene (comprendenti oppure no i lati orizzontali) viene imposta a 'quadgk' una quantità come parametro opzionale 'AbsTol', che dipende da *tol* (questi particolari riguardanti la stima vengono illustrati nella seguente sessione). Invece per il metodo di integrazione su tutto il bordo del poligono è proprio *tol* che viene data in input a 'quadgk'.

Facendo queste distinzioni sono stati implementati per 'MySoftware', in definitiva, tre metodi (vedere cap. 5): quello che si può denominare 'Metodo delle catene', applicato con la funzione 'VertexHorizontal', lo stesso applicato però con 'VertexNotHorizontal' e infine l'integrazione sull'intero $\partial\Omega$. Si può così valutare quale sia la procedura migliore dal punto di vista computazionale: innanzitutto si capisce se conviene calcolare l'integrale di $F(x,y)$ in (2.3) sull'intero bordo del poligono o approssimarlo con quello di $F(x,y)$ sull'insieme $\{y \in \mathbb{R} : |F(x,y)| > \epsilon\}$. Infine emerge anche l'utilità o meno dell'escludere i lati orizzontali dalle catene che intersecano la striscia.

2.2.2 *Stima dell'errore assoluto e ampiezza della striscia*

Con $C_k = \bigcup_j [V_{k,j}, V_{k,j+1}]$ viene indicata la k-esima catena, mentre con $C_l^{ext} = \bigcup_j [V_{l,j}, V_{l,j+1}]$ la catena esterna l-esima e $[V_{l,j}, V_{l,j+1}]$ il suo lato j-esimo. Le catene esterne sono quelle che si ottengono togliendo le sequenze di lati individuate intersecando la striscia $\{y \in \mathbb{R} : |F(x,y)| > \epsilon\}$ con il poligono stesso.

L'errore assoluto, E, che viene commesso nel 'Metodo delle catene' è somma di due fattori:

$$E = E_{val} + E_{int}.$$

Con E_{val} si intende l'errore di valutazione che viene fatto nella prima parte del programma; se ne può avere una stima ragionando come segue. In base alle notazioni assunte si ha:

$$\oint_{\partial\Omega} F(x,y)dy = \int_{\bigcup_k C_k} F(x,y)dy + \int_{\bigcup_l C_l^{ext}} F(x,y)dy \tag{2.9}$$

con

$$\partial\Omega = \left(\bigcup_k C_k\right) \cup \left(\bigcup_l C_l^{ext}\right).$$

Ricordando poi l'approssimazione (2.6) scelta, si trova allora che:

$$\begin{aligned} E_{val} &= \left| \int_{\bigcup_l C_l^{ext}} F(x,y)dy \right| \\ &= \left| \sum_l \left(\int_{C_l^{ext}} F(x,y)dy \right) \right| \\ &= \left| \sum_l \left(\sum_j \left(\int_{[V_{l,j},V_{l,j+1}]} F(x,y)dy \right) \right) \right| \\ &\leq \sum_l \left(\sum_j \left(\left| \int_{[V_{l,j},V_{l,j+1}]} F(x,y)dy \right| \right) \right). \end{aligned} \tag{2.10}$$

Ora la parametrizzazione del lato j-esimo della catena esterna l-esima, $[V_{l,j}, V_{l,j+1}]$, è la stessa di quella vista nella precedente sessione: presi i vertici $V_{l,j} = (\alpha_{l,j}, \beta_{l,j})$, $V_{l,j+1} = (\alpha_{l,j+1}, \beta_{l,j+1})$ si ha

$$\begin{aligned} P_{l,j}(t) &= (x_{l,j}(t), y_{l,j}(t)) \in [V_{l,j}, V_{l,j+1}] \\ &= \left(\frac{\Delta\alpha_{l,j}}{2} t + \frac{\alpha_{l,j} + \alpha_{l,j+1}}{2}, \frac{\Delta\beta_{l,j}}{2} t + \frac{\beta_{l,j} + \beta_{l,j+1}}{2} \right), \end{aligned} \tag{2.11}$$

per $t \in [-1,1]$. Essendo $y'_{l,j}(t) = \frac{\Delta\beta_{l,j}}{2}$ si ottiene:

$$\begin{aligned}
\left| \int_{[V_{l,j},V_{l,j+1}]} F(x,y)dy \right| &= \left| \int_{-1}^{1} F(x_{l,j}(t), y_{l,j}(t)) y'_{l,j}(t) dt \right| \\
&\leq \int_{-1}^{1} |F(x_{l,j}(t), y_{l,j}(t))| \frac{|\Delta\beta_{l,j}|}{2} dt \\
&\leq \|F\|_{\infty,[V_{l,j},V_{l,j+1}]} \frac{|\Delta\beta_{l,j}|}{2} \underbrace{\int_{-1}^{1} dt}_{=2} \\
&\leq \epsilon \, |\Delta\beta_{l,j}| \qquad (2.12)
\end{aligned}$$

dove $\|F\|_{\infty,[V_{l,j},V_{l,j+1}]} = max\{|F(x,y)| \, | (x,y) \in [V_{l,j}, V_{l,j+1}]\}$. Siccome $[V_{l,j}, V_{l,j+1}] \in \bigcup_l C_l^{ext}$, si ha $|F(x,y)| \leq \epsilon \; \forall (x,y) \in [V_{l,j}, V_{l,j+1}]$ e perció $\|F\|_{\infty,[V_{l,j},V_{l,j+1}]} \leq \epsilon$: questo giustifica l'ultimo passaggio in (2.12).
A questo punto si arriva alla stima di E_{val}:

$$\begin{aligned}
E_{val} &\leq \sum_l \sum_j \epsilon \, |\Delta\beta_{l,j}| \\
&= \epsilon \left(\sum_l \sum_j |\Delta\beta_{l,j}| \right) \\
&= \epsilon \sum_{[V_{l,j},V_{l,j+1}] \in \bigcup_l C_l^{ext}} |\Delta\beta_{l,j}| \\
&\leq \epsilon \sum_{\partial\Omega} |\Delta\beta|
\end{aligned} \qquad (2.13)$$

dove il termine $\sum_{\partial\Omega} |\Delta\beta|$ rappresenta la somma, per tutti i lati del poligono, dei moduli delle differenze tra le ordinate dei due vertici di ciascun lato.

E_{int} è invece l'errore totale di integrazione con 'quadgk' (cfr. [4]):

$$E_{int} = \sum_{C_k} E_{int}(C_k)$$

è cioè la somma degli errori di integrazione sulle singole catene.

Come è già stato detto all'inizio, tale codice richiede all'utente di inserire da tastiera anche la tolleranza per l'errore assoluto, *tol*, con la quale si vuole che venga calcolato il valore dell'integrale. Di conseguenza deve verificarsi che:

$$E = E_{val} + E_{int} \leq \epsilon \sum_{\partial\Omega} |\Delta\beta| + \sum_{C_k} E_{int}(C_k) \leq tol.$$

Si è deciso di distribuire equamente tale tolleranza, cioè si chiede che:

$$\epsilon \sum_{\partial\Omega} |\Delta\beta| = \frac{tol}{2}$$

e

$$\sum_{C_k} E_{int}(C_k) \leq \frac{tol}{2}.$$

Dalla prima di queste due disuguaglianze ricaviamo la semi-ampiezza della striscia $\{y \in \mathbb{R} : |F(x,y)| > \epsilon\}$, infatti:
$\epsilon \sum_{\partial\Omega} |\Delta\beta| = \frac{tol}{2} \Rightarrow \epsilon = \frac{tol}{2} \frac{1}{(\sum_{\partial\Omega} |\Delta\beta|)} \Rightarrow |F(x,y)| \leq \epsilon = \frac{tol}{2} \frac{1}{(\sum_{\partial\Omega} |\Delta\beta|)} \Leftrightarrow$
$\frac{1}{2} \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(y-\mu_y)^2}{2\sigma_y^2}\right) \underbrace{\left| erf\left(\frac{\sigma_x\rho(\mu_y - y) + \sigma_y(x-\mu_x)}{\sqrt{2(1-\rho^2)}\sigma_x\sigma y}\right)\right|}_{\leq 1} \leq \epsilon \Leftrightarrow$

$$\exp\left(-\frac{(y-\mu_y)^2}{2\sigma_y^2}\right) \leq 2\epsilon\sqrt{2\pi\sigma_y^2} \Rightarrow -\frac{(y-\mu_y)^2}{2\sigma_y^2} \leq \log\left(2\epsilon\sqrt{2\pi\sigma_y^2}\right) \Rightarrow$$

$$(y-\mu_y)^2 \geq -2\sigma_y^2 \log\left(2\epsilon\sqrt{2\pi\sigma_y^2}\right). \tag{2.14}$$

La (2.14) è un vincolo effettivo se

$$-2\sigma_y^2 \log\left(2\epsilon\sqrt{2\pi\sigma_y^2}\right) \geq 0 \Rightarrow \log\left(2\epsilon\sqrt{2\pi\sigma_y^2}\right) \leq 0$$

$$\Rightarrow \left(2\epsilon\sqrt{2\pi\sigma_y^2}\right) \leq 1 \ \Rightarrow \sigma_y^2 \leq \frac{1}{8\pi\epsilon^2}\ ,$$

e in questo caso diventa

$$\begin{aligned} |y-\mu_y| &\geq \sqrt{-2\sigma_y^2 \log\left(2\epsilon\sqrt{2\pi\sigma_y^2}\right)} \\ &= \sqrt{-2\sigma_y^2 \log\left(\frac{tol}{(\sum_{\partial\Omega}|\Delta\beta|)}\sqrt{2\pi\sigma_y^2}\right)} = h(\sigma_y^2). \end{aligned} \tag{2.15}$$

$h(\sigma_y^2)$ viene così assunta come la semiampiezza della striscia: il suo valore dipende sia da σ_y^2 sia da *tol*. Se si fissa σ_y^2 e la si considera come una funzione della variabile *tol*, si trova che:

$$D_{tol}(h(\sigma_y^2)) = \underbrace{\frac{-\sigma_y^2}{\sqrt{-2\sigma_y^2 \log\left(\frac{tol}{(\sum_{\partial\Omega}|\Delta\beta|)}\sqrt{2\pi\sigma_y^2}\right)}}}_{<0} \underbrace{\frac{1}{tol\sqrt{2\pi\sigma_y^2}}}_{>0} < 0, \tag{2.16}$$

cioè h è funzione strettamente descrescente di *tol*. D'altra parte risulta anche intuitivo che più piccola viene richiesta la tolleranza, più larga dovrà essere la striscia.

Poi h può essere riscritta così:

$$\begin{aligned} h(\sigma_y^2) &= \sigma_y \sqrt{-2\log\left(2\epsilon\sqrt{2\pi\sigma_y^2}\right)} \\ &= \sigma_y \sqrt{\log\left(\frac{1}{8\pi\epsilon^2\sigma_y^2}\right)}. \end{aligned} \tag{2.17}$$

Tenendo ora costante *tol* e prendendo σ_y^2 come variabile si verifica che:

$$h'(\sigma_y^2) = \underbrace{\sqrt{\log\left(\frac{1}{8\pi\epsilon^2\sigma_y^2}\right)}}_{\geq 0} + \underbrace{\frac{8\pi\epsilon^2\sigma_y^4}{\sqrt{log\left(\frac{1}{8\pi\epsilon^2\sigma_y^2}\right)}}}_{\geq 0} \geq 0.$$

L'ultima disuguaglianza è giustificata da quanto detto sopra:

$$\sigma_y^2 \leq \frac{1}{8\pi\epsilon^2} \Rightarrow \frac{1}{8\pi\epsilon^2\sigma_y^2} \geq 1 \Rightarrow \log\left(\frac{1}{8\pi\epsilon^2\sigma_y^2}\right) \geq 0.$$

Quindi h è funzione crescente di σ_y^2: più grande è la varianza della gaussiana 1-dimensionale, più tale curva sarà piatta e, a parità di tolleranza, più larga dovrà essere la striscia.

Dalla seconda disuguaglianza si ottiene invece la tolleranza da dare in input a quadgk (cfr. [4]) per ciascuna catena:

$$\sum_{C_k} E_{int}(C_k) \leq \frac{tol}{2} \Rightarrow E_{int}(C_k) \leq \frac{tol}{2card\left(\{C_k\}\right)},$$

dove si è supposto che l'errore di integrazione $E_{int}(C_i)$ sia lo stesso per ognuna delle successioni di lati.
Nel programma si prende come valore di riferimento per l'integrale esatto, quello ottenuto integrando $F(x, y)$ in (2.3) su tutto il bordo del poligono $\partial\Omega$ con la funzione 'quadgk', alla quale viene data in input, come parametro opzionale, una tolleranza molto bassa per l'errore assoluto (`1E-14`).

Poi, per valutare l'errore effettivo, si prende il valore dell'integrale sulle catene comprensive dei lati orizzontali e si calcola E come la differenza, in modulo, tra quello e l'integrale esatto. Lo stesso procedimento viene adottato quando si valuta $F(x, y)$ in (2.3) sulle catene senza lati orizzontali e su tutto il poligono (dando in input a 'quadgk' la tolleranza *tol*).

2.3 polygauss

Questo secondo codice [11] integra f sul poligono $\Omega \in \mathbb{R}^2$ grazie ad una formula di cubatura del tipo:

$$\int\int_{\Omega} f(x,y)dxdy \approx \sum_{(\xi,\eta)\in\Xi_{2n-1}} w_{\xi,\eta} f(\xi,\eta).$$

Questa viene 'costruita' appunto utilizzando il teorema di Green (cfr. [1]) e una formula di quadratura di Gauss-Legendre (cfr. [3]); risulta esatta per tutti i polinomi in due variabili di grado al più $2n-1$, e stabile (cioè tale che $\sum_{(\xi,\eta)\in\Xi_{2n-1}} |w_{\xi,\eta}|$ é limitata).

Prima di illustrare i particolari del metodo, è sicuramente utile ricordare un pó di teoria (cfr. [2],[3]).

2.3.1 *Quadratura e cubatura numerica*

L'obiettivo della quadratura numerica è quello di sostituire un problema nel continuo, con uno nel discreto: dati cioè $n+1$ punti distinti $x_0, \ldots, x_n \in [a,b]$ si approssima $\phi(f) = \int_b^a f(x)w(x)dx$, $w \in L^1(a,b)$, detto 'funzionale integrale', con $\phi_n(f) = \sum_{j=0}^n w_j f(x_j)$, detto 'funzionale formula di quadratura', una somma pesata di valori della funzione integranda f, in cui i punti x_j sono i nodi e i coefficienti w_j sono i pesi.

La prima classe di formule di quadratura che si incontra, sono quelle algebriche (o interpolatorie): esse si ottengono integrando al posto di f il suo polinomio interpolatore di grado n. Si capisce che la costruzione di tali formule è strettamente legata ad un caso particolare, quello polinomiale, della teoria dell'interpolazione. In tale ambito si considera in genere un sottospazio $S = \langle \Phi_1, \ldots, \Phi_N \rangle \in C(K)$ $(dimS = N)$ dello spazio delle funzioni continue su un compatto $K \in \mathbb{R}^d, d \geq 1$ e N punti di interpolazione $\{x_1, \ldots, x_N\} \subset K$. Interpolare significa trovare $\Phi \in S$, $\Phi = \sum_{j=1}^N c_j \Phi_j$ tale che
$\Phi(x_i) = f(x_i)\ \forall 1 \leq i \leq N \Leftrightarrow \sum_{j=1}^N c_j \Phi_j(x_i) = f(x_i)\ \forall 1 \leq i \leq N$.
Bisogna cioè risolvere il sistema $Vc = \{f(x_i)\}$, dove $V = \{v_{i,j}\} = \{\Phi_j(x_i)\}$ é la matrice di Vandermonde generalizzata.

L'interpolante risulta allora essere:

$$L_{\{x_i\}} f(x) = \sum_{j=1}^{N} f(x_i) l_{x_i(x)}$$

con $l_i(x) = l_{x_i}(x) = \frac{\det(V(x_1,\ldots,x_{i-1},x,x_{i+1},\ldots,x_N))}{\det(V(x_1,\ldots,x_N))}$ e $l_i(x_k) = \delta_{i,k}$.
Quando $S = \mathbb{P}_n = \langle 1, x_1, \ldots, x_n \rangle$ $(dimS = dim\mathbb{P}_n = n+1 = N)$, il polinomio interpolatore di f di grado n è:

$$L_n f(x) = \sum_{i=1}^{n+1} f(x_i) l_{x_i}(x),$$

e in tal caso $l_{x_i}(x) = \prod_{k \neq i} \frac{(x-x_k)}{(x_i-x_k)}$ sono i cosiddetti polinomi elementari di Lagrange e $V = \{v_{i,j}\} = \left\{x_i^{j-1}\right\}$.

Tornando allora alle formule algebriche, viene fatta la seguente approssimazione: $\phi(f) \approx \phi(L_n(f))$, poi si dimostra che effettivamente $\phi(L_n f) = \sum_{j=0}^{n} w_j f(x_j)$ con $w_j = \phi(l_j)$ e che tali formule sono esatte su $\mathbb{P}_n$: cioé $\phi_n(p) = \phi(p)\ \forall p \in \mathbb{P}_n$.

Il secondo tipo sono le formule composte, che si generano integrando al posto di f la sua interpolante composta polinomiale a tratti di grado s, con s che è un multiplo di n e $n+1$ è sempre il numero di nodi dove viene campionata la funzione.

Infine la terza ed ultima classe, di cui fa parte anche la formula di quadratura di Gauss-Legendre, è quella delle gaussiane. Queste in realtà sono una sottoclasse di quelle algebriche e vengono create usando come nodi gli $n+1$ zeri di Φ_{n+1}, polinomio ortogonale di grado $n+1$ rispetto al prodotto scalare per $w \in C(a,b) \cap L^1(a,b)$, $w \geq 0$. Preso lo spazio

$$L_w^2(a,b) = \left\{ f : \int_b^a |f(x)|^2 w(x)dx < \infty, w \in L_+^1(a,b) \right\}$$

e due funzioni $f, g \in L_w^2(a,b)$, allora il prodotto scalare indotto dalla 'funzione peso' w è

$$(f,g)_w := \int_b^a f(x)g(x)w(x)dx.$$

Partendo dalla base canonica di $\mathbb{P}_n = \langle 1, x_1, \ldots, x_n \rangle$ ed utilizzando il procedimento di ortogonalizzazione di Gram-Schmidt si arriva alla base ortogonale $\langle \Phi_0, \Phi_1, \ldots, \Phi_n \rangle$: tali polinomi hanno quindi tutti la proprietà

$$(\Phi_i, \Phi_j)_w = \int_b^a \Phi_i(x)\Phi_j(x)w(x)dx = 0$$

$\forall i \neq j$.

Di conseguenza se $x_0, \ldots, x_n$ sono gli zeri di $\Phi_{n+1}(x)$ e $L_n f(x)$ è il polinomio interpolatore di grado n di f, ottenuto campionando la funzione su $x_0, \ldots, x_n$, si dimostra di nuovo che il 'funzionale formula di quadratura gaussiana' diventa una somma pesata, cioè: $\phi_n(f) = \phi(L_n f(x)) = \sum_{j=0}^{n} w_j f(x_j)$.

Le formule di questo terzo tipo sono sicuramente esatte su $\mathbb{P}_n$, in quanto algebriche, ma si può provare che lo sono anche su tutti i polinomi, fino a quelli di grado $2n+1$.

Il grado di esattezza di una formula algebrica è definito come il massimo numero naturale m_n tale che $\phi_n(p) = \phi(p)\ \forall p \in \mathbb{P}_{m_n}$, con $n \leq m_n \leq 2n+1$.

Una formula di quadratura si dice convergente sulla funzione f se

$$\lim_{n \to +\infty} \phi_n(f) = \phi(f).$$

In particolare qualsiasi tipo di formula algebrica è convergente se la corrispondente interpolazione polinomiale lo è, infatti: preso il funzionale lineare e continuo $\phi : (C[a,b], \|\cdot\|_\infty) \longmapsto (\mathbb{R}, |\cdot|)$ e la sua norma

$$\|\phi\| = \sup_{f \neq 0} \frac{|\phi(f)|}{\|f\|_\infty} = \sup_{\|f\|_\infty = 1} |\phi(f)|$$

dove $\|f\|_\infty = max\{|f(x)| \, | x \in [a,b]\}$, si trova

$$\begin{aligned} |\phi(f)| &= \left| \int_b^a f(x) w(x) dx \right| \leq \int_b^a |f(x)| \, |w(x)| \, dx \\ &\leq \|f\|_\infty \int_b^a |w(x)| \, dx \\ &= \|f\|_\infty \|w\|_1 \end{aligned} \tag{2.18}$$

$\Rightarrow \|\phi\| \leq \|w\|_1 < +\infty$ dato che $w \in L^1(a,b)$. Di conseguenza:

$$\begin{aligned} |\phi_n(f) - \phi(f)| &= |\phi(L_n(f) - \phi(f)| \\ &= |\phi(L_n f - f)| \\ &\leq \underbrace{\|\phi\|}_{<\infty} \|L_n f - f\|_\infty \end{aligned} \tag{2.19}$$

cioè se $\|L_n f - f\|_\infty \to 0$ per $n \to \infty$, allora $|\phi_n(f) - \phi(f)| \to 0$ per $n \to \infty$, o equivalentemente $\lim_{n \to +\infty} \phi_n(f) = \phi(f)$.

Quindi la discussione si sposta di nuovo nell'ambito dell'interpolazione: occorre fare una stima della quantità $\|L_n f - f\|_\infty$, cioè capire di quanto i grafici di f e della sua interpolante polinomiale distino tra loro.

Con $L : (C(K), \|\cdot\|_\infty) \longmapsto S$ si indica l'operatore di interpolazione lineare continuo (limitato) e con

$$\|L\| = \sup_{f \neq 0} \frac{\|Lf\|_\infty}{\|f\|_\infty} = \sup_{\|f\|_\infty = 1} \|Lf\|$$

la sua norma operatoriale. Si dimostra che $\forall f \in C(K)$, $\|Lf\|_\infty \leq \|L\| \, \|f\|_\infty$. Si prende poi Φ^*, il polinomio di migliore approssimazione di f, che come si può provare esiste sempre ed è tale che $\inf_{\Phi \in S} \|f - \Phi\| = \min_{\Phi \in S} \|f - \Phi\| = \|f - \Phi^*\|$.
Si ha così che:

$$\begin{aligned} \|Lf - f\|_\infty &= \left\| Lf - L\Phi^* + \underbrace{L\Phi^* - \Phi^*}_{=0} + \Phi^* - f \right\|_\infty \\ &\leq \|Lf - L\Phi^*\|_\infty + \|\Phi^* - f\|_\infty = \|L(f - \Phi^*)\|_\infty + \|\Phi^* - f\|_\infty \\ &\leq \|L\| \, \|f - \Phi^*\|_\infty + \|\Phi^* - f\|_\infty = (1 + \|L\|) \, \|\Phi^* - f\|_\infty . \end{aligned} \tag{2.20}$$

Lo stesso ragionamento vale anche per $S = \mathbb{P}_n$:

$$\|L_n f - f\|_\infty \leq (1 + \|L_n\|) \|\Phi^* - f\|_\infty$$

con $\|L_n\| =: \Lambda_n$ che è la cosiddetta costante di Lebesgue, che dipende solamente dal grado n e dai nodi scelti. Chiaramente l'interpolazione sarà tanto più buona quanto più lentamente crescerà quest'ultima quantità.

Infine per stimare l'errore di migliore approssimazione $E_n(f) = \|f - \Phi^*\|_\infty$ si ricorre al seguente teorema noto:

Teorema 2 (Teorema di Jackson)**.** *Data $f \in C^r[a,b]$, $r \geq 0$, allora:*

$$E_n(f) \leq c_r osc\left(f^{(r)}; \frac{1}{n}\right) \frac{(b-a)^r}{n(n-1)\cdots(n-r+1)}$$

con $n > r$, c_r una costante indipendente da a, b, f e $osc_{[a,b]}(g,h) = \max_{x,y\in[a,b],|x-y|\leq h} |g(x) - g(y)|$.

Più semplicemente questo risultato dice che $E_n(f) = O(n^{-r})$. Fortunatamente è stato approvato un risultato fondamentale sulla quadratura, che permette di affrontare il problema della convergenza in maniera più diretta:

Teorema 3 (Teorema di Polya-Steklov)**.** *La formula di quadratura ϕ_n è convergente, cioè $\lim_{n\to+\infty} \phi_n(f) = \phi(f)$, $\forall f \in C[a,b]$ fissata, se e solo se:*

1. $\lim_{n\to+\infty} \phi_n(p) = \phi(p)\ \forall p \in \mathbb{P}$
2. $\exists k > 0 : \sum_{j=0}^n |w_j| \leq k$

Si nota che:

$$\begin{aligned} |\phi_n(f)| &= \left|\sum_{j=0}^n w_j f(x_j)\right| \leq \sum_{j=0}^n |w_j|\,|f(x_j)| \\ &\leq \|f\|_\infty \sum_{j=0}^n |w_j| \end{aligned} \tag{2.21}$$

$\Rightarrow \|\phi_n\| \leq \sum_{j=0}^n |w_j|$ (e si può dimostrare che $\|\phi_n\| = \sum_{j=0}^n |w_j|$).

Quindi il teorema di Polya-Steklov afferma che una formula di quadratura è convergente su $C[a,b]$ se e solo se lo è sull'insieme dei polinomi $\mathbb{P}$ e la successione delle norme dei funzionali $\|\phi_n\|$ è limitata.

Per quanto riguarda infine la cubatura numerica, l'obiettivo è quello di calcolare approssimativamente integrali doppi. Preso $\Omega \subset \mathbb{R}^2$ e $f \in C(\Omega)$ si cercano punti di campionamento per tale funzione su un sottoinsieme $X \in \Omega$ discreto e più piccolo possibile e si usano anche in questo caso delle somme pesate:

$$\int\int_\Omega f(x,y)dxdy \approx \sum_{(\xi,\eta)\in X} w_{\xi,\eta} f(\xi,\eta).$$

2.3.2 *Il metodo*

Bisogna ora capire nei particolari come è stata 'costruita' la formula di cubatura (cfr. [10]).

Sia al solito $\Omega \subset \mathbb{R}^2$ un poligono limitato, con bordo $\partial\Omega$ descritto dalla sequenza in senso antiorario di L vertici $V_i = (\alpha_i, \beta_i)\,,\ i = 1 \dots L$:

$$\partial\Omega = [V_1, V_2] \cup [V_2, V_3] \cup \dots \cup [V_L, V_{L+1}]\,, V_{L+1} = V_1.$$

In generale si deve supporre che f sia continua su tutto un rettangolo $\mathcal{R} = [a, b] \times [c, d]$ contenente Ω: condizione che sicuramente è verificata dalla gaussiana in due variabili.

Siano poi $\left\{\tau_j^s\right\}$ e $\left\{\lambda_j^s\right\}$, $1 \le j \le s$, i nodi ed i pesi della formula di quadratura di Gauss-Legendre di grado di esattezza $2s - 1$ (cfr. [3]) su $[-1, 1]$. Chiamata $F(x, y)$ la x-primitiva di f, dal teorema di Green (cfr. [1]) si può scrivere:

$$\begin{aligned}
\int\int_\Omega f(x,y)dxdy &= \oint_{\partial\Omega} F(x,y)dy \\
&= \sum_{i=1}^{L} \int_{[V_i,V_{i+1}]} F(x,y)dy \\
&= \sum_{i\ :\ \Delta\beta_i \neq 0} \int_{[V_i,V_{i+1}]} F(x,y)dy
\end{aligned} \tag{2.22}$$

dove l'ultimo passaggio indica che la somma può essere fatta solamente sui lati non orizzontali all'asse x. Preso $[V_i, V_{i+1}]$ uno di tali lati, se $P_i(t) \in [V_i, V_{i+1}]$ allora

$$P_i(t) = (x_i(t), y_i(t)) = \left(\frac{\Delta\alpha_i}{2}t + \frac{\alpha_i + \alpha_{i+1}}{2}, \frac{\Delta\beta_i}{2}t + \frac{\beta_i + \beta_{i+1}}{2}\right)\ t \in [-1, 1]\,.$$

Quindi utilizzando una formula di quadratura di Gauss-Legendre di grado di esattezza $2n_i - 1$ (cfr. [3]) si ha:

$$\begin{aligned}
\int_{[V_i,V_{i+1}]} F(x,y)dy &= \int_{-1}^{1} F(x_i(t), y_i(t))y_i'(t)dt = \int_{-1}^{1} F(x_i(t), y_i(t))\frac{\Delta\beta_i}{2}dt \\
&= \frac{\Delta\beta_i}{2}\int_{-1}^{1} F(x_i(t), y_i(t))dt \\
&\approx \frac{\Delta\beta_i}{2}\sum_{j=1}^{n_i} \lambda_j^{n_i} F(x_i(\tau_j^{n_i}), y_i(\tau_j^{n_i})).
\end{aligned} \tag{2.23}$$

Poi una x-primitiva di $f(x,y)$ é data da

$$F(x,y) = \int_{\alpha}^{x} f(u,y)du, \alpha \in [a,b]$$

e se $Q_i(t) \in (\alpha, x)$, allora $Q_i(t) = \frac{x-\alpha}{2}t + \frac{x+\alpha}{2}, t \in [-1,1]$.
Così $F(x,y)$ viene a sua volta approssimata con una formula di quadratura di Gauss-Legendre di grado di esattezza $2n-1$ sull'intervallo (α, x):

$$\begin{aligned} F(x,y) &= \int_{-1}^{1} f\left(\frac{x-\alpha}{2}t + \frac{x+\alpha}{2}, y\right)\frac{x-\alpha}{2}dt \\ &= \frac{x-\alpha}{2}\int_{-1}^{1} f\left(\frac{x-\alpha}{2}t + \frac{x+\alpha}{2}, y\right)dt \\ &\approx \frac{x-\alpha}{2}\sum_{k=1}^{n}\lambda_j^n f\left(\frac{x-\alpha}{2}\tau_k^n + \frac{x+\alpha}{2}, y\right). \end{aligned} \tag{2.24}$$

Ovviamente $F(x,y)$ risulta nulla se $x = \alpha$, quindi nella formula di cubatura risultante si possono escludere, oltre agli indici dei lati paralleli all'asse x, anche quelli dei lati che giacciono sulla retta $x = \alpha$.

Preso allora l'insieme

$$\mathcal{I}_{\Omega,\alpha} = \{i : \Delta\beta_i \neq 0\} \cap \{i : \alpha_i \neq \alpha \text{ o } \alpha_{i+1} \neq \alpha\} \subseteq \{1 \dots L\}$$

$$n_i = \begin{cases} n & \text{se } \Delta\alpha_i = 0 \\ n+1 & \text{se } \Delta\alpha_i \neq 0 \end{cases} \tag{2.25}$$

e combinando le due formule (2.18),(2.19),(2.20) si ottiene:

$\int\int_{\Omega} f(x,y)dxdy \approx$
$\sum_{i\in\mathcal{I}_{\Omega,\alpha}} \frac{\Delta\beta_i}{2} \sum_{j=1}^{n_i} \lambda_j^{n_i} \left(\frac{x_i(\tau_j^{n_i})-\alpha}{2} \sum_{k=1}^{n} \lambda_k^n f\left(\frac{x_i(\tau_j^{n_i})-\alpha}{2}\tau_k^n + \frac{x_i(\tau_j^{n_i})+\alpha}{2}, y_i(\tau_j^{n_i}) \right) \right) =$
$\sum_{i\in\mathcal{I}_{\Omega,\alpha}} \sum_{j=1}^{n_i} \sum_{k=1}^{n} \frac{\Delta\beta_i}{4} \left(x_i(\tau_j^{n_i}) - \alpha \right) \lambda_j^{n_i} \lambda_k^n f\left(\frac{x_i(\tau_j^{n_i})-\alpha}{2}\tau_k^n + \frac{x_i(\tau_j^{n_i})+\alpha}{2}, y_i(\tau_j^{n_i}) \right) =$
$\sum_{i\in\mathcal{I}_{\Omega,\alpha}} \sum_{j=1}^{n_i} \sum_{k=1}^{n} w_{i,j,k} f(\xi_{i,j,k}, \eta_{i,j}) =: I_{2n-1}(f)$

con $w_{i,j,k} = \frac{\Delta\beta_i}{4}\left(x_i(\tau_j^{n_i}) - \alpha\right)\lambda_j^{n_i}\lambda_k^n$, $\xi_{i,j,k} = \frac{x_i(\tau_j^{n_i})-\alpha}{2}\tau_k^n + \frac{x_i(\tau_j^{n_i})+\alpha}{2}$ e
$\eta_{i,j} = y_i(\tau_j^{n_i})$.

Questa è la formula cercata, esatta per tutti i polinomi in due variabili di grado al massimo $2n-1$. Perciò quando è applicata ad un polinomio di grado d, è sufficiente scegliere $n \geq \frac{d+1}{2}$ per ottenere l'integrale con un errore pari alla precisione di macchina.

Si nota infine che ponendo $m = card\,(\mathcal{I}_{\Omega,\alpha})$, il numero totale di nodi di cubatura è

$$\mathcal{V} = \mathcal{V}_{n,\Omega,\alpha} = n \sum_{i\in\mathcal{I}_{\Omega,\alpha}} n_i$$

e, come diretta conseguenza della definizione degli n_i in (2.25), si ottiene

$$\frac{L}{2}n^2 \leq mn^2 \leq \mathcal{V} \leq mn\,(n+1) \leq Ln\,(n+1)\,. \tag{2.26}$$

Il codice 'polygauss' viene chiamato nel modo seguente (cfr. [11]):
[`cubatureVal`, `nodesX`, `nodesY`, `weights`] =
`polygauss`(`intfcn`, `N`, `polygonSides`, `rotation`, `P`, `Q`, `quadf`, `cubatureType`, `varargin`).
Gli argomenti in input sono:

- `intfcn`, la funzione integranda che in questo caso è f in (1.1);
- `N`, il grado della formula di quadratura di Gauss-Legendre;
- `polygonSides`, una matrice di dimensione $(L+1) \times 2$ contenente i vertici del poligono ordinati in senso antiorario, con la convenzione che l'ultima riga sia uguale alla prima, cioè l'$L+1$-esimo vertice coincida con il primo;
- `rotation`, indicato con $\mathcal{R} = [a,b] \times [c,d]$ il più piccolo rettangolo contenente Ω, tale variabile viene posta in genere pari a 0, vale altrimenti 1 se si tratta di un poligono convesso, oppure 2. In questo contesto si sceglie sempre `rotation` = `0`.
- [`P`, `Q`], un segmento di vertici `P`, `Q` che dà la direzione di preferenza se `rotation` = `2` (altrimenti `P` = [`0`, `0`], `Q` = [`0`, `0`]);
- `quadf`, viene posta [] se la funzione integranda è definita 'inline', come nella maggioranza dei casi;
- `cubatureType`, serve per scegliere la formula di quadratura, può assumere 4 valori (1:Fejer1, 2:Fejer2, 3:Clenshaw Curtis, 4:Gauss-Legendre). In questo contesto si pone sempre `cubatureType` = `4`;
- `varargin`, parametri opzionali.

Mentre quelli in output:

- `cubatureVal`, il valore di cubatura della funzione specificata in `intfcn` sul bordo descritto in `polygonSides`, che si ottiene utilizzando una formula di quadratura di Gauss-Legendre di grado `N`;
- `nodesX`, `nodesY`, vettori contenenti i nodi della formula;
- `weights`, vettore contenente i pesi della formula.

2.3.3 *Convergenza e stabilitá*

Per quanto riguarda la stabilità, osservando che vale $\sum_{j=1}^{s} \lambda_j^s = 2$ per i pesi della formula di quadratura di Gauss-Legendre e che $|x-\alpha| \leq \max|\alpha - \alpha_i|$, $\forall (x,y) \in \partial\Omega$, si ottiene la seguente stima:

$$\begin{aligned}\sum_{i\in\mathcal{I}_{\Omega,\alpha}}\sum_{j=1}^{n_i}\sum_{k=1}^{n}|w_{i,j,k}| &= \sum_{i\in\mathcal{I}_{\Omega,\alpha}}\sum_{j=1}^{n_i}\sum_{k=1}^{n}|\Delta\beta_i|\left|x_i(\tau_j^{n_i})-\alpha\right| \\ &\leq \max_{i\in\mathcal{I}_{\Omega,\alpha}}|\alpha-\alpha_i|\left(\sum_{i\in\mathcal{I}_{\Omega,\alpha}}|\Delta\beta_i|\right) =: C_{\Omega,\alpha}.\end{aligned} \tag{2.27}$$

Sia poi p_{2n-1}^* il polinomio di migliore approssimazione di f su Ω con grado $2n-1$; si trova allora la seguente stima dell'errore:

$$\begin{aligned}&\left|\int\int_\Omega f(x,y)dxdy - I_{2n-1}(f)\right| \leq \left|\int\int_\Omega f(x,y)dxdy - \int\int_\Omega p_{2n-1}^*(x,y)dxdy\right| \\ &\quad + \left|\int\int_\Omega p_{2n-1}^*(x,y)dxdy - I_{2n-1}(p_{2n-1}^*)\right| + \left|I_{2n-1}(p_{2n-1}^*) - I_{2n-1}(f)\right| \\ &= \left|\int\int_\Omega f(x,y)dxdy - \int\int_\Omega p_{2n-1}^*(x,y)dxdy\right| + \left|I_{2n-1}(p_{2n-1}^*) - I_{2n-1}(f)\right| \\ &\leq \int\int_\Omega \left|f(x,y)-p_{2n-1}^*(x,y)\right|dxdy + \sum_{i\in I_{\Omega,\alpha}}\sum_{j=1}^{n_i}\sum_{k=1}^{n}|w_{i,j,k}|\left|f(\xi_{i,j,k},\eta_{i,j})-p_{2n-1}^*(\xi_{i,j,k},\eta_{i,j})\right| \\ &\quad \leq \int\int_\Omega \left\|f-p_{2n-1}^*\right\|_{\infty,\mathcal{R}}dxdy + \sum_{i\in I_{\Omega,\alpha}}\sum_{j=1}^{n_i}\sum_{k=1}^{n}|w_{i,j,k}|\left\|f-p_{2n-1}^*\right\|_{\infty,\mathcal{R}} \\ &\qquad = \left(\int\int_\Omega dxdy + \sum_{i\in I_{\Omega,\alpha}}\sum_{j=1}^{n_i}\sum_{k=1}^{n}|w_{i,j,k}|\right)\left\|f-p_{2n-1}^*\right\|_{\infty,\mathcal{R}} \\ &\qquad\quad = \left(meas(\Omega) + \sum_{i\in I_{\Omega,\alpha}}\sum_{j=1}^{n_i}\sum_{k=1}^{n}|w_{i,j,k}|\right)E_{2n-1}(f;\mathcal{R})\end{aligned}$$

dove si è usata la notazione

$$\begin{aligned}E_{2n-1}(f;\mathcal{R}) &= \min_{p\in\mathbb{P}_{2n-1}^2}\|f-p\|_{\infty,\mathcal{R}} \\ &= \left\|f-p_{2n-1}^*\right\|_{\infty,\mathcal{R}} \\ &= \max\left\{\left|f(x,y)-p_{2n-1}^*(x,y)\right| \mid (x,y)\in\mathcal{R}\right\}.\end{aligned} \tag{2.28}$$

e che

$$\left| \int \int_{\Omega} p^*_{2n-1}(x,y)dxdy - I_{2n-1}(p^*_{2n-1}) \right| = 0$$

per il fatto che I_{2n-1} è esatta su tutti i polinomi in due variabili, fino a quelli di grado $2n-1$.

Infine per quanto concerne la convergenza della formula per $n \to \infty$, dall'estensione in più variabili del teorema di Jackson (cfr. [6]), si ottiene:

$$\int \int_{\Omega} f(x,y)dxdy = I_{2n-1}(f) + O\left((2n-1)^{-(p+\theta)}\right)$$

per ogni funzione $f \in C^{p+\theta}(\mathcal{R})$, con derivate parziali p-esime continue, $p \geq 0$ e $\theta \in (0,1]$.

Il motivo per cui f è supposta continua su tutto $\mathcal{R}$ e la stima dell'errore coinvolge la norma infinito su $\mathcal{R}$, è che i nodi possono in generale cadere sia all'interno del poligono sia nella regione esterna compresa nel rettangolo.

Capitolo 3

Integrazione con dblquad e twoD

In questo capitolo si vogliono descrivere le caratteristiche e le funzionalità degli altri due codici, 'dblquad' (cfr. [4]) e 'twoD' [9], utilizzati in questo contesto per integrare nuovamente la gaussiana in due variabili sul poligono $\Omega \subset \mathbb{R}^2$. Questi in generale servono per la valutazione di integrali doppi nel piano e affrontano tale problema con approcci completamente diversi rispetto ai precedenti due programmi illustrati.

Il primo, 'dblquad', è una funzione del Matlab che viene chiamata nel modo seguente (cfr. [4])

$$\texttt{Q} = \texttt{dblquad}(\texttt{FUN}, \texttt{XMIN}, \texttt{XMAX}, \texttt{YMIN}, \texttt{YMAX})$$

e approssima l'integrale doppio di $\texttt{FUN}(\texttt{X}, \texttt{Y})$ sul rettangolo di vertici $(\texttt{XMIN}, \texttt{YMIN})$, $(\texttt{XMAX}, \texttt{YMIN})$, $(\texttt{XMAX}, \texttt{YMAX})$, $(\texttt{XMIN}, \texttt{YMAX})$. La funzione $\texttt{Z} = \texttt{FUN}(\texttt{X}, \texttt{Y})$ accetta un vettore `X` ed uno scalare `Y`, mentre restituisce il vettore `Z` contenente i vettori dell'integranda.

'dblquad' permette di scegliere solo la tolleranza per l'errore assoluto, il quale di default è `1E-06` e può essere variato aggiungendo il parametro opzionale `TOL` al momento della chiamata:

$$\texttt{Q} = \texttt{dblquad}(\texttt{FUN}, \texttt{XMIN}, \texttt{XMAX}, \texttt{YMIN}, \texttt{YMAX}, \texttt{TOL})$$

Se, come in questo caso, la regione che si considera non è un rettangolo, ma un dominio più complicato come appunto un poligono, si deve porre l'integranda pari a zero fuori da tale area. Nello specifico si utilizza la funzione 'inpolygon' (cfr. [4]):

$$\texttt{IN} = \texttt{inpolygon}(\texttt{X}, \texttt{Y}, \texttt{XV}, \texttt{YV}).$$

Essa ritorna in output la matrice `IN` della stessa dimensione di `X` e `Y`: $\texttt{IN}(\texttt{p}, \texttt{q}) = \texttt{1}$ se il punto $(\texttt{X}(\texttt{p}, \texttt{q}), \texttt{Y}(\texttt{p}, \texttt{q}))$ è strettamente contenuto od appartiene al bordo del poligono i cui vertici sono specificati nei vettori `XV` e `YV`, altrimenti $\texttt{IN}(\texttt{p}, \texttt{q}) = \texttt{0}$.

La funzione `FUN` che viene data in input a 'dblquad' (cfr. [4]) è allora il prodotto della gaussiana bidimensionale per la funzione caratteristica del dominio, realizzata con 'inpolygon' (cfr. [4]). È chiaro che in questa maniera viene creata una discontinuità artificiale sul bordo del poligono, la quale, come si vedrà nei risultati numerici, è una causa dell'inefficienza di tale metodo.

Nell'ambito della cubatura, l'approccio più naturale per valutare un integrale del tipo

$$I = \int_a^b \int_{c(x)}^{d(x)} f(x,y)dydx \tag{3.1}$$

é quello di scrivere I come $\int_a^b g(x)dx$ e, ogni qualvolta viene richiesto il valore $g(\xi)$, lo si ottiene approssimando

$$g(\xi) = \int_{c(\xi)}^{d(\xi)} f(\xi,y)dy.$$

'dblquad' implementa questa idea con formule adattative 1-dimensionali. In letteratura si trovano però argomentazioni che dissuadono dal seguire questa linea, ritenendo migliore l'approssimazione di $g(\xi)$ con formule non adattative. Inoltre non è chiaro come nel precedente approccio si sfrutti la possibilità di vettorizzare le variabili dell'integranda. Come già detto la funzione 'dblquad' richiede la vettorizzazione solo rispetto alla x, ma tratta la valutazione nella y come un calcolo scalare. Proprio in questo consiste la prima sostanziale differenza tra i due algoritmi: 'twoD' permette infatti di utilizzare la vettorizzazione in maniera più consistente (cfr. [8]).

La sua interfaccia è di semplice utilizzo; l'integrale (3.1) viene approssimato con la seguente chiamata (cfr. [9]):

$$\texttt{Q} = \texttt{twoD}(\texttt{fun}, \texttt{a}, \texttt{b}, \texttt{c}, \texttt{d}).$$

`fun` è la funzione integranda f, che accetta come variabili i vettori `X` e `Y` e restitutisce il vettore $Z = f(X,Y)$ dei corrispondenti valori: $\forall\ (i,j)\ Z_{i,j} = f(X_{i,j}, Y_{i,j})$. La regione di default è un rettangolo generalizzato $a \leq x \leq b$, $c(x) \leq y \leq d(x)$, che viene fornito al programma inserendo i parametri a,b,c,d in input (a e b sono costanti, mentre c e d possono essere costanti o funzioni). C'è inoltre un'opzione '`Sector`' che permette di descrivere regioni in coordinate polari. Ponendo:

```
Q=twoD(fun,a,b,c,d,'Sector',true)
```

si utilizzano la variabile angolare θ, $0 \leq a \leq \theta \leq b$ e quella radiale r, $c(\theta) \leq r \leq d(\theta)$.

In questo contesto, essendo la regione di integrazione un poligono, si è dovuto ricorrere alla funzione 'inpolygon' (cfr. [4]), nella stessa maniera descritta per 'dblquad' (cfr. [4]). Anche qui quindi, si estende la gaussiana

in due variabili a tutto il più piccolo rettangolo contenente il poligono, dandole valore zero al di fuori dello stesso. In questo caso però le conseguenze sono meno negative: con 'twoD' vi è infatti la possibilità di utilizzare la trasformazione opzionale '`Singular`', che viene applicata assegnandole valore '`true`'. Questa, che costituisce la seconda importante differenza rispetto a 'dblquad', fa in modo che 'twoD' restituisca dei risultati attendibili anche per le valutazioni di funzioni integrande singolari sul bordo.

In 'twoD' (cfr. [19]) `ATOL=1E-05` è la tolleranza di default per l'errore assoluto, ma qualsiasi altro valore $ATOL \geq 0$ può essere specificato usando l'opzione '`Abstol`'. La tolleranza di default per l'errore relativo è invece $RTOL = 0$, ma qualunque $RTOL \in [0, 1)$ può essere scelto con l'opzione '`Reltol`'.

Tale programma cerca di calcolare un'approssimazione di `Q`, con un errore stimato E tale che:

$$|E| \leq \max\left(ATOL, RTOL * |Q|\right) = TOL.$$

Nel linguaggio Matlab, il costo di valutazione di un'integranda vettorizzata dipende pesantemente dal numero di prove (o campioni) che si considerano. Si preferisce quindi non sprecare un numero troppo elevato di prove quando l'integrale su una regione non è sufficientemente accurato. Nel caso di 'twoD' la regione è un rettangolo (cfr. [8]): ogni valutazione dell'integranda fornisce abbastanza valori da poter approssimare l'integrale su delle sottoregioni, a loro volta rettangoli, e stimare l'errore su ciascuna di queste. Più valori si prendono su ciascun sottorettangolo, più alto sarà il grado di precisione della formula, ma contemporaneamente più prove saranno state sprecate se l'integrale non è sufficientemente accurato.

Su ognuna di queste regioni più piccole viene adottata una formula di quadratura di Kronrod usata in forma prodotto tensoriale e l'approssimazione è accettata quando la stima del suo errore in valore assoluto è minore di $\frac{TOL}{8}$ volte il rapporto tra l'area del sottorettangolo e quella del rettangolo di partenza. Se questo è verificato, l'approssimazione risultante sommando quelle su ogni sottorettangolo, ha un errore stimato con $\frac{TOL}{8}$, che è sufficientemente piccolo rispetto all'accuratezza richiesta. Lo svantaggio di questa scelta sta nel fatto che non si pone attenzione alla possibilità che gli errori di segno opposto si cancellino tra di loro, diminuendo il valore della somma.

Si nota infine che in 'twoD' c'è una somma corrente sia degli integrali approssimati sui sottorettangoli, sia dei valori assunti dagli errori che passano il test dell'errore locale.

Capitolo 4

Risultati numerici

Come test sono presi in esame tre poligoni che contano rispettivamente 52, 37, 18 lati. Su ognuno si integra una gaussiana bivariata, i cui parametri variano di volta in volta. Ciascuna integrazione è eseguita con tutti e quattro i software a disposizione, scegliendo di fissare la tolleranza $tol =$ `1E-10` da imporre a ciascuno. Si confrontano poi gli effettivi errori assoluti con cui vengono calcolati i risultati, in rapporto al tempo necessario per l'esecuzione del codice. Tutti i test sono stati effettuati in Matlab 7.6.0 (R2008a), su un processore Intel Core i5-2410M, 2.3GHz con 4Gb di RAM. Come già spiegato in 2.2.2, il valore di riferimento per l'integrale viene calcolato integrando la x-primitiva sull'intero bordo, dando in input a 'quadgk' (cfr. [4]) tolleranza `1E-14`. Non in tutti i casi però il risultato viene restituito con un errore assoluto minore di tale quantità e 'quadgk' fa apparire in automatico un messaggio di attenzione (`Warning`). Questo è riportato in seguito negli esempi in cui questo problema si verifica, poichè di conseguenza i valori riportati per 'MySofware' sono fittizi. Per quanto riguarda il primo algoritmo sono stati applicati in sequenza i tre procedimenti descritti nel primo capitolo: il 'Metodo delle catene' prima con la funzione 'VertexHorizontal', caso (a), poi con 'VertexNotHorizontal', caso (b), e per ultima l'integrazione su tutto $\partial\Omega$ (si veda cap. 5). I tempi riportati per il caso (a) sono comprensivi dell'esecuzione della funzione 'VertezHorizontal' (tra parentesi) e della successiva integrazione della x-primitiva $F(x, y)$ in (2.3) sulle sequenze di lati, inclusi quelli orizzontali; mentre nel (b) comprendono l'esecuzione dell'altra funzione 'VertexNotHorizontal' (tra parentesi) e di nuovo l'integrazione di $F(x, y)$ sulle catene senza lati orizzontali. Infine il tempo per l'integrazione su tutto il bordo corrisponde con quello che ci impiega la funzione 'quadgk' applicata un'unica volta all'intera sequenza di lati di $\partial\Omega$. Per 'polygauss' [11] viene riportato anche il grado n necessario per raggiungere il risultato con l'errore assoluto richiesto (`1E-10`). Infine il simbolo *, presente in diversi casi tra i valori di 'twoD' [9], sta ad indicare che il software non riesce a dare in output il valore dell'integrale, in un tempo maggiore ai 5 minuti.

4.1 Poligono con 52 lati

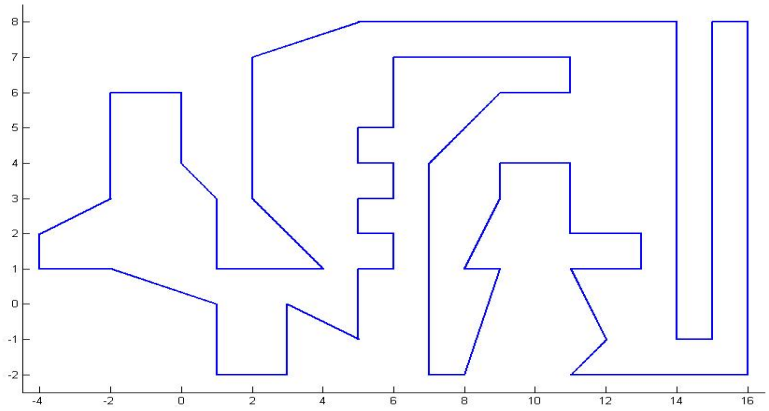


Figura 4.1: Poligono con 52 lati.

Per questo primo poligono vengono illustrati i 3 esempi seguenti:

1.1 Parametri della gaussiana bivariata:
$\sigma_x^2 = 0.1$, $\sigma_y^2 = 0.1$, $\mu_x = 5$, $\mu_y = 1$, $\rho = 0.5$.
Integrale di riferimento: 0.667306621736361.

	MySoft.(a)	MySoft.(b)	Integ. su $\partial\Omega$	polygauss
Errore ass.	4.4E-16	5.5E-16	3.3E-16	2.3E-11(n=83)
Tempo (sec.)	(0)+0.031	(0)+0.062	0.078	0.23

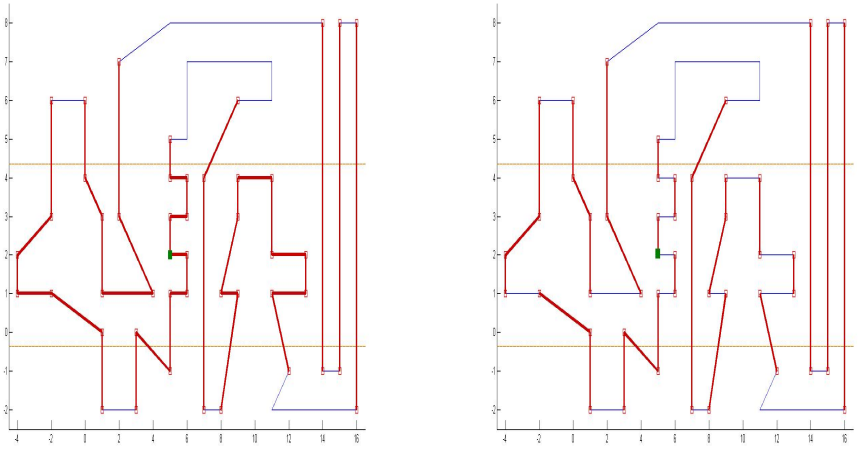
	dblquad	twoD
Errore ass.	2.9E-04	*
Tempo (sec.)	31.7	*

Tabella 4.1: Poligono di 52 lati, risultati esempio 1.1.

1.2 Parametri della gaussiana bivariata:
$\sigma_x^2 = 4$, $\sigma_y^2 = 0.29$, $\mu_x = 14.5$, $\mu_y = 0$, $\rho = 0.99$.
Integrale di riferimento: 0.483677517566564.

Da questi primi risultati si può cominciare ad impostare il confronto tra gli algoritmi.

Per ciascuno dei due esempi i risultati di 'MySoftware' (cap. 5) sono molto buoni: il tempo di esecuzione rimane sempre dell'ordine di centesimi di



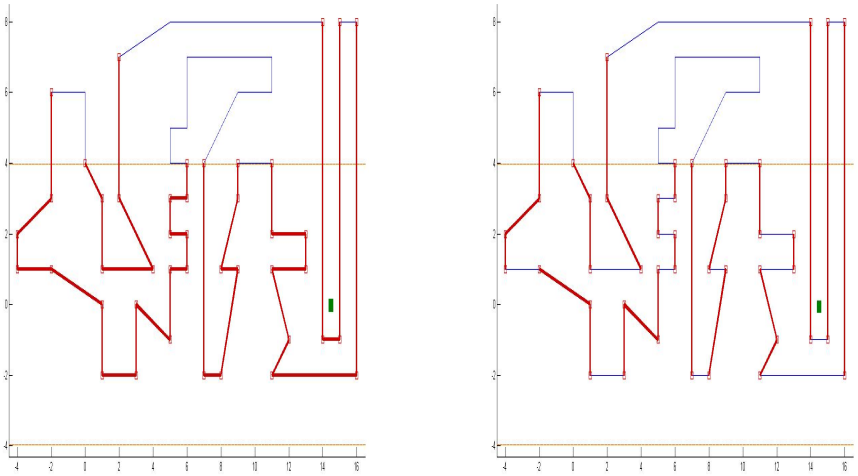
(a) *'VertexHorizontal'*. (b) *'VertexNotHorizontal'*.

Figura 4.2: Poligono con 52 lati, esempio 1.1.

	MySoft.(a)	MySoft.(b)	Integ. su $\partial\Omega$	polygauss
Errore ass.	2.8E-14	2.8E-14	0.0E+00	4.2E-11(n=168)
Tempo (sec.)	(0)+0.047	(0)+0.078	0.062	0.61

	dblquad	twoD
Errore ass.	6.9E-04	*
Tempo (sec.)	15.4	*

Tabella 4.2: Poligono con 52 lati, risultati esempio 1.2.



(a) *'VertexHorizontal'.* (b) *'VertexNotHorizontal'.*

Figura 4.3: Poligono con 52 lati, esempio 1.2.

secondo e l'errore assoluto è inferiore alla tolleranza richiesta ($tol =$ `1E-10`). In particolare nel 'Metodo delle catene' entrambe le funzioni 'VertexHorizontal' e 'VertexNotHorizontal' hanno costo nullo, ossia il tempo impiegato per determinare tali sequenze di lati è prossimo allo zero (più precisamente è inferiore alla sensibilità del CPU timer).

Il primo algoritmo risulta più efficiente, sempre a livello di tempi, nel caso (a), cioè quando l'integrazione della x-primitiva $F(x, y)$ in (2.3) viene compiuta sulle catene comprendenti i lati orizzontali, determinate con 'VertexHorizontal'. Infatti sia in 1.1 che in 1.2, passando dal caso (a) al (b), l'errore assoluto rimane dello stesso ordine (`1E-16` e `1E-14` rispettivamente), ma il tempo di esecuzione aumenta.

Questo sembra in contraddizione con quanto detto in precedenza al capitolo uno: l'altra funzione, 'VertexNotHorizontal', è stata ideata proprio con l'intento di avere a disposizione le sequenze di lati nella striscia, private di quelli orizzontali, dove appunto l'integrale di $F(x, y)$ in (2.3) è pari a zero, per poter così ridurre i costi computazionali e velocizzare l'algoritmo. Durante l'implementazione del codice, ci si accorge però che una tale scelta comporta spesso dei problemi. Infatti quando in generale si integra la x-primitiva sulle catene, si deve chiamare ripetutamente la funzione 'quadgk' (cfr. [4]), in modo da applicarla a ciascuna sequenza di lati. Nella pratica questo viene eseguito con un ciclo, caratterizzato allora da tante iterazioni quante sono le catene su cui integrare la funzione. Dagli esperimenti si è riscontrato che tale ciclo diventa tanto più lento, quanto più elevato è il numero di iterazioni,

ossia il numero di invocazioni di 'quadgk'. Per un poligono come questo, formato da molti lati orizzontali (poco meno della metà del totale), la quantità di catene che li escludono (spesso formate da un singolo lato), è maggiore rispetto alla quantità di quelle che invece li includono: nell'esempio 1.1 ci sono dieci catene di differenza tra il caso (b) e il caso (a), mentre nell'1.2 sono dodici. Si capisce allora perchè il 'Metodo delle catene' applicato con 'VertexNotHorizontal' è il meno efficiente tra i due: il ciclo attraverso il quale viene fatta l'integrazione, conta molte iterazioni in più rispetto all'altro e di conseguenza la sua durata aumenta o addirittura raddoppia.

Proseguendo nell'analisi si riscontra che la performance di 'MySoftware' (cap. 5) con l'integrazione sull'intero $\partial\Omega$, confrontata con l'altro metodo, varia da esempio a esempio. L'ordine dell'errore assoluto che riporta il risultato nell'1.1 (`1E-16`) è lo stesso di quello del 'Metodo delle catene', ma il tempo di esecuzione è più lungo. Nell'1.2, i casi (a) e (b) presentano un errore di `2.8E-14`, mentre integrando su tutto il bordo quello viene approssimato a zero: evidentemente il valore che 'quadgk' restituisce in tal caso è lo stesso di quello preso come riferimento.

Per provare a motivare tale comportamento si deve tenere conto di vari fattori. Nel 'Metodo delle catene' si approssima l'integrale della $F(x, y)$ su tutto il bordo del poligono, con quello di $F(x, y)$ sulle sole catene di lati compresi nella striscia $\{y \in \mathbf{R} : |F(x, y)| > \epsilon\}$. Si può facilmente verificare come la funzione utilizzata per l'integrazione, 'quadgk', abbia dei tempi di esecuzione più lunghi, all'aumentare della lunghezza della sequenza di lati che le viene data in input. Quindi diminuire il numero di lati su cui integrare la x-primitiva, cioè non integrare su tutto il bordo, è sicuramente una buona idea per abbassare i costi computazionali. Se si sceglie allora di utilizzare il 'Metodo delle catene', si deve implementare un ciclo contenente in ogni iterazione la funzione 'quadgk' (cfr. [4]). Questa viene invocata, di volta in volta, per integrare $F(x, y)$ su una singola sequenza di lati contenuti nella striscia. Come già notato prima però, a sua volta questo procedimento può comportare degli svantaggi: il ciclo diventa tanto più lento, quante sono di più, in numero, le sue iterazioni e cioè quante più sono le catene. Si capisce perciò che considerando un esempio in cui ci sono tante sequenze di lati che intersecano la striscia, può venire meno la convenienza di tale metodo. In tal caso invocare una sola volta 'quadgk', dandole in input la sequenza di tutti i lati del poligono (cioè integrare direttamente $F(x, y)$ su $\partial\Omega$), può risultare paragonabile, come tempi, rispetto al ciclo.
In secondo luogo si deve pensare a come è stata impostata la stima nel 'Metodo delle catene': se tol è la tolleranza per l'errore assoluto imposta a tutti e tre i metodi di 'MySofware', allora (indicando con $card(\{C_k\})$ il numero di catene) $atol = \frac{tol}{2card(\{C_k\})}$ diventa la quantità da dare in input a 'quadgk' (cfr. [4]) quando viene invocata per l'integrazione su una singola sequenza di lati nella striscia. Perciò negli esempi risulta che a 'quadgk'

viene data tolleranza `1E-10` per l'integrazione sull'intero bordo, mentre nel 'Metodo delle catene' ogni volta che quadgk viene invocata le si impone *atol*, sicuramente inferiore alla precedente. Dando in input alla funzione una tolleranza più piccola non si può sempre essere certi che quella restituisca il risultato con un errore minore, ma ci si aspetta che il tempo di esecuzione sia maggiore.
Tornando a guardare ora i due esempi, si nota che il numero di lati (inclusi gli orizzontali) compresi nella striscia supera abbondantemente la metà del totale di quelli del bordo. Questo, per tale poligono, comporta che le catene siano numerose e *atol* risulti sempre dell'ordine di `1E-12`. Per quanto detto prima allora, l'elevato numero di iterazioni del ciclo e la tolleranza più piccola imposta a 'quadgk' in ogni iterazione, dovrebbero rendere le performance del 'Metodo delle catene' peggiori di quelle per l'integrazione su tutto $\partial\Omega$. Il fatto che esse in realtà siano migliori in un esempio su due, si giustifica osservando che tale poligono è formato da un numero elevato di lati (52): passando a 'quadgk' una sequenza così lunga, per integrare $F(x, y)$ in (2.3) direttamente su tutto il bordo, il suo tempo di esecuzione sarà comunque elevato.

Per quanto riguarda ancora 'MySoftware', si osserva che il 'Metodo delle catene' restituisce il risultato con un errore assoluto dell'ordine di `1E-16` nell'1.1, mentre nell'1.2 questo diventa circa `1E-14` e contemporaneamente i tempi aumentano rispetto ai precedenti, sia nel caso (a) che nel (b). Per il momento si può pensare che la causa di questa peggiore performance nell'esempio 1.2, stia nella particolare scelta dei parametri della gaussiana bivariata: in tale contesto sono si sono scelte le varianze marginali $\left(\sigma_x^2, \sigma_y^2\right) = (4, 0.29)$, le medie marginali, $(\mu_x, \mu_y) = (14.5, 0)$ (punto che sta al di fuori del poligono) ed un coefficiente di correlazione, $\rho = 0.99$, vicino ad uno.

Legato ai parametri di $f(x, y)$ sembra essere anche l'andamento dei risultati di 'polygauss' [11]. Tale algoritmo compie la sua migliore performance nell'esempio 1.1, dove sono stati scelti $\left(\sigma_x^2, \sigma_y^2\right) = (0.1, 0.1)$,$(\mu_x, \mu_y) = (5, 2)$ e $\rho = 0.5$. Qui il risultato viene restituito con un errore assoluto dell'ordine di `1E-11`, con grado 83 e un tempo misurato pari a 0.23 secondi. Nell'1.2, in cui è riportata la performance peggiore, l'errore rimane circa `1E-11`, ma il costo computazionale aumenta: il tempo di esecuzione è di sei decimi di secondo e il grado 168.

'polygauss' riesce quindi a calcolare l'integrale rispettando la tolleranza richiesta, ma è meno efficiente di 'MySoftware'.

Decisamente più scarsi sono i risultati di 'dblquad' (cfr. [4]): in ciascuno dei due esempi l'errore assoluto con cui viene restituito il risultato è molto superiore alla tolleranza richiesta e i tempi variano da 15 a 31 secondi. Nel precedente capitolo si è già spiegato che questo integratore automatico del Matlab viene molto penalizzato dal fatto di dover ricorrere alla funzione 'inpolygon' (cfr. [4]): ogni volta che, all'interno del codice, questa deve essere valutata, essendo parte dell'integranda, viene perso molto tempo.

Non sono migliori le performance di 'twoD' [9], il quale non riesce mai a calcolare l'integrale di $F(x, y)$ in (2.3) con la tollerenza scelta, $tol =$ `1E-10`, che gli viene passata in input al momento della chiamata. Questo appare da subito strano, considerando il fatto che tale codice ha invece due proprietà che lo avvantaggiano rispetto al precedente: la vettorizzazione e l'opzione 'Singular'. Viene spontaneo supporre che il comportamento di 'twoD' dipenda dall'ordine di grandezza della tolleranza che gli è imposta. Nelle tabelle 4.3 e 4.4 sono stati riportati i risultati di 'dblquad' e 'twoD' quando viene assegnata loro in input $tol =$ `1E-04`.

	dblquad	twoD
Errore ass.	4.2E-01	1.4E-06
Tempo (sec.)	0.17	4.5

Tabella 4.3: Poligono di 52 lati, esempio 1.1, risultati di 'dblquad' e 'twoD' con $tol =$ `1E-04`.

	dblquad	twoD
Errore ass.	2.1E-03	6.6E-06
Tempo (sec.)	0.13	0.47

Tabella 4.4: Poligono di 52 lati, esempio 1.2, risultati di 'dblquad' e 'twoD' con $tol =$ `1E-04`.

Come ci si aspettava le performance di 'twoD' [9] non solo sono più soddisfacenti, ma superano anche quelle di 'dblquad' con $tol =$ `1E-10`. Evidentemente l'algoritmo migliora le sue performance quando la tolleranza che gli viene imposta per l'errore assoluto non è troppo piccola. Al contrario, dando in input a 'dblquad' (cfr. [4]) una tolleranza più grande, i suoi risultati peggiorano.

4.2 Poligono con 37 lati

Per questo secondo poligono vengono illustrati i 5 esempi seguenti:

2.1 Parametri della gaussiana bivariata:
$\sigma_x^2 = 0.02$, $\sigma_y^2 = 0.01$, $\mu_x = 0$, $\mu_y = 1$, $\rho = 0.5$.
Integrale di riferimento: 0.999971552913487.

2.2 Parametri della gaussiana bivariata:
$\sigma_x^2 = 4$, $\sigma_y^2 = 0.00001$, $\mu_x = 0$, $\mu_y = 1$, $\rho = 0$.
Integrale di riferimento: 0.395561146936899.

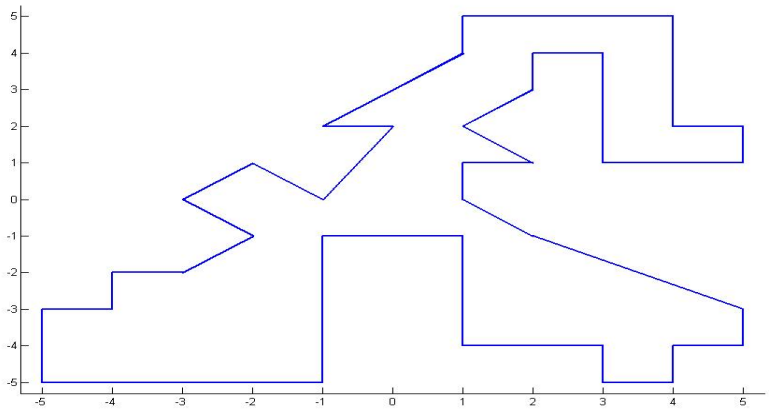


Figura 4.4: Poligono con 37 lati.

	MySoft.(a)	MySoft.(b)	Integ. su $\partial\Omega$	polygauss
Errore ass.	4.4E-16	2.2E-16	9.9E-16	8.0E-11(n=38)
Tempo (sec.)	(0)+0.047	(0)+0.047	0.062	0.031

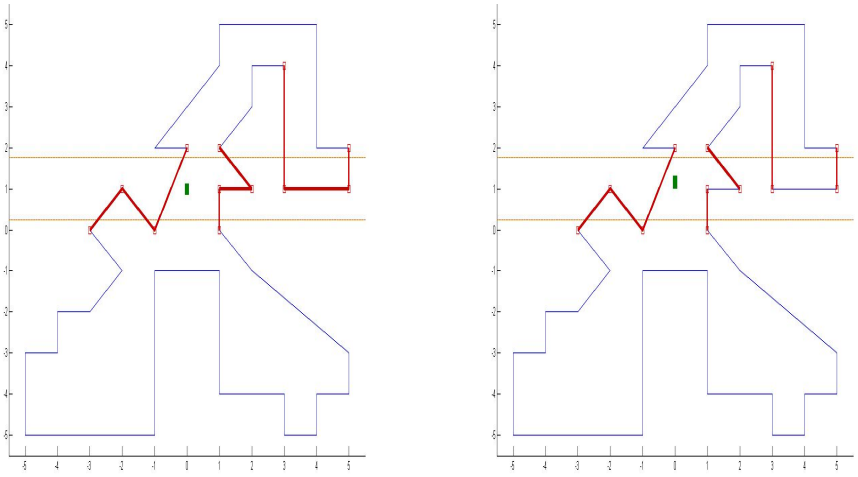
	dblquad	twoD
Errore ass.	6.9E-04	*
Tempo (sec.)	15.8	*

Tabella 4.5: Poligono di 37 lati, risultati esempio 2.1.

	MySoft.(a)	MySoft.(b)	Integ. su $\partial\Omega$	polygauss
Errore ass.	3.9E-16	1.2E-15	2.2E-16	1.7E-02(n=350)
Tempo (sec.)	(0)+0.093	(0)+0.093	0.19	2.29

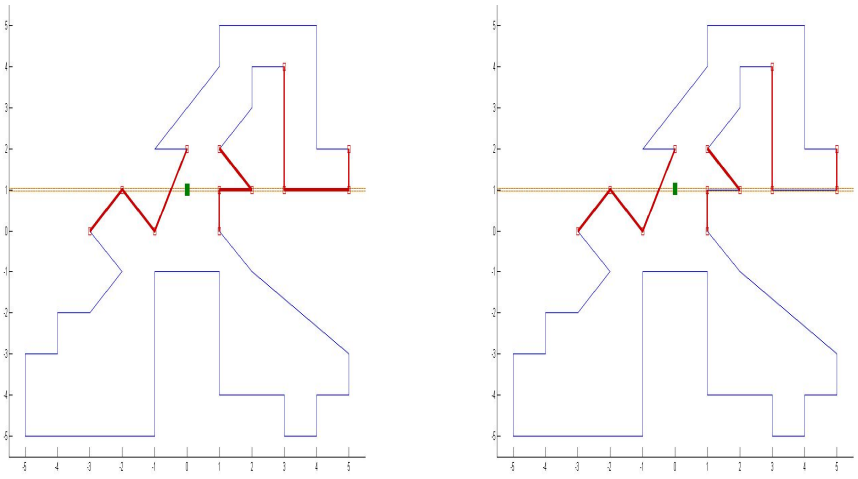
	dblquad	twoD
Errore ass.	4.0E-01	4.0E-01
Tempo (sec.)	0.14	0.031

Tabella 4.6: Poligono con 37 lati, risultati esempio 2.2.



(a) *'VertexHorizontal'*. (b) *'VertexNotHorizontal'*.

Figura 4.5: Poligono con 37 lati, esempio 2.1.



(a) *'VertexHorizontal'*. (b) *'VertexNotHorizontal'*.

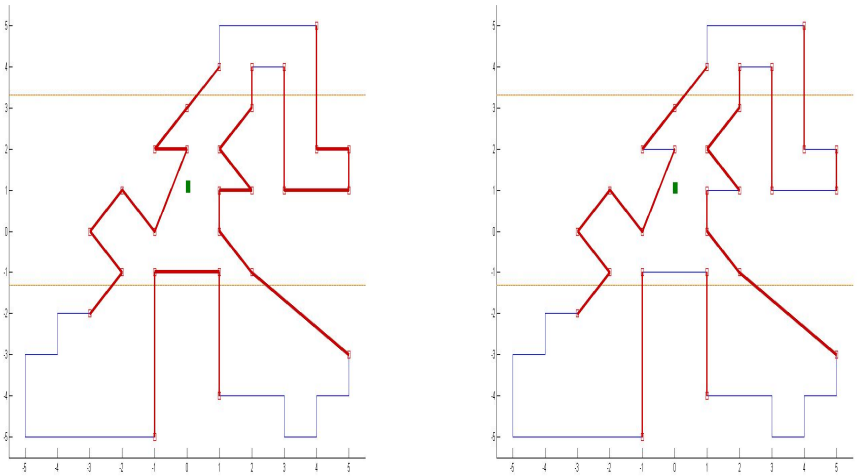
Figura 4.6: Poligono con 37 lati, esempio 2.2.

2.3 Parametri della gaussiana bivariata:
$\sigma_x^2 = 0.02$, $\sigma_y^2 = 0.1$, $\mu_x = 0$, $\mu_y = 1$, $\rho = 0.99999$.
Integrale di riferimento: 0.999999999873787.

	MySoft.(a)	MySoft.(b)	Integ. su $\partial\Omega$	polygauss
Errore ass.	2.5E-14	2.8E-13	1.9E-13	4.7E-03(n=350)
Tempo (sec.)	(0)+0.016	(0)+0.062	0.031	2.17

	dblquad	twoD
Errore ass.	9.9E-01	9.9E-01
Tempo (sec.)	0.047	0.031

Tabella 4.7: Poligono con 37 lati, risultati esempio 2.3.



(a) *'VertexHorizontal'*. (b) *'VertexNotHorizontal'*.

Figura 4.7: Poligono con 37 lati, esempio 2.3.

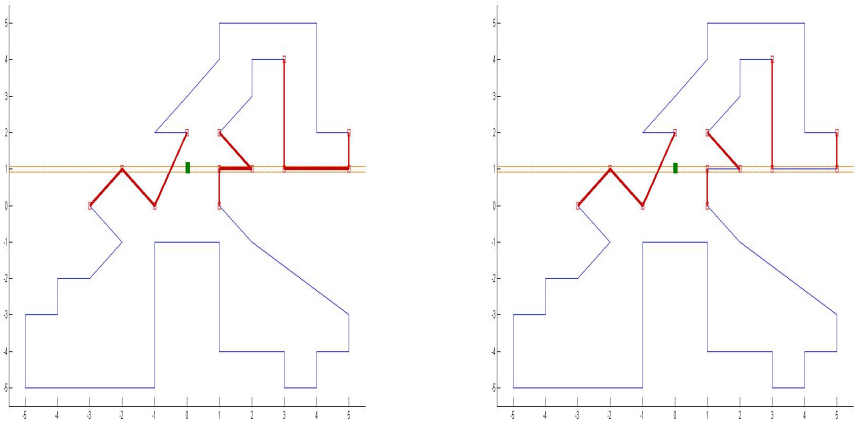
2.4 Parametri della gaussiana bivariata:
$\sigma_x^2 = 0.2$, $\sigma_y^2 = 0.0001$, $\mu_x = 0$, $\mu_y = 1$, $\rho = 0.9988$.
Integrale di riferimento: 0.870898459019559.

2.5 Parametri della gaussiana bivariata:
$\sigma_x^2 = 9$, $\sigma_y^2 = 0.0009$, $\mu_x = -4$, $\mu_y = 3$, $\rho = 0.99999$.
Integrale di riferimento: 0.072256948682601.

	MySoft.(a)	MySoft.(b)	Integ. su $\partial\Omega$	polygauss
Errore ass.	2.0E-15	1.2E-15	1.4E-15	4.7E-02(n=350)
Tempo (sec.)	(0)+0.17	(0)+0.20	0.55	2.28

	dblquad	twoD
Errore ass.	8.7E-01	*
Tempo (sec.)	0.031	*

Tabella 4.8: Poligono con 37 lati, risultati esempio 2.4.



(a) *'VertexHorizontal'*. (b) *'VertexNotHorizontal'*.

Figura 4.8: Poligono con 37 lati, esempio 2.4.

	MySoft.(a)	MySoft.(b)	Integ. su $\partial\Omega$	polygauss
Errore ass.	3.8E-16	3.8E-16	1.9E-16	1.3E-02(n=350)
Tempo (sec.)	(0)+0.92	(0)+0.92	3.25	3.24

	dblquad	twoD
Errore ass.	7.2E-02	7.2E-02
Tempo (sec.)	0.06	0.13

Tabella 4.9: Poligono con 37 lati, risultati esempio 2.5.

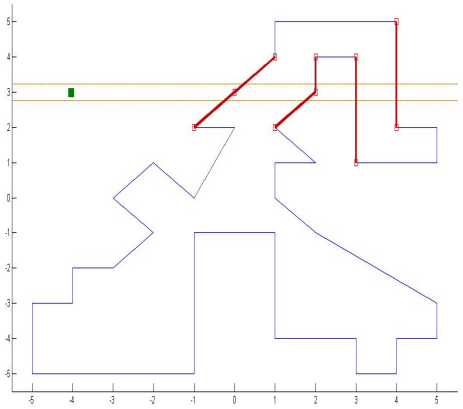


Figura 4.9: Poligono con 37 lati, esempio 2.5,
(a)'VertexHorizontal' e (b)'VertexNotHorizontal'.

2.6 Parametri della gaussiana bivariata:
$\sigma_x^2 = 2$, $\sigma_y^2 = 0.0008$, $\mu_x = 1$, $\mu_y = -3$, $\rho = 0.9999$.
Integrale di riferimento: 0.575622712102470 (`Warning`).

	MySoft.(a)	MySoft.(b)	Integ. su $\partial\Omega$	polygauss
Errore ass.	7.8E-16	1.6E-15	0.0E+00	2.5E-02(n=350)
Tempo (sec.)	(0)+0.22	(0)+0.23	1.28	2.28

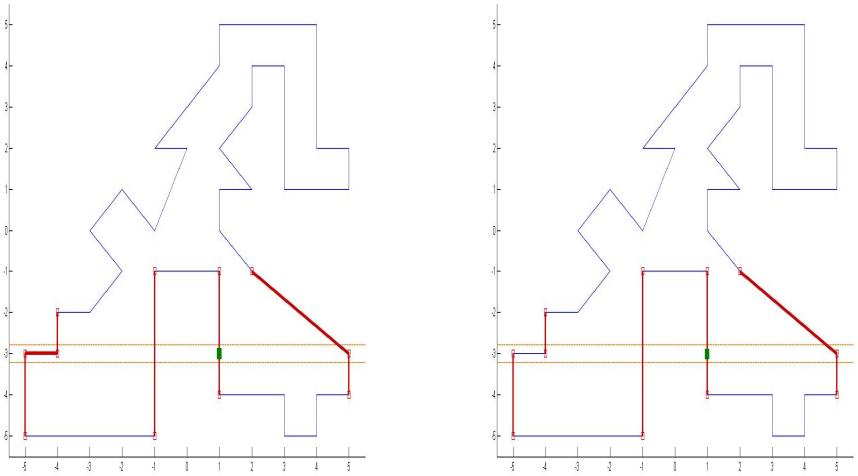
	dblquad	twoD
Errore ass.	3.9E-01	5.7E-01
Tempo (sec.)	39	0.031

Tabella 4.10: Poligono con 37 lati, risultati esempio 2.6.

I risultati contenuti in questi esempi sono per la maggior parte in accordo con quanto trovato nel primo test e rafforzano quindi le ipotesi fatte.

Si trova innanzitutto che le performance di 'MySoftware' (cap. 5) risultano di nuovo le migliori tra quelle dei quattro algoritmi: il valore dell'integrale viene sempre calcolato con un errore minore di `1E-10`. Si deve notare però che anche questo algoritmo incontra delle difficoltà in alcuni esempi, dove il tempo di esecuzione di alza e l'errore assoluto diventa di poco più grande (rimanendo però sempre minore di *tol*) .

Per il 'Metodo delle catene' i tempi di esecuzione delle due funzioni 'VertexHorizontal' e 'VertezNotHorizontal' sono approssimati automaticamente a zero (sono cioè inferiori alla sensibilità del CPU timer).



(a) *'VertexHorizontal'.* (b) *'VertexNotHorizontal'.*

Figura 4.10: Poligono con 37 lati, esempio 2.6.

Tale poligono contiene meno lati orizzontali rispetto al precedente (poco più di un terzo del totale) e negli esempi riportati il numero di catene che li escludono (calcolate con 'VertexNotHorizontal') quasi sempre supera solo di due quello delle catene che li comprendono (determinate con 'VertexHorizontal'). Due iterazioni in più nel ciclo finalizzato all'integrazione di $F(x, y)$ in (2.3), ossia due chiamate in più di 'quadgk' (cfr. [4]), non aumentano di molto il tempo di esecuzione del metodo. In quattro esempi su sei si trova infatti che il costo computazionale del 'Metodo delle catene' nel caso (a) è vicino o addirittura uguale a quello nel caso (b). Nei rimanenti, gli esempi 2.3 e 2.6, il (b) torna invece ad essere il caso peggiore. Questo si nota in particolare nel 2.3, che è l'unico esempio in cui il numero di catene senza lati orizzontali supera di cinque il numero di quelle che tali lati li includono.

In tutti gli esempi l'integrazione sull'intero bordo del poligono è meno efficiente del 'Metodo delle catene': l'errore assoluto o rimane dello stesso ordine o aumenta, mentre i tempi si allungano. La differenza maggiore si registra nel 2.5, dove occorrono ben 3.25 secondi perchè $F(x, y)$ in (2.3) sia integrata su tutto $\partial\Omega$ con un errore di circa `1E-16`, contro i 92 decimi di secondo necessari al 'Metodo delle catene'. Nel 2.3 invece l'integrazione sul bordo riporta un errore dell'ordine di `1E-13` in 0.031 secondi: una performance migliore di quella dell'altro metodo nel caso (b), però peggiore di quella nel caso (a).

Da una parte la diminuzione del numero totale di lati dovrebbe causare un miglioramento nei risultati che si ottengono con l'integrazione sull'intero

bordo. Dall'altra però vi sono due fattori che favoriscono in maniera molto più consistente il 'Metodo delle catene': tranne nell'esempio 2.3, il numero di lati nella striscia è al massimo pari ad un quarto del totale, quindi le catene sono poco numerose e *atol*, la tolleranza passata a 'quadgk' (cfr. [4]) in ogni iterazione del ciclo, è circa `1E-11` (diminuisce di un solo ordine di grandezza rispetto a *tol*). Nel 2.3, nonostante i lati compresi nelle catene siano più della metà, si trova comunque che l'integrazione su tutto $\partial\Omega$ è più costosa del 'Metodo delle catene' nel caso (a).

Nell'esempio 2.6 l'errore riportato integrando sull'intero bordo risulta nullo: ciò significa che il valore dell'integrale calcolato da 'quadgk' con tolleranza `1E-10`, è lo stesso di quello che viene preso come riferimento. Il messaggio di attenzione (`Warning`) restituito in output, segnala però che quest'ultimo valore differisce dall'integrale esatto, di un errore assoluto maggiore della tolleranza richiesta (`1E-14`). Allora i risultati di tutti i quattro i codici sono in realtà fittizi, in quanto fanno riferimento a un dato a sua volta non affidabile. In particolare quello di 'MySoftware', ottenuto con l'integrazione sull'intero bordo, risulta erroneamente essere quello più preciso, dato che nel fare questo si utilizza lo stesso metodo di quello usato appunto per avere l'integrale di riferimento: si invoca una sola volta 'quadgk' su tutta la sequenza di lati del poligono, con $tol =$ `1E-10` invece di `1E-14`.

Proseguendo si percepisce come, anche 'MySoftware', seppur restando l'algoritmo più efficiente, raggiunga dei risultati meno soddisfacenti con particolari scelte dei parametri della gaussiana bivariata. Passando dall'esempio 2.1 al 2.2 si sono tenute fisse le medie marginali $(\mu_x, \mu_y) = (0, 1)$, punto interno al poligono, si è presa la varianza marginale in y molto più piccola, $\sigma_y^2 = 0.00001$, e il coefficiente di correlazione $\rho = 0$. Per tutti e tre i metodi aumentano i tempi di esecuzione: con l'integrazione su tutto $\partial\Omega$, in particolare, occorrono quasi due decimi di secondo. Un simile peggioramento si verifica anche nel 2.3 mantenendo le stesse medie marginali, prendendo però al contrario $\sigma_y^2 = 0.1$ più grande di prima, e $\rho = 0.99999$ prossimo a uno. In questo caso i tempi restano dell'ordine di centesimi di secondo, ma l'errore assoluto sale anche a circa `1E-13`. Nell'esempio 2.4 si sceglie di nuovo $(\mu_x, \mu_y) = (0, 1)$, $\sigma_y^2 = 0.0001$ vicino a zero e $\rho = 0.9988$: qui l'errore assoluto ha ordine `1E-15`, ma i tempi di esecuzione variano all'incirca da due a cinque decimi di secondo. Infine anche negli ultimi due esempi, 2.5 e 2.6, il livello delle performance di 'MySoftware' si abbassa leggermente: il punto che rappresenta le medie marginali viene posto fuori dal poligono nel 2.5, sul bordo nel 2.6, mentre la varianza marginale in y rimane molto piccola e il coefficiente di correlazione vicino ad uno.

Da quanto detto finora, sembra che il primo codice incontri le difficoltà maggiori sia quando ρ è prossimo a uno, sia quando σ_y^2 è molto vicino a zero, sia se tali scelte dei parametri vengono fatte assieme.

'polygauss' [11] calcola il valore dell'integrale con un errore assoluto minore della tolleranza richiesta solo in un esempio, mentre nei rimanenti l'er-

rore, a grado 350, è ancora dell'ordine di `1E-03` o anche superiore. Queste difficoltà sono dovute, con molta probabilità, alla scelta di σ_y^2 prossimo a zero, di ρ molto vicino ad uno, oppure delle due messe assieme. In particolare la performance migliore si trova nell'esempio 2.1, dove il tempo necessario per avere un errore di `8.0E-11` è di tre centesimi di secondo e grado 38. L'esempio 2.5 è invece il peggiore ($\sigma_y^2 = 0.0009$, $\rho = 0.99999$): il tempo di esecuzione supera i 3 secondi e l'errore è dell'ordine di `1E-02`.

'dblquad' (cfr. [4]) non riesce mai a restituire il risultato con un errore minore della tollerenza richiesta. La performance migliore è riportata nell'esempio 2.1 ($\sigma_y^2 = 0.01$, $\rho = 0.5$): l'errore è `6.9E-04` e il tempo di esecuzione 15.8 secondi. I risultati peggiorano drasticamente invece negli esempi dove la varianza marginale in y tende a zero e il coefficiente di correlazione ad uno.

Si ha poi conferma del fatto che, con la tolleranza scelta (tol = `1E-10`), 'twoD' [9] o calcola l'integrale con un errore dell'ordine di `1E-01`, `1E-02`, oppure addirittura non riesce nemmeno a darlo in output. Come prima si è provato ad analizzare meglio la questione. I dati riportati nelle tabelle 4.11, 4.12, 4.13 sono relativi ai risultati dei due algoritmi con tolleranza `1E-05`.

	dblquad	twoD
Errore ass.	1.5E-04	1.7E-06
Tempo (sec.)	0.452	0.062

Tabella 4.11: Poligono con 37 lati, esempio 2.1, risultati di 'dblquad' e 'twoD' con tol = `1E-05`.

	dblquad	twoD
Errore ass.	4.0E-01	4.0E-01
Tempo (sec.)	0.031	0.016

Tabella 4.12: Poligono con 37 lati, esempio 2.2, risultati di 'dblquad' e 'twoD' con tol = `1E-05`.

	dblquad	twoD
Errore ass.	7.2E-02	7.2E-02
Tempo (sec.)	0.047	0.016

Tabella 4.13: Poligono con 37 lati, esempio 2.5, risultati di 'dblquad' e 'twoD' con tol = `1E-05`.

Le cose non cambiano quando ρ è prossimo a uno o σ_y^2 tende a zero: negli esempi 2.2 e 2.5 entrambi i risultati di 'dblquad' (cfr. [4]) e twoD' [9] restano negativi e sembrano allora non essere legati alla tolleranza scelta. Nel 2.1 invece ($\sigma_y^2 = 0.01$, $\rho = 0.5$) 'twoD' ottiene, in circa sei centesimi di secondo, un errore di `1.7E-06`. Nonostante quest'ultimo rimanga comunque superiore alla tolleranza richiesta, bisogna notare che tale performance è migliore anche di quella di 'dblquad' con *tol* = `1E-10`. Si rafforza quindi l'ipotesi che 'twoD' renda di più con tolleranze dell'ordine di `1E-04`, `1E-05`.

4.3 Poligono con 18 lati

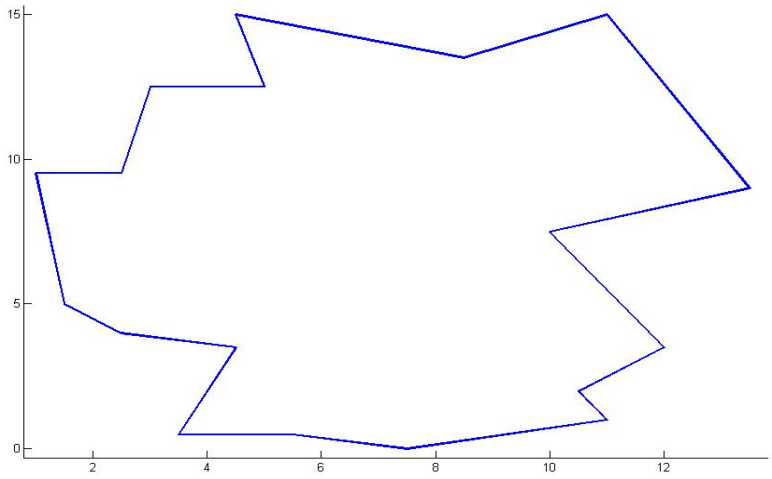


Figura 4.11: Poligono con 18 lati.

Per il terzo poligono vengono illustrati i 6 esempi seguenti:

3.1 Parametri della gaussiana bivariata:
$\sigma_x^2 = 4$, $\sigma_y^2 = 1$, $\mu_x = 7$, $\mu_y = 8$, $\rho = 0$.
Integrale di riferimento: 0.972452862221792.

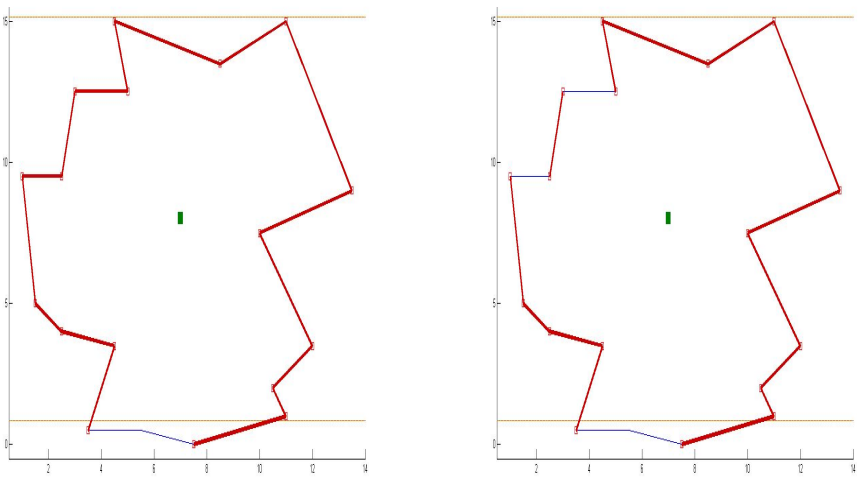
3.2 Parametri della gaussiana bivariata:
$\sigma_x^2 = 2$, $\sigma_y^2 = 0.00001$, $\mu_x = 2.8$, $\mu_y = 0.5$, $\rho = 0.99999$.
Integrale di riferimento: 0.310212925790640 (`Warning`).

3.3 Parametri della gaussiana bivariata:
$\sigma_x^2 = 4$, $\sigma_y^2 = 1$, $\mu_x = 3$, $\mu_y = 12.5$, $\rho = 0$.
Integrale di riferimento: 0.347235085861977.

	MySoft.(a)	MySoft.(b)	Integ. su $\partial\Omega$	polygauss
Errore ass.	5.7E-15	5.9E-15	3.3E-16	4.5E-11(n=15)
Tempo (sec.)	(0)+0.016	(0)+0.047	0.062	0.031

	dblquad	twoD
Errore ass.	9.7E-01	*
Tempo (sec.)	0.62	*

Tabella 4.14: Poligono con 18 lati, risultati esempio 3.1.



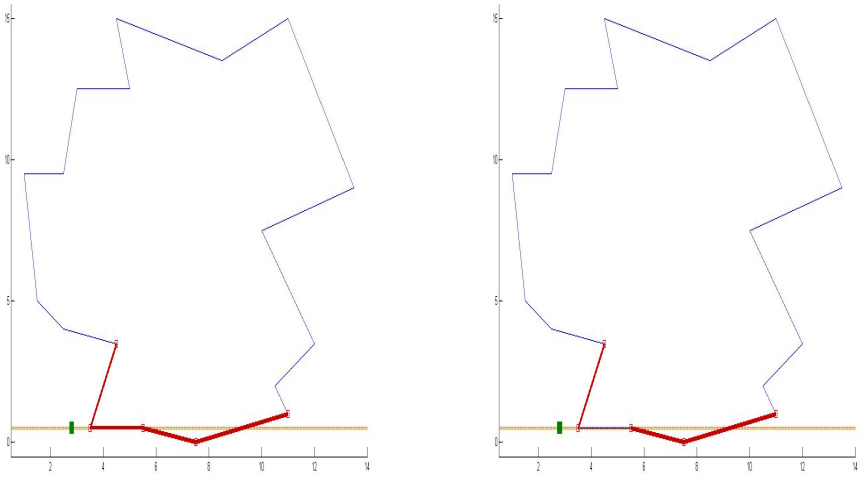
(a) *'VertexHorizontal'*. (b) *'VertexNotHorizontal'*.

Figura 4.12: Poligono con 18 lati, esempio 3.1.

	MySoft.(a)	MySoft.(b)	Integ. su $\partial\Omega$	polygauss
Errore ass.	0.3E+00	3.6E-05	1.1E-16	1.9E-01(n=350)
Tempo (sec.)	(0)+0.047	(0)+1.40	1.45	1.51

	dblquad	twoD
Errore ass.	3.1E-01	3.1E-01
Tempo (sec.)	0.031	0.031

Tabella 4.15: Poligono con 18 lati, risultati esempio 3.2.



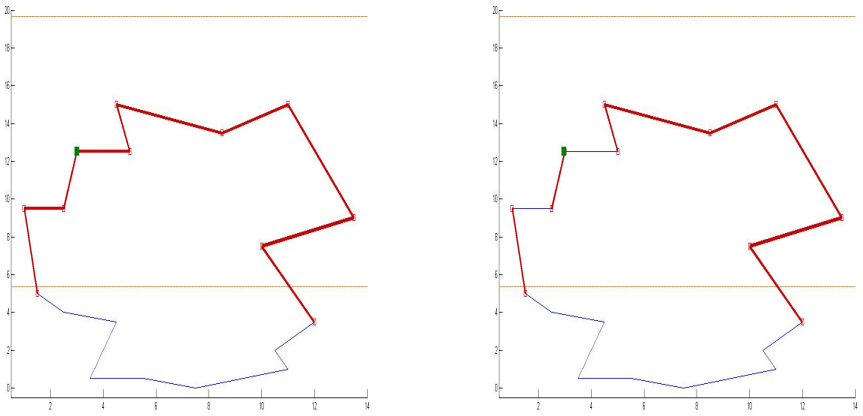
(a) *'VertexHorizontal'.* (b) *'VertexNotHorizontal'.*

Figura 4.13: Poligono con 18 lati, esempio 3.2.

	MySoft.(a)	MySoft.(b)	Integ. su $\partial\Omega$	polygauss
Errore ass.	8.2E-15	8.0E-15	1.1E-16	2.2E-12(n=15)
Tempo (sec.)	(0)+0.016	(0)+0.047	0.062	0

	dblquad	twoD
Errore ass.	3.5E-01	*
Tempo (sec.)	0.031	*

Tabella 4.16: Poligono con 18 lati, risultati esempio 3.3.



(a) *'VertexHorizontal'.* (b) *'VertexNotHorizontal'.*

Figura 4.14: Poligono con 18 lati, esempio 3.3.

3.4 Parametri della gaussiana bivariata:
$\sigma_x^2 = 6$, $\sigma_y^2 = 0.00001$, $\mu_x = 12$, $\mu_y = 11$, $\rho = 0.01$.
Integrale di riferimento: 0.607172462364816.

	MySoft.(a)	MySoft.(b)	Integ. su $\partial\Omega$	polygauss
Errore ass.	2.9E-14	2.9E-14	4.4E-16	1.2E+00(n=350)
Tempo (sec.)	(0)+0.078	(0)+0.078	0.16	1.39

	dblquad	twoD
Errore ass.	6.1E-01	6.1E-01
Tempo (sec.)	0.047	0.031

Tabella 4.17: Poligono con 18 lati, risultati esempio 3.4.

3.5 Parametri della gaussiana bivariata:
$\sigma_x^2 = 6$, $\sigma_y^2 = 0.001$, $\mu_x = 11.5$, $\mu_y = 3$, $\rho = 0.9999$.
Integrale di riferimento: 0.498350567600161.

3.6 Parametri della gaussiana bivariata:
$\sigma_x^2 = 10$, $\sigma_y^2 = 0.0002$, $\mu_x = 13$, $\mu_y = 9$, $\rho = 0.9999$.
Integrale di riferimento: 0.562620703710175 (`Warning`).

Questi ultimi risultati sono significativi e aiutano a trarre le conclusioni del confronto.

Più in generale si ha conferma del fatto che, anche il primo algoritmo incontra difficoltà se il coefficiente di correlazione è prossimo a uno oppure

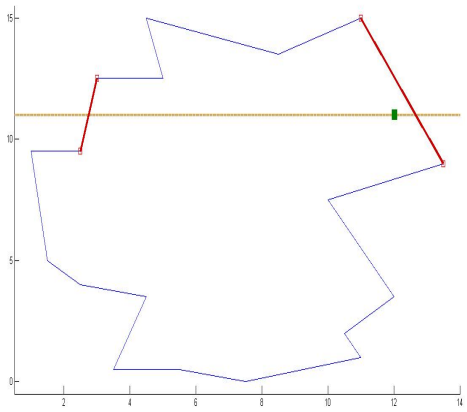


Figura 4.15: Poligono con 18 lati, esempio 3.4,
(a)'VertexHorizontal' e (b)'VertexNotHorizontal'.

	MySoft.(a)	MySoft.(b)	Integ. su $\partial\Omega$	polygauss
Errore ass.	1.2E-15	1.2E-15	1.1E-16	8.5E-03(n=350)
Tempo (sec.)	(0)+0.062	(0)+0.062	0.45	1.42

	dblquad	twoD
Errore ass.	5.0E-01	5.0E-01
Tempo (sec.)	0.031	0

Tabella 4.18: Poligono con 18 lati, risultati esempio 3.5.

	MySoft.(a)	MySoft.(b)	Integ. su $\partial\Omega$	polygauss
Errore ass.	1.1E-14	1.1E-14	4.4E-16	1.8E-03(n=350)
Tempo (sec.)	(0)+0.30	(0)+0.30	1.37	1.48

	dblquad	twoD
Errore ass.	5.6E-01	5.6E-01
Tempo (sec.)	0.047	0.031

Tabella 4.19: Poligono con 18 lati, risultati esempio 3.6.

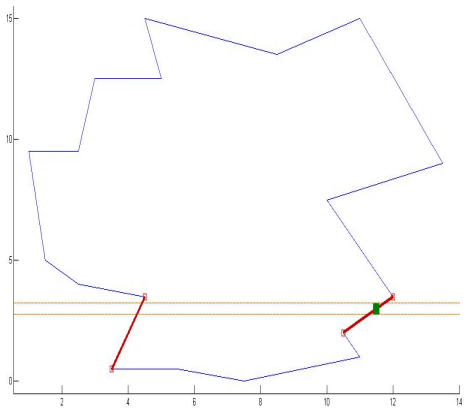


Figura 4.16: Poligono con 18 lati, esempio 3.5,
(a)‘VertexHorizontal’ e (b)‘VertexNotHorizontal’.

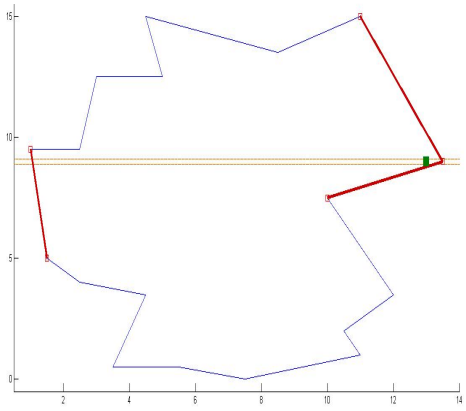


Figura 4.17: Poligono con 18 lati, esempio 3.6,
(a)‘VertexHorizontal’ e (b)‘VertexNotHorizontal’.

se si prende la varianza marginale in y molto piccola. Nonostante questo i risultati che restituitsce hanno sempre un errore minore della tolleranza richiesta.

Nell'esempio 3.1 ($\sigma_y^2 = 1$, $\rho = 0$) la performance di 'MySoftware' (cap. 5) è buona: in qualche centesimo di secondo l'errore risulta dell'ordine di `1E-15`, per il 'Metodo delle catene', `1E-16` con l'integrazione sull'intero bordo. Passando al 3.3 si mantengono gli stessi valori per la varianza marginale in y e per il coefficiente di correlazione, mentre il punto costituito dalle medie marginali, invece di essere all'interno del poligono, coincide con un vertice. I risultati del primo algoritmo che ne derivano sono soddisfacenti, allo stesso livello dei precedenti. Passando all'esempio 3.4 la performance di 'MySoftware' peggiora: il 'Metodo delle catene' sia in (a) che in (b) riporta un errore dell'ordine di `1E-14` in 0.078 secondi, mentre per l'integrazione su tutto $\partial\Omega$ ne occorrono 0.16 per un errore di `4.4E-16`. In tal caso le medie marginali sono $(\mu_x, \mu_y) = (12, 11)$, all'interno del poligono, $\sigma_y^2 = 0.00001$, $\rho = 0.01$. Nel 3.5 si fa crescere il coefficiente di correlazione, prossimo a uno, si prende invece la varianza marginale in y un pò più grande e il punto delle medie marginali sul bordo del poligono. Il 'Metodo delle catene' peggiora in maniera simile a quanto accadeva per il 3.4: il risultato riporta un errore dell'ordine di `1E-15`, con un tempo di esecuzione di circa sei centesimi di secondo. Ancora meno buona è invece la performance con l'integrazione sull'intero bordo, per la quale servono circa cinque decimi di secondo, per avere un errore di circa `1E-16`.

I costi computazionali crescono in maniera più notevole nell'esempio 3.6: il punto delle medie marginali viene collocato all'interno del poligono in prossimità del bordo, σ_y^2 si avvicina molto a zero e il coefficiente di correlazione rimane pari a 0.9999. Per il 'Metodo delle catene' l'errore assoluto diventa dell'ordine di `1E-14` e i tempi di esecuzione si alzano (tre decimi di secondo). Con l'integrazione su tutto $\partial\Omega$ l'errore rimane circa `1E-16` e occorre più di un secondo. Infine la performance peggiore viene realizzata nel 3.2 con $\rho = 0.99999$, $(\mu_x, \mu_y) = (2.8, 0.5)$ appena fuori da $\partial\Omega$ e $\sigma_y^2 = 0.00001$. Qui il 'Metodo delle catene' con 'VertexHorizontal' sbaglia completamente il calcolo dell'integrale (approssimandolo a zero), mentre nel caso (b) viene restituito con un errore dell'ordine di `1E-05` in più di un secondo. Con l'integrazione sull'intero bordo, come nel 3.6, si compie un errore di circa `1E-16` in poco più di un secondo. L'andamento dei risultati in questi ultimi due esempi non rispecchia però la realtà: infatti per entrambi viene visualizzato il messaggio di attenzione (`Warning`) da parte di 'quadgk' (cfr. [4]), la quale non riesce a fornire un valore di riferimento per l'integrale con un errore assoluto minore della tolleranza richiesta (`1E-14`). Quanto detto per il precedente poligono resta vero anche qui: il metodo di integrazione della x-primitiva sull'intero bordo ne risulta avantaggiato (riporta comunque un errore di `1E-16`), quando invece dovrebbe avere performance peggiori, come avviene per il 'Metodo delle catene', in conseguenza alla particolare scelta

dei parametri della $f(x, y)$ in (1.1).

Ora tornando al discorso iniziale, si vuole risalire alle cause del peggioramento delle performance di 'MySoftware', in corrispondenza a specifiche scelte dei parametri della gaussiana bivariata. Si è visto che ha dei risultati meno soddisfacenti quando innanzitutto σ_y^2 tende a zero. Questo avviene poichè la gaussiana univariata, che è un fattore della x-primitiva $F(x, y)$ in (2.3), quando la sua varianza (σ_y^2) è molto piccola, si concentra vicino alla media μ_y. La funzione risulta quindi crescere, prima, e decrescere, dopo, molto rapidamente in prossimità della media e diventa quindi difficile da integrare.

D'altra parte 'MySoftware' incontra simili difficoltà anche quando il coefficiente di correlazione, ρ, è prossimo a uno: i costi computazionali aumentano sensibilmente e l'errore assoluto può peggiorare (pur restando quest'ultimo sotto la tolleranza richiesta). Questo si spiega analizzando il comportamento della erf, presente nell'espressione della x-primitiva $F(x, y)$ in (2.3), calcolata con l'integratore simbolico [16]. Quando ρ è vicino ad uno, il denominatore dell'argomento della 'funzione errore' tende a zero. Quindi se il numeratore è positivo l'argomento tende a $+\infty$ e la erf ad 1, mentre se è negativo l'argomento tende a $-\infty$ e la erf a -1 (bisogna ricordare infatti che è una funzione dispari). Cioè in un caso del genere la 'funzione errore' è molto simile alla funzione segno: essa vale circa 1 per tutte le coppie di punti nel piano xy tali che il numeratore sia positivo, circa -1 per tutte le rimanenti. Osservando in (2.3) l'espressione del numeratore, si capisce allora che la erf vale circa 1 per tutti gli (x, y) tali che $\sigma_x\rho(\mu_y - y) + \sigma_y(x - \mu_x) > 0$, vale circa -1 per gli (x, y) tali che $\sigma_x\rho(\mu_y - y) + \sigma_y(x - \mu_x) < 0$. Di conseguenza $F(x, y)$ ha una 'discontinuità' sulla retta

$$\sigma_x\rho(\mu_y - y) + \sigma_y(x - \mu_x) = 0. \tag{4.1}$$

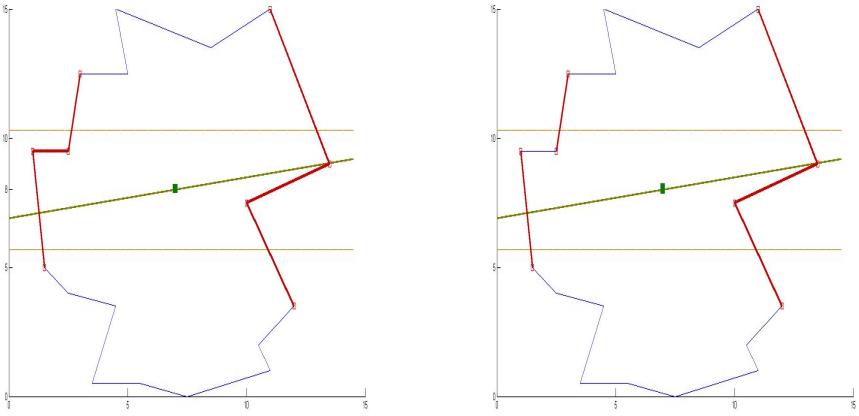
Per l'algoritmo diventa difficile integrare F in un suo punto di 'discontinuità'. Questo inconveniente, legato alla scelta di ρ vicino ad uno, si presenta per il 'Metodo delle catene' quando i punti di 'discontinuità' della x-primitiva si trovano sulle sequenze di lati nella striscia (dove tale funzione viene integrata), cioè quando le intersezioni della retta in (4.1) con il bordo del poligono sono contenute nelle catene. Questo è quello che avviene nel seguente esempio, il 3.7, dove i parametri scelti per la gaussiana bivariata sono: $\sigma_x^2 = 4$, $\sigma_y^2 = 0.1$, $\mu_x = 7$, $\mu_y = 8$, $\rho = 0.999999$. Mentre l'integrale di riferimento è 0.997335444614285.

Si osserva che qui il 'Metodo delle catene' ha dei costi computazionali alti rispetto alle sue performance migliori: il risultato viene restituito con un errore dell'ordine di `1E-16` in tre decimi di secondo.

Nell'esempio 3.8 si scelgono gli stessi parametri, a parte la varianza marginale in x, $\sigma_x^2 = 0.001$, presa più piccola per fare in modo che il coefficiente angolare della retta in (4.1) sia maggiore di prima e quindi la retta,

	MySoft.(a)	MySoft.(b)	Integ. su $\partial\Omega$
Errore ass.	3.3E-16	1.1E-16	1.1E-16
Tempo (sec.)	(0)+0.31	(0)+0.31	0.48

Tabella 4.20: Poligono con 18 lati, risultati esempio 3.7.



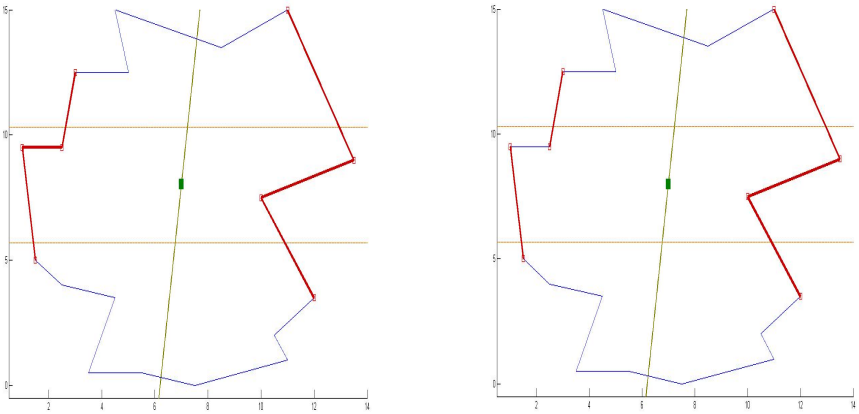
(a) *'VertexHorizontal'*. (b) *'VertexNotHorizontal'*.

Figura 4.18: Poligono con 18 lati, esempio 3.7.

essendo più inclinata, non intersechi il bordo del poligono in corrispondenza alle catene. L'integrale di riferimento è 1.

	MySoft.(a)	MySoft.(b)	Integ. su $\partial\Omega$
Errore ass.	4.4E-16	2.2E-16	0.0E+00
Tempo (sec.)	(0)+0.016	(0)+0.047	0.031

Tabella 4.21: Poligono con 18 lati, risultati esempio 3.8.



(a) *'VertexHorizontal'*. (b) *'VertexNotHorizontal'*.

Figura 4.19: Poligono con 18 lati, esempio 3.8.

Si verifica che in questo caso la performance del 'Metodo delle catene' è molto buona, nonostante il coefficiente di correlazione sia prossimo ad uno: infatti $F(x, y)$ viene integrata su sequenze di lati che non contengono suoi punti di 'discontinuità'.
Per quanto riguarda il metodo di integrazione su tutto il bordo, ci si aspetterebbe che esso riportasse delle performance non buone in entrambi gli esempi 3.7 e 3.8. Infatti la retta in (4.1) passa per il punto (μ_x, μ_y), interno al poligono, e interseca necessariamente il bordo in due punti. Con questo metodo cioè la x-primitiva viene per forza integrata anche su due suoi punti di 'discontinuità' e quindi il costo computazionale dovrebbe essere sempre elevato. In realtà questo non avviene nell'esempio 3.8, poichè la retta interseca il bordo del poligono fuori dalla striscia e di conseguenza i punti di 'discontinuità' di $F(x, y)$ presenti sul bordo, si trovano in una zona dove la funzione in modulo è piccola e quindi anche il suo 'salto' (in corrispondenza del punto di discontinuità) lo è.
Infine si capisce che quando ρ è vicino ad uno e il punto costituito dalle

medie marginali di $f(x, y)$ si trova sul bordo del poligono, le performance di 'MySoftware' non possono essere le migliori. In tal caso infatti la retta in (4.1) interseca il bordo in (μ_x, μ_y) e quindi in entrambi i metodi la x-primitiva viene integrata dove essa ha un grande 'salto', dato che lì è 'discontinua' e assume il valore massimo in modulo.

Proseguendo nell'analisi dei risultati, il 'Metodo delle catene' è più efficiente del metodo di integrazione su tutto il bordo nell'esempio 3.4 e nel 3.5. In quest'ultimo integrando su $\partial\Omega$ l'errore si mantiene di un ordine più piccolo `1E-16`, ma d'altra parte il tempo di esecuzione sale a quasi mezzo secondo. Infine nel 3.1 e nel 3.3 i due metodi sono alla pari e ciò si giustifica osservando che il numero di lati contenuti nella striscia costituisce buona parte del totale.

Poi questo poligono contiene solamente tre lati orizzontali su diciotto, così, analogamente al secondo test, si osserva che il numero di catene che escludono i lati orizzontali, supera al massimo di due quello delle catene che li includono. Di conseguenza i costi computazionali dei casi (a) e (b) del 'Metodo delle catene' sono uguali per buona parte degli esempi (tra l'altro notando che, ancora una volta, le due funzioni per la determinazione delle catene hanno costo nullo in termini di tempi).

Per 'polygauss' [11] la diminuzione del numero complessivo di lati del poligono è sicuramente motivo di miglioramento delle sue performance. In particolare quando il coefficiente di correlazione è pari oppure prossimo a zero, come negli esempi 3.1 e 3.3, i suoi tempi di esecuzione sono dell'ordine di centesimi di secondo, o addirittura più bassi, e l'integrale viene calcolato con un errore assoluto dell'ordine rispettivamente di `1E-11`, `1E-12` e grado 15. Invece un risultato molto negativo viene conseguito da questo codice nell'esempio 3.4, dove $\sigma_y^2 = 0.00001$.

'dblquad' (cfr. [4]) e 'twoD' [9] rimangono i due algoritmi dai risultati più scarsi: entrambi non riescono mai a calcolare il valore dell'integrale con un errore assoluto minore della tolleranza richiesta. Dalle tabelle 4.22, 4.23 e 4.24 si nota che cambiando la tolleranza i due algoritmi si comportano come nei test precedenti.

	dblquad	twoD
Errore ass.	9.7E-01	8.0E-07
Tempo (sec.)	0.047	1.20

Tabella 4.22: Poligono con 18 lati, esempio 3.1, risultati di 'dblquad' e 'twoD' con $tol =$ `1E-05`.

Nell'esempio 3.5 ($\sigma_y^2 = 0.001$, $\rho = 0.9999$) non si trova nessun miglioramento dei risultati: entrambi commettono un errore assoluto enorme nel calcolare l'integrale. Nel 3.1, richiedendo tolleranza `1E-05`, la performance

	dblquad	twoD
Errore ass.	3.5E-01	4.0E-06
Tempo (sec.)	0.062	3.31

Tabella 4.23: Poligono con 18 lati, esempio 3.3, risultati di 'dblquad' e 'twoD' con $tol =$ `1E-04`.

	dblquad	twoD
Errore ass.	5.0E-01	5.0E-01
Tempo (sec.)	0.031	0.016

Tabella 4.24: Poligono con 18 lati, esempio 3.5, risultati di 'dblquad' e 'twoD' con $tol =$ `1E-04`.

di 'dblquad' non migliora rispetto a quella con $tol =$ `1E-10`; 'twoD' invece restituisce il risultato con un errore assoluto dell'ordine di `1E-07` in poco più di un secondo. Nel 3.3 'twoD' risulta di nuovo il migliore imponendo una tolleranza di `1E-04`.

4.4 Conclusioni

Da quanto analizzato finora si possono trarre delle conclusioni, riassumere problematiche e vantaggi di ciascun codice, per giudicare infine quale sia il più efficiente.

'MySoftware' (cap. 5) riesce sempre a calcolare il valore dell'integrale con un errore assoluto inferiore, anche di molto, alla tolleranza richiesta (`1E-10`), in tempi in generale piccoli (centesimi o al più decimi di secondo). Questo algoritmo ha due punti di forza: grazie al teorema di Green (cfr. [1]) affronta l'integrazione sul poligono come un problema in una dimensione e poi utilizza la funzione 'quadgk' (cfr. [4]), dai costi computazionali in generale bassi.

Il 'Metodo delle catene' sfrutta la proprietà di parità e di decadimento di una gaussiana in una variabile. La funzione 'VertexNotHorizontal' è la meno conveniente per tale metodo: il motivo non è il suo tempo di esecuzione (prossimo allo zero come per 'VertexHorizontal'), ma il fatto che essa determina le catene di lati escludendo quelli orizzontali, le quali, come avviene nel primo test, possono essere molte di più delle catene che li comprendono. Al crescere della numerosità di tali successioni, cresce anche il numero di iterazioni del ciclo finalizzato all'integrazione e quindi il suo costo computazionale. Man mano che il numero di lati orizzontali del poligono diminuisce, il problema si attenua gradualmente, ma allora, allo stesso tempo, descresce anche la necessità di avere a disposizione una funzione come 'VertexNotHorizontal', che tali lati li esclude dalle catene.

Il confronto tra le performance del metodo di integrazione su tutto $\partial\Omega$ e quelle del 'Metodo delle catene', permette di proclamare migliore tra i due il secondo. Nel primo test ci si aspettava che integrare sulle catene fosse il procedimento alla lunga meno conveniente: in realtà questo non si è verificato perchè anche l'altra scelta ha comportato dei problemi, considerato il grande numero di lati del poligono. Per quanto riguarda gli altri due test, nella grande maggioranza degli esempi il 'Metodo delle catene' è il più efficiente tra i due, mentre nei rimanenti le performance sono alla pari.

Lo svantaggio principale di 'MySoftware' consiste nel fatto che, calcolando la x-primitiva con l'integratore simbolico, viene coinvolta la 'funzione d'errore' erf (cfr. [13]). Ogni volta che 'quadgk' valuta l'integranda $F(x, y)$ in (2.3) perde una quantità considerevole di tempo per questa sua componente: si è verificato che nell'esempio 1.2 su 0.062 secondi necessari per l'integrazione su tutto il bordo, 0.031 sono persi per la erf; oppure ancora nel 2.3 occorrono 0.61 secondi per la valutazione della erf su 3.25 secondi totali di esecuzione.

Infine nei test analizzati si è riscontrato che il primo algoritmo ha costi computazionali maggiori e riporta spesso il risultato con un errore assoluto più grande, quando il coefficiente di correlazione tende a uno, oppure quando la varianza marginale in y è vicina a zero, o anche se queste due scelte dei parametri vengono fatte assieme. Nel caso in cui σ_y^2 è prossima a zero, un'idea

per evitare il problema potrebbe essere quella di utilizzare il teorema di Green nell'altro verso, ossia integrando sul bordo la y-primitiva, che sarà composta da un fattore gaussiano di media μ_x e varianza σ_x^2.

Per quanto riguarda 'polygauss' [11], si è visto come le sue performance, pur essendo spesso buone, non raggiungano mai il livello di quelle di 'MySoftware'. Nei casi migliori l'integrale viene calcolato con un errore di `1E-11`, `1E-12` in tempi molto brevi (centesimi di secondo); nei peggiori a grado 350 tale valore risulta completamente sbagliato.

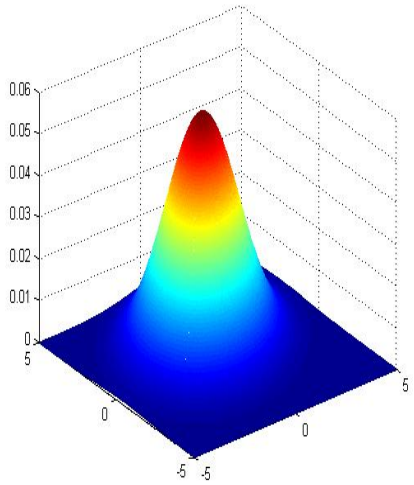
Il punto di forza di 'polygauss' è di nuovo il teorema di Green (cfr. [1]), che permette in un certo senso di spostare il problema dall'ambito della cubatura numerica a quello della quadratura. D'altra parte però le prestazioni di questo algoritmo sono penalizzate in primo luogo dal fatto che l'integranda $f(x, y)$, la gaussiana bivariata, pur essendo una funzione regolare, è molto più complicata di un polinomio. Da quanto risulta nei test i tempi di esecuzione peggiorano, come succede in 'MySoftware', quando innanzitutto il coefficiente di correlazione viene scelto prossimo a uno (oppure a -1). È semplice comprenderne il motivo osservando come la gaussiana bivariata $f(x, y)$ in (1.1) cambia all'aumentare di ρ. Dalla figura 4.20 si capisce infatti come cresca la difficoltà nell'integrare una tale funzione più il coefficiente di correlazione si avvicina ad uno. In questo particolare esempio si sono scelti come parametri $\sigma_x^2 = 3$, $\sigma_y^2 = 3$, $\mu_x = 0$, $\mu_y = 1$.

In secondo luogo anche prendendo la varianza marginale in y, σ_y^2, molto vicina a zero, le performance di 'polygauss' ne risentono in maniera negativa. Questo si spiega guardando nuovamente come si trasforma la $f(x, y)$ in (1.1) al diminuire di questo parametro. La figura 4.21 mostra che tale funzione si comprime sempre più lungo la direzione dell'asse y all'avvicinarsi di σ_y^2 a zero, risultando così sempre più difficile da integrare.

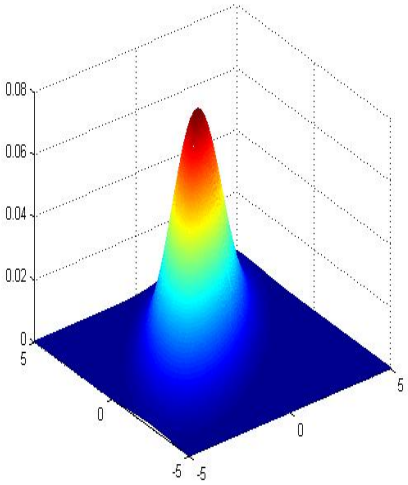
Infine anche la scelta di poligoni formati da un elevato numero di lati, ha causato un peggioramento nei risultati di 'polygauss': dalla formula (2.26) si nota infatti che il numero di punti di cubatura necessari all'algoritmo per approssimare l'integrale aumenta al cresce del numero complessivo di lati di $\partial\Omega$.

Passando infine agli ultimi due codici, i test analizzati mostrano come le loro performance siano nettamente peggiori rispetto a quelle dei primi due. In particolare si è osservato come cambiano i loro risultati al variare della tolleranza imposta. Con tol = `1E-10` 'dblquad' (cfr. [4]) calcola l'integrale con un errore assoluto in generale più piccolo, anche se mai minore di tol, e i tempi di esecuzione sono molto alti. Al contrario 'twoD' [9] ha un costo computazionale minore se la tolleranza è dell'ordine di `1E-04`, `1E-05`: i suoi risultati diventano migliori di quelli dell'altro integratore automatico. Tale algoritmo riesce a garantire un errore assoluto dell'ordine di `1E-06`, `1E-07`, in tempi dell'ordine dei secondi.

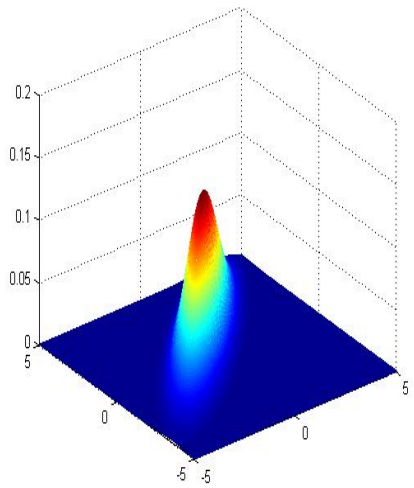
'dblquad' è sicuramente penalizzato dall'utilizzo di 'inpolygon' (cfr. [4]). Lo stesso avviene per 'twoD': in questo caso l'utilizzo dell'opzione 'Singular'



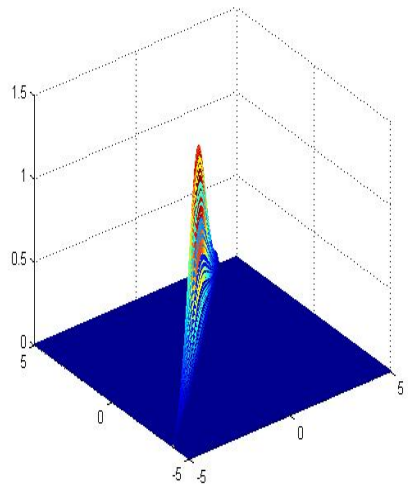
(a) $\rho = 0.3$.



(b) $\rho = 0.7$.

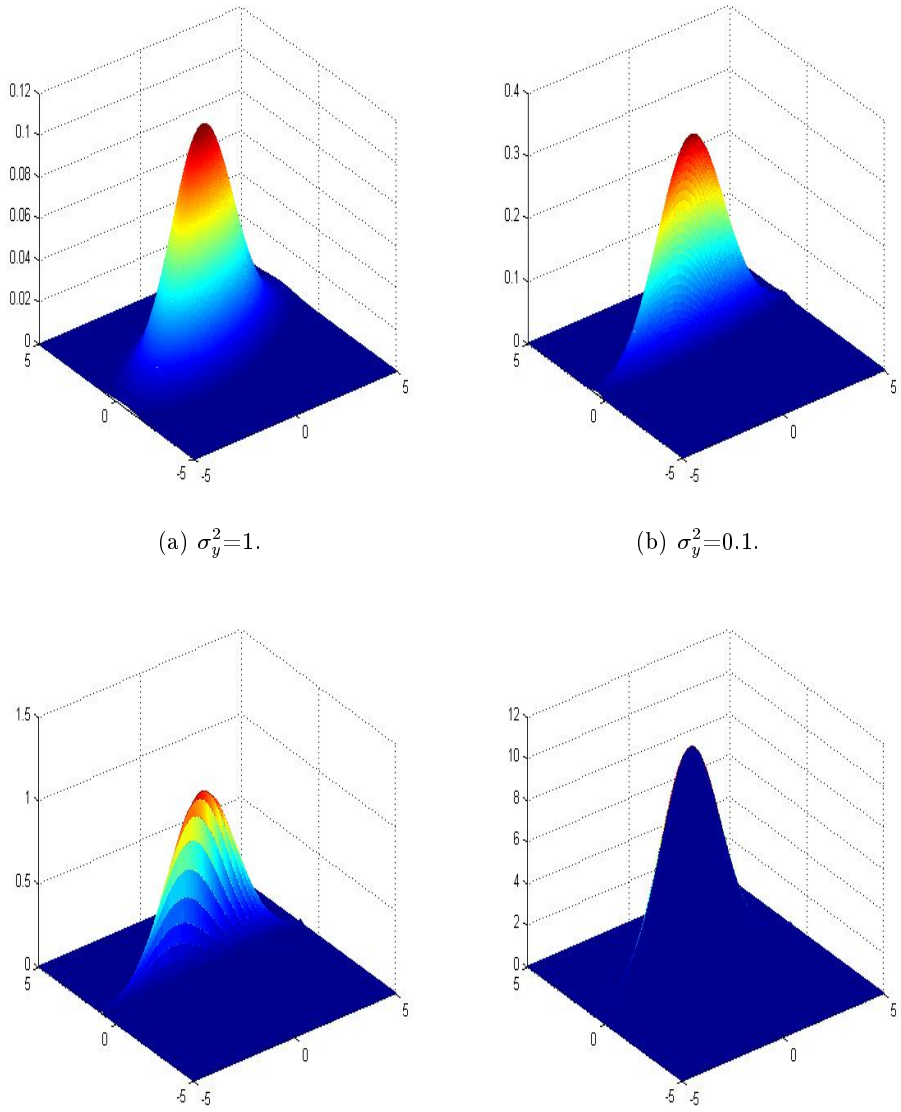


(c) $\rho = 0.9$.



(d) $\rho = 0.999$.

Figura 4.20: Gaussiana Bivariata al variare di ρ.



(a) σ_y^2=1. (b) σ_y^2=0.1.

(c) σ_y^2=0.01. (d) σ_y^2=0.0001.

Figura 4.21: Gaussiana Bivariata al variare di σ_y^2.

attenua il problema, anche se non lo risolve del tutto. Infatti buona parte del tempo di esecuzione di 'twoD' viene occupato per la valutazione di 'inpolygon'. Per citare un caso, nell'esempio 1.1 con tol = `1E-04`, su 4.5 secondi totali di esecuzione ben 1.3 sono persi in tal modo.

Si è notato poi che ulteriori difficoltà per questi due algoritmi si presentano quando il coefficiente di correlazione è prossimo ad uno o la varianza marginale in y è vicina a zero: accade che in alcuni esempi il valore dell'integrale viene completamente sbagliato anche variando la tolleranza. Le spiegazioni sono le stesse date nel caso di polygauss: all'avvicinarsi di ρ ad uno la funzione $f(x, y)$ diventa difficile da valutare e quindi da integrare. Bisogna osservare però che le performance di 'dblquad' e 'twoD' peggiorano anche quando la varianza marginale in x è prossima a zero: i due codici infatti devo affrontare il problema in dimensione due e cioè integrano direttamente la gaussiana in due variabili. Di conseguenza incontrano difficoltà nell'integrazione anche quando $f(x, y)$ è compressa nella direzione dell'asse x, che accade se σ_x^2 è molto piccola. Ciò in realtà avviene anche per 'polygauss', nel quale la x-primitiva viene calcolata numericamente per utilizzare il teorema di Green. Per provare questo si può modificare l'esempio 3.4 scegliendo $\sigma_x^2 = 0.00001$, $\sigma_y^2 = 6$ e lasciando invariati gli altri parametri. Le tabelle 4.25 e 4.26 riportano i risultati che si ottengono con 'polygauss', 'dblquad' e 'twoD'.

	polygauss	dblquad	twoD
Errore ass.	2.0E-02(n=350)	6.0E-01	6.0E-01
Tempo (sec.)	1.34	0.031	0.016

Tabella 4.25: Poligono con 18 lati, esempio 3.4, risultati di 'polygauss', 'dblquad' e 'twoD' con tol = `1E-10`.

	dblquad	twoD
Errore ass.	6.0E-01	6.0E-01
Tempo (sec.)	0.016	0.016

Tabella 4.26: Poligono con 18 lati, esempio 3.4, risultati di 'dblquad' e 'twoD' con tol = `1E-04`.

In definitiva i risultati riportati con 'MySoftware' sono decisamente soddisfacenti: nonostante si siano incontrati anche dei casi problematici, le sue performance sono migliori di quelle degli altri tre metodi. In conclusione è stato creato un algoritmo efficiente per il risolvere il problema dell'integrazione numerica di una gaussiana bivariata su un poligono semplice.

Capitolo 5

Codici Matlab

5.1 Metodo delle catene con VertexHorizontal

Listato MATLAB

```
function [mySoftResult]=MySoftware1(a,b,u,v,r,A,tol)

n=size(A,1);
A(n+1,2)=A(1,2);
A(n+1,1)=A(1,1);

w=A(1:n+1,1)+i*A(1:n+1,2);

T=quadgk(@(z)myfun(z,a,b,u,v,r),w(1),w(n+1),'Waypoints',w(2:n),...
'AbsTol',1.e-14,'RelTol',0);
fprintf('\n \t [RISULTATO ESATTO]: %5.15f',imag(T));

l=0;
for k=1:n
    l=l+abs(A(k,2)-A(k+1,2));
end

gamma=(tol/2)*(1/l);
s=sqrt(-(2*b)*log(2*gamma*sqrt(2*pi*b)));

if s<abs(max(A(:,2)))

    %"METODO CATENE CON LATI ORIZZONTALI"
    [begin,final]=VertexHorizontal(A,v,s);

    atol=(tol/2)*(1/length(begin));
```

```
    mySoftResult=0;
    cputime1=cputime;
    for m=1:length(begin)
       P=quadgk(@(z)myfun(z,a,b,u,v,r),...
       A(begin(m),1)+i*A(begin(m),2),...
       A(final(m),1)+i*A(final(m),2),'Waypoints',...
       A((begin(m)+1):(final(m)-1),1)+i*A((begin(m)+1):(final(m)-1),2),...
       'AbsTol',atol,'RelTol',0);
       mySoftResult=mySoftResult+imag(P);
    end
    cputime2=cputime;

fprintf('\n \t [RISULTATO "METODO CATENE CON LATI ORIZZ."]:%5.15e',...
mySoftResult);
fprintf('\n \t [ERRORE ASSOLUTO]: %5.15e',abs(imag(T)-mySoftResult));
fprintf('\n \t [TEMPO IMPIEGATO]: %5.15e',cputime2-cputime1);
else
    warning('MATLAB:My_Software:NessunaCatena', ...
            'Tutti i lati del poligono sono inclusi nella striscia.');

    %'INTEGRAZIONE SU TUTTO IL BORDO DEL POLIGONO'
    cputime5=cputime;
    R=quadgk(@(z)myfun(z,a,b,u,v,r),A(1,1)+i*A(1,2),...
    A(1,1)+i*A(1,2),'Waypoints',A(2:n,1)+i*A(2:n,2));
    cputime6=cputime;
    mySoftResult=imag(R);

 fprintf('\n \t [RISULTATO INTEGRAZIONE SU TUTTO IL BORDO]: %5.15e',...
 mySoftResult);
 fprintf('\n \t [ERRORE ASSOLUTO]: %5.15e',abs(imag(T)-mySoftResult));
 fprintf('\n \t [TEMPO TOTALE IMPIEGATO]: %5.15e',cputime6-cputime5);
end
%-------------------------------------------------------------------------
function w = myfun(z,sigma_xquadro,sigma_yquadro,u,v,r)
z=real(z)+i*imag(z);
z1=(1/2)*(1/sqrt(2*pi*sigma_yquadro)).*exp(-((imag(z)-v).^2)./...
    (2*sigma_yquadro));
z2=erf((sqrt(sigma_xquadro)*r.*(v-imag(z))+sqrt(sigma_yquadro).*...
    (real(z)-u))./(sqrt(2*(1-(r^2)))*sqrt(sigma_xquadro)*...
    sqrt(sigma_yquadro)));
w=z1.*z2;
end
%-------------------------------------------------------------------------
function [begin,final]=VertexHorizontal(A,v,s)
```

```
n=size(A,1);
A(n+1,2)=A(1,2);
A(n+1,1)=A(1,1);
I=[n,1:n+1];
cputime1=cputime;
k=0;
p=1;
row=zeros(n,1);
rowworth=zeros(n,1);
for j=1:n
p1=((A(j,2)-A(j+1,2)~=0 & (abs(A(j,2)-v)<s | abs(A(j+1,2)-v)<s |...
    (v-A(j+1,2)>=s & A(j,2)-v>=s) | (A(j+1,2)-v>=s & v-A(j,2)>=s) |...
    abs(A(I(j),2)-v)<s | (v-A(I(j),2)>=s & A(j,2)-v>=s) |...
    (A(I(j),2)-v>=s & v-A(j,2)>=s))) | (A(j,2)-A(j+1,2)==0 ...
    & (abs(A(j,2)-v)<s | abs(A(I(j),2)-v)<s | (v-A(j,2)>=s &...
    A(I(j),2)-v>=s) | (A(j,2)-v>=s & v-A(I(j),2)>=s))));
p2=(A(j,2)-A((j)+1,2)==0 & ((abs(A(j,2)-v)>s & abs(A(I(j),2)-v)<s) |...
    ((A(j,2)-v)>s & (v-A(I(j),2))>s) | ((v-A(j,2))>s & (A(I(j),2)-v)>s)));
 if (p1 == 1)
     k=k+1;
     row(k)=j;
 end;
 if (p2 == 1)
     rowworth(p)=j;
     p=p+1;
 end;
end;
if row(n)~=n
  h=1;
  t=1;
  if row(1)~=row(2)-1 || row(1)==rowworth(t)
     begin(h)=row(2);
       if row(1)==rowworth(t)
           t=t+1;
       end
  else
      begin(h)=row(1);
  end

  j=0;
  for i=2:k
      if (row(i)~=row(i+1)-1 || row(i)==rowworth(t))
          h=h+1;
          j=j+1;
```

```
          begin(h)=row(i+1);
          final(j)=row(i);
          if row(i)==rowworth(t)
             t=t+1;
          end
       end
  end
  begin(h)=[];

  if row(1)~=row(2)-1 || row(1)==rowworth(1)
     final(j)=row(1);
  end

  if row(1)==1 && row(k)==n && row(k)~=rowworth(p) &&...
     row(1)~=rowworth(1) && row(1)==row(2)-1
     begin(1)=begin(h-1);
     begin(j)=[];
     final(j)=[];
  end
cputime2=cputime;
fprintf('\n \t [TEMPO DI CALCOLO DELLE CATENE]: %5.15f',cputime2-cputime1);

x=linspace(min(A(:,1))-1,max(A(:,1))+1,1000);
hold on
plot(x,v+s,'LineWidth',2,'Color',[0.87 0.49 0])
plot(x,v-s,'LineWidth',2,'Color',[0.87 0.49 0])
plot(A(1:(n+1),1),A(1:(n+1),2))
if row(1)==1 && row(k)==n && row(k)~=rowworth(p) && row(1)==row(2)-1
    if row(1)~=rowworth(1)
        for i=2:length(begin)
        plot(A(begin(i):final(i),1),A(begin(i):final(i),2),'rs')
        plot(A(begin(i):final(i),1),A(begin(i):final(i),2),'LineWidth',...
        3,'Color',[.8 0 0])
        end
        x1=[A(begin(1):n,1);A(1:final(1),1)];
        y1=[A(begin(1):n,2);A(1:final(1),2)];
        plot(x1,y1,'rs')
        plot(x1,y1,'LineWidth',3,'Color',[.8 0 0])
    else
        for i=1:(length(begin)-1)
        plot(A(begin(i):final(i),1),A(begin(i):final(i),2),'rs')
        plot(A(begin(i):final(i),1),A(begin(i):final(i),2),'LineWidth',...
        3,'Color',[.8 0 0])
        end
```

```
        x1=[A(begin(length(begin)):n,1);A(1:final(length(final)),1)];
        y1=[A(begin(length(begin)):n,2);A(1:final(length(final)),2)];
        plot(x1,y1,'rs')
        plot(x1,y1,'LineWidth',3,'Color',[.8 0 0])
    end
else
    for i=1:length(begin)
        plot(A(begin(i):final(i),1),A(begin(i):final(i),2),'rs')
        plot(A(begin(i):final(i),1),A(begin(i):final(i),2),'LineWidth',...
        3,'Color',[.8 0 0])
    end
end
hold off
else
    x=linspace(min(A(:,1))-1,max(A(:,1))+1,1000);
    hold on
    plot(x,v+s,'LineWidth',2,'Color',[0.87 0.49 0])
    plot(x,v-s,'LineWidth',2,'Color',[0.87 0.49 0])
    plot(A(1:(n+1),1),A(1:(n+1),2),'rs')
    plot(A(1:(n+1),1),A(1:(n+1),2),'LineWidth',3,'Color',[.8 0 0])
    hold off
    warning('MATLAB:vertex_with_or:NessunaCatena', ...
            'Tutti i lati del poligono intersecano la striscia.');
    begin=[];
    final=[];
end
```

5.2 Metodo delle catene con VertexNotHorizontal

Listato MATLAB

```
function [mySoftResult]=MySoftware2(a,b,u,v,r,A,tol)

n=size(A,1);
A(n+1,2)=A(1,2);
A(n+1,1)=A(1,1);

w=A(1:n+1,1)+i*A(1:n+1,2);

T=quadgk(@(z)myfun(z,a,b,u,v,r),w(1),w(n+1),'Waypoints',w(2:n),...
'AbsTol',1.e-14,'RelTol',0);
fprintf('\n \t [RISULTATO ESATTO]: %5.15f',imag(T));
```

```
l=0;
for k=1:n
    l=l+abs(A(k,2)-A(k+1,2));
end

gamma=(tol/2)*(1/l);
s=sqrt(-(2*b)*log(2*gamma*sqrt(2*pi*b)));

if s<abs(max(A(:,2)))

    %'METODO CATENE SENZA LATI ORIZZONTALI'
    [begin,final]=VertexNotHorizontal(A,v,s);

    atol=(tol/2)*(1/length(begin));
    mySoftResult=0;
    cputime3=cputime;
    for m=1:length(begin)
      Q=quadgk(@(z)myfun(z,a,b,u,v,r),...
      A(begin(m),1)+i*A(begin(m),2),...
      A(final(m),1)+i*A(final(m),2),'Waypoints',...
      A((begin(m)+1):(final(m)-1),1)+i*A((begin(m)+1):(final(m)-1),2),...
      'AbsTol',atol,'RelTol',0);
       mySoftResult=mySoftResult+imag(Q);
    end
    cputime4=cputime;

fprintf('\n \t [RISULTATO "METODO CATENE SENZA LATI ORIZZ."]: %5.15e',...
mySoftResult);
fprintf('\n \t [ERRORE ASSOLUTO]: %5.15e',abs(imag(T)-mySoftResult));
fprintf('\n \t [TEMPO IMPIEGATO]: %5.15e',cputime4-cputime3);
else
    warning('MATLAB:My_Software:NessunaCatena', ...
            'Tutti i lati del poligono sono inclusi nella striscia.');

    %'INTEGRAZIONE SU TUTTO IL BORDO DEL POLIGONO'
    cputime5=cputime;
    R=quadgk(@(z)myfun(z,a,b,u,v,r),A(1,1)+i*A(1,2),...
    A(1,1)+i*A(1,2),'Waypoints',A(2:n,1)+i*A(2:n,2));
    cputime6=cputime;
    mySoftResult=imag(R);

fprintf('\n \t [RISULTATO INTEGRAZIONE SU TUTTO IL BORDO]: %5.15e',...
imag(R));
```

```
fprintf('\n \t [ERRORE ASSOLUTO]: %5.15e',abs(imag(T)-mySoftResult));
fprintf('\n \t [TEMPO IMPIEGATO]: %5.15e',cputime6-cputime5);
end
%-------------------------------------------------------------------------
function w = myfun(z,sigma_xquadro,sigma_yquadro,u,v,r)
z=real(z)+i*imag(z);
z1=(1/2)*(1/sqrt(2*pi*sigma_yquadro)).*exp(-((imag(z)-v).^2)./...
    (2*sigma_yquadro));
z2=erf((sqrt(sigma_xquadro)*r.*(v-imag(z))+sqrt(sigma_yquadro).*...
    (real(z)-u))./(sqrt(2*(1-(r^2)))*sqrt(sigma_xquadro)*...
    sqrt(sigma_yquadro)));
w=z1.*z2;
end
%-------------------------------------------------------------------------

function [begin,final]=VertexNotHorizontal(A,v,s)
n=size(A,1);
A(n+1,2)=A(1,2);
A(n+1,1)=A(1,1);
I=[n,1:n+1];
cputime1=cputime;
k=0;
p=0;
row=zeros(n,1);
rowworth=zeros(n,1);
for j=1:n
p1=((A(j,2)-A(j+1,2)~=0 & (abs(A(j,2)-v)<s | abs(A(j+1,2)-v)<s |...
   (v-A(j+1,2)>=s & A(j,2)-v>=s) | (A(j+1,2)-v>=s & v-A(j,2)>=s) |...
    abs(A(I(j),2)-v)<s | (v-A(I(j),2)>=s & A(j,2)-v>=s) |...
   (A(I(j),2)-v>=s & v-A(j,2)>=s))) | (A(j,2)-A(j+1,2)==0 &...
   (abs(A(j,2)-v)<s | abs(A(I(j),2)-v)<s |...
   (v-A(j,2)>=s & A(I(j),2)-v>=s) | (A(j,2)-v>=s & v-A(I(j),2)>=s))));
p2=((A(j,2)-A(j+1,2))==0 & (abs(A(j,2)-v)<s | abs(A(I(j),2)-v)<s |...
    (v-A(j,2)>=s & A(I(j),2)-v>=s) | (A(j,2)-v>=s & v-A(I(j),2)>=s)));
 if (p1 == 1)
     k=k+1;
     row(k)=j;
 end;
 if (p2 == 1)
     p=p+1;
     rowworth(p)=j;
 end;
end;
if row(n)~=n
```

```
  h=1;
  t=1;
  if row(1)~=row(2)-1 || row(1)==rowworth(t)
      begin(h)=row(2);
       if row(1)==rowworth(t)
           t=t+1;
       end
  else
      begin(h)=row(1);
  end

  j=0;
  for i=2:k
      if (row(i)~=row(i+1)-1 || row(i)==rowworth(t))
          h=h+1;
          j=j+1;
          begin(h)=row(i+1);
          final(j)=row(i);
          if row(i)==rowworth(t)
             t=t+1;
          end
      end
  end
  begin(h)=[];

  if row(1)~=row(2)-1 || row(1)==rowworth(1)
      final(j)=row(1);
  end
```

5.3 Integrazione su tutto il bordo

Listato MATLAB

```
function [mySoftResult]=MySoftware3(a,b,u,v,r,A,tol)

n=size(A,1);
A(n+1,2)=A(1,2);
A(n+1,1)=A(1,1);

w=A(1:n+1,1)+i*A(1:n+1,2);

T=quadgk(@(z)myfun(z,a,b,u,v,r),w(1),w(n+1),'Waypoints',w(2:n),...
```

```
'AbsTol',1.e-14,'RelTol',0);
fprintf('\n \t [RISULTATO ESATTO]: %5.15f',imag(T));

%'INTEGRAZIONE SU TUTTO IL BORDO DEL POLIGONO'
cputime5=cputime;
R=quadgk(@(z)myfun(z,a,b,u,v,r),w(1),w(n+1),...
'Waypoints',w(2:n),'AbsTol',tol,'RelTol',0);
cputime6=cputime;
mySoftResult=imag(R);

fprintf('\n \t [RISULTATO INTEGRAZIONE SU TUTTO IL BORDO]: %5.15e',...
mySoftResult);
fprintf('\n \t [ERRORE ASSOLUTO]: %5.15e',abs(imag(T)-imag(R)));
fprintf('\n \t [TEMPO IMPIEGATO]: %5.15e',cputime6-cputime5);
end
```

Bibliografia

[1] T. M. Apostol, *Calculus*, vol. II, 2nd edition, Blaisdell, 1969.

[2] V. Comincioli. *Analisi numerica: Metodi Modelli Applicazioni.* Nuova edizione, in formato e-book, 981 pp. Apogeo, Feltrinelli Milano, `http://www.apogeonline.com`, 2005.

[3] W. Gautschi, *Orthogonal Polynomials: Computation and Approximation*, Oxford University Press, Oxford, 2004.

[4] The MathWorks, *MATLAB Documentation Set*, 2010 version (`http://www.mathworks.com`).

[5] S. Meyer, *Spatio-Temporal Infectious Disease Epidemiology based on Point Processes*, Master's Thesis, 18 December 2009.

[6] W. Pleśniak, *Remarks on Jackson's theorem in* $\mathbb{R}^N$, East J. Approx. 2 (1996), no. 3, 301-308.

[7] S. M. Ross, *Calcolo delle probabilità*, Apogeo, 2004.

[8] L. F. Shampine, *MATLAB Program for Quadrature in 2D*, Appl. Math. Comput. 202 (2008),266-274.

[9] L. F. Shampine, *TwoD Numerically evaluate integral of* $f(x, y)$ *over a plane region*, software downloadable from: `http://faculty.smu.edu/shampine/current.html`.

[10] A. Sommariva and M. Vianello, *Product Gauss cubature over polygons based on Green's integration formula*, BIT Numerical Mathematics 47 (2007), 441-453.

[11] A. Sommariva and M. Vianello, *Polygauss: Matlab code for Gauss-like cubature over polygons*, software downloadable from: `www.math.unipd.it/~marcov/software.html`.

[12] E. W. Weisstein, *Bivariate Normal Distribution.* From *MathWorld*-A Wolfram Web Resource. (`http://mathworld.wolfram.com/BivariateNormalDistribution.html`)

[13] E. W. Weisstein, *Erf.* From *MathWorld*-A Wolfram Web Resource. (`http://mathworld.wolfram.com/Erf.html`)

[14] E. W. Weisstein, *Normal Distribution.* From *MathWorld*-A Wolfram Web Resource. (`http://mathworld.wolfram.com/NormalDistribution.html`)

[15] E. W. Weisstein, *Simple Polygon.* From *MathWorld*-A Wolfram Web Resource. (`http://mathworld.wolfram.com/SimplePolygon.html`)

[16] Wolfram *Mathematica*, Online Integrator (2010), (`http://integrals.wolfram.com/index.jsp`).