

POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Matematica

Metodi Numerici per le equazioni alle derivate parziali

Metodo degli elementi finiti per problemi evolutivi e di diffusione-convezione-reazione



Supervisori

prof. Stefano Berrone
prof. Claudio Canuto

Candidato

Lorenzo Bellantuono
Michele Lupini
Andrea Tataranni

Anno Accademico 2022-2023

Indice

Introduzione	2
1 Impostazione del problema discreto	3
1.1 Mesh triangolare	3
1.2 Problema stazionario	6
2 Polinomi di grado 1	9
2.1 Introduzione teorica	9
2.2 Simulazione	11
2.3 Calcolo degli errori e degli ordini di convergenza	13
2.4 Codice Matlab	15
3 Polinomi di grado 2	20
3.1 Introduzione teorica	20
3.2 Simulazione	21
3.3 Codice Matlab	24
4 Problema di diffusione-convezione	27
4.1 Problema di diffusione-convezione	27
4.2 Problema in 1D	27
4.3 Problema in 2D	28
4.4 Dettagli di implementazione	32
4.5 Codice Matlab	33
5 Problema di diffusione-reazione	35
5.1 Problema di diffusione-reazione	35
5.2 Codice Matlab	38
6 Problema parabolico	39
6.1 Problema Parabolico	39
6.2 Ordine di convergenza	42
6.3 Dettagli di implementazione	45
6.4 Codice Matlab	46

Introduzione

Diversi fenomeni fisici si presentano come processi di diffusione-trasporto-reazione e possono essere descritti matematicamente da equazioni alle derivate parziali. In questa trattazione descriveremo l'approccio suggerito dal metodo degli elementi finiti (FEM, dall'inglese *Finite Element Method*) e ne analizzeremo le applicazioni per alcuni problemi prototipo. A partire da un sistema di equazioni differenziali alle derivate parziali su un certo dominio di definizione, è possibile ridurre il problema a un sistema di equazioni algebriche con il metodo degli elementi finiti, sfruttando la costruzione di una mesh per discretizzare il dominio. L'approssimazione della soluzione è strettamente influenzata dalla qualità della discretizzazione e per questo è opportuno caratterizzare gli elementi in modo che siano sufficientemente regolari.

Il sistema lineare così ottenuto può essere risolto con le tecniche standard del calcolo numerico, per determinare la soluzione approssimata del problema. Nelle applicazioni che studieremo ci serviremo di metodi diretti in virtù della ridotta dimensione delle matrici generate dai problemi, tuttavia osserviamo che in presenza di matrici sparse di grandi dimensioni è indicato servirsi di metodi iterativi in modo da ridurre i tempi di calcolo e l'uso della memoria. Un'ulteriore strategia per ottimizzare la risoluzione del problema prevede l'adattamento della mesh alle specifiche caratteristiche del dominio e della soluzione, infatti la costruzione degli elementi finiti può essere un'operazione dispendiosa e, qualora fosse richiesta una notevole precisione dell'approssimazione, in principio sarebbe necessario ricorrere ad una discretizzazione globalmente fine del dominio. Tuttavia raffinando la mesh in prossimità delle singolarità e mantenendola più grossolana lontano da queste, è possibile ridurre l'errore per un costo computazionale inferiore.

Nello studio dei problemi ci occuperemo di correlare la taglia degli elementi finiti con l'errore di approssimazione, in particolare sullo stesso dominio costruiremo diverse discretizzazioni, caratterizzate da elementi finiti con certi valori di area massima precedentemente definiti, e conoscendo la soluzione esatta calcoleremo l'errore con cui poi stimeremo l'ordine di convergenza sperimentale confrontandolo con i risultati teorici.

Nel caso stazionario considereremo in generale un problema di diffusione-convezione-reazione misto di Dirichlet-Neumann e nelle successive applicazioni analizzeremo nel dettaglio i casi di diffusione-convezione e diffusione-reazione, introducendo rispettivamente il metodo di stabilizzazione *Streamline Upwind Petrov Galerkin* e il processo di *mass lumping*. Una variazione sul tema ci permetterà di trattare i problemi evolutivi, infatti con il metodo delle linee sfrutteremo la precedente struttura di discretizzazione spaziale e vi aggiungeremo la discretizzazione temporale con uno schema di avanzamento nel tempo.

Capitolo 1

Impostazione del problema discreto

Richiamiamo la seguente definizione assiomatica di elemento finito.

Definizione 1. Un elemento finito in \mathbb{R}^d è una tripla (E, V_E, \mathcal{L}_E) , dove:

- $E \neq \emptyset$ è un insieme compatto, connesso e perfetto ($\overline{\text{int}(E)} = E$) in \mathbb{R}^d tale che ∂E sia Lipschitz-continuo,
- V_E è uno spazio vettoriale di funzioni definite su E , tale che $\dim V_E = N_E < +\infty$,
- $\mathcal{L}_E = \{\ell_j : V_E \rightarrow \mathbb{R} \text{ forme lineari}, j = 1, \dots, N_E\}$ unisolvente per V_E .

Inoltre ad ogni elemento finito associamo una base canonica Φ in V_E che permette di rappresentare ogni funzione attraverso i suoi gradi di libertà:

$$\Phi = \{\varphi_k \in V_E, k = 1, \dots, N_E\}$$

dove φ_k sono univocamente definite dalle condizioni

$$\ell_j(\varphi_k) = \delta_{jk}, \quad j = 1, \dots, N_E$$

In modo che ogni $v \in V_E$ si possa rappresentare come

$$v = \sum_{k=1}^{N_E} \ell_k(v) \varphi_k$$

1.1 Mesh triangolare

Nello specifico tratteremo il caso di dominio bidimensionale poligonale $\Omega \subset \mathbb{R}^2$ e per farlo introduciamo la seguente definizione

Definizione 2. Una partizione $\mathcal{T} = \{E_n\}_{n=1}^N$ di Ω si dice triangolazione conforme se è costituita da N elementi triangolari E_n tali che

- $\bigcup_{n=1}^N E_n = \overline{\Omega}$,
- $\text{int}(E_l) \cap \text{int}(E_m) = \emptyset \quad \forall l \neq m, \quad l, m = 1, \dots, N,$
- $E_l \cap E_m$ è vuoto, un punto o un intero lato comune per i due triangoli $\forall l \neq m$.

Nel seguito denoteremo i vertici dei triangoli in \mathcal{T} con \mathbf{a}_k per $k = 1, \dots, N_P$, e per ogni triangolo $E \in \mathcal{T}$ introduciamo una mappa suriettiva invertibile che associa l'indice globale k di ciascuno dei suoi vertici al corrispettivo indice locale in $\{1, 2, 3\}$. Denoteremo quindi con k_1, k_2, k_3 gli indici globali dei vertici del triangolo $E \in \mathcal{T}$ e con 1, 2, 3 i rispettivi indici locali, in entrambi i casi supposti in senso antiorario.

Nelle applicazioni la creazione della mesh triangolare del dominio sarà affidata al generatore Matlab® chiamato BBTR, in grado di costruire una triangolazione Delaunay di un dominio poligonale non necessariamente convesso. In particolare, avremo a disposizione una triangolazione in cui si massimizza l'angolo $\theta_{\mathcal{T}} = \min_{E \in \mathcal{T}} \theta_E$, dove $\theta_E \leq \pi/3$ è il minimo angolo interno dell'elemento $E \in \mathcal{T}$, cioè tutti i triangoli dovranno avere angoli interni non troppo piccolo (o equivalentemente non troppo grandi). Inoltre eserciteremo un controllo sull'area massima dei triangoli che costituiscono la mesh, nel seguito denotata come **RefiningArea**, e nella stima dell'ordine di convergenza della soluzione approssimata ci serviremo del diametro massimo dell'elemento definito come $h = \sqrt{\text{RefiningArea}}$, che fornisce una misura del diametro degli elementi della mesh. Infatti ricordiamo che per ogni elemento $E \in \mathcal{T}$, posti A_E la sua area, h_E la lunghezza del suo lato più lungo, θ_E il suo angolo interno minimo, si ha

$$\frac{h_E^2}{4} \sin \theta_E \leq A_E \leq \frac{h_E^2}{2}.$$

Introduciamo l'elemento di riferimento triangolare

$$\hat{E} = \left\{ \hat{\mathbf{x}} = \begin{bmatrix} \hat{x} \\ \hat{y} \end{bmatrix} \subset \mathbb{R}^2 \mid 0 \leq \hat{x}, \hat{y} \leq 1, 0 \leq 1 - \hat{x} - \hat{y} \leq 1 \right\}.$$

Ogni triangolo $E \in \mathcal{T}$ con vertici $\{\mathbf{a}_k = (x_k, y_k)\}_{k=1}^3$ può essere costruito a partire dall'elemento di riferimento attraverso la mappa affine

$$\mathbf{x} = \mathbf{F}_E(\hat{\mathbf{x}}) = \mathbf{a}_3 + \mathbf{B}\hat{\mathbf{x}} = \mathbf{a}_1\hat{N}_1(\hat{\mathbf{x}}) + \mathbf{a}_2\hat{N}_2(\hat{\mathbf{x}}) + \mathbf{a}_3\hat{N}_3(\hat{\mathbf{x}}),$$

dove \mathbf{B} è la matrice avente come colonne i lati del triangolo con origine \mathbf{a}_3 : $\mathbf{v}_1 = \mathbf{a}_1 - \mathbf{a}_3$, $\mathbf{v}_2 = \mathbf{a}_2 - \mathbf{a}_3$, cioè

$$\mathbf{B} = \begin{bmatrix} x_1 - x_3 & x_2 - x_3 \\ y_1 - y_3 & y_2 - y_3 \end{bmatrix}$$

oppure, equivalentemente, $\{\hat{N}_k(\hat{\mathbf{x}})\}_{k=1}^3$ sono le funzioni della base di Lagrange in $\mathbb{P}_1(\hat{E})$

$$\begin{aligned} \hat{N}_1(\hat{x}, \hat{y}) &= \hat{x}, \\ \hat{N}_2(\hat{x}, \hat{y}) &= \hat{y}, \\ \hat{N}_3(\hat{x}, \hat{y}) &= 1 - \hat{x} - \hat{y}. \end{aligned} \tag{1.1}$$

Inoltre osserviamo che la mappa \mathbf{F}_E è invertibile se e solo se l'elemento E è non degenere, infatti per l'ipotesi di ordinamento antiorario dei vertici, si ha che $\det \mathbf{B} = 2|E|$, dove $|E|$ è l'area del triangolo E

Denotiamo con $\hat{\nabla} := \nabla_{\hat{x}, \hat{y}}$ il gradiente rispetto alle coordinate \hat{x}, \hat{y} dell'elemento di riferimento e con $\nabla := \nabla_{x,y}$ il gradiente rispetto alle coordinate x, y e ricordiamo che la matrice jacobiana di una funzione $\mathbf{G} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ è data da

$$\mathbf{J}_{\mathbf{G}} = \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \cdots & \frac{\partial g_1}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial g_m}{\partial x_1} & \cdots & \frac{\partial g_m}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \nabla g_1^t \\ \vdots \\ \nabla g_m^t \end{bmatrix}$$

Consideriamo le funzioni $\hat{f} : \hat{E} \subset \mathbb{R}^2 \rightarrow \mathbb{R}$, $f : E \subset \mathbb{R}^2 \rightarrow \mathbb{R}$ tali che

$$\hat{f}(\hat{x}, \hat{y}) = f(\mathbf{F}_E(\hat{x}, \hat{y})) \quad \forall (\hat{x}, \hat{y}) \in \hat{E}.$$

Osservando che in questo caso le matrici jacobiane di \hat{f}, f coincidono con i rispettivi gradienti trasposti, cioè

$$\begin{aligned} \mathbf{J}_{\hat{f}}(\hat{x}, \hat{y}) &= \hat{\nabla} \hat{f}(\hat{x}, \hat{y})^t = \begin{bmatrix} \frac{\partial \hat{f}}{\partial \hat{x}} & \frac{\partial \hat{f}}{\partial \hat{y}} \end{bmatrix}, \\ \mathbf{J}_f(x, y) &= \nabla f(x, y)^t = \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \end{bmatrix}, \end{aligned}$$

e che la matrice jacobiana di \mathbf{F}_E è data da $\mathbf{J}_{\mathbf{F}_E}(\hat{x}, \hat{y}) = \mathbf{B}$, per la regola della catena per funzioni composte si ha che

$$\mathbf{J}_{\hat{f}}(\hat{x}, \hat{y}) = \mathbf{J}_f(\mathbf{F}_E(\hat{x}, \hat{y})) \mathbf{J}_{\mathbf{F}_E}(\hat{x}, \hat{y}) = \mathbf{J}_f(\mathbf{F}_E(\hat{x}, \hat{y})) \mathbf{B}$$

da cui, riconducendoci ai gradienti, otteniamo

$$\begin{aligned} \hat{\nabla} \hat{f}(\hat{x}, \hat{y}) &= \mathbf{B}^t \nabla f(\mathbf{F}_E(\hat{x}, \hat{y})), \\ \nabla f(\mathbf{F}_E(\hat{x}, \hat{y})) &= \mathbf{B}^{-t} \hat{\nabla} \hat{f}(\hat{x}, \hat{y}). \end{aligned}$$

Osservazione. Nel seguito faremo uso delle seguenti proprietà di integrazione, basate sui cambiamenti di variabile appena definiti:

$$\begin{aligned} \int_E f(x, y) dx dy &= \int_{\hat{E}} f(\mathbf{F}_E(\hat{x}, \hat{y})) |\det \mathbf{B}| d\hat{x} d\hat{y} \\ &= 2|E| \int_{\hat{E}} f(\mathbf{F}_E(\hat{x}, \hat{y})) d\hat{x} d\hat{y} \\ \int_E \nabla f(x, y) \cdot \nabla g(x, y) dx dy &= \int_{\hat{E}} (\nabla f(\mathbf{F}_E(\hat{x}, \hat{y})))^t \nabla g(\mathbf{F}_E(\hat{x}, \hat{y})) |\det \mathbf{B}| d\hat{x} d\hat{y} \\ &= 2|E| \int_{\hat{E}} (\hat{\nabla} \hat{f}(\hat{x}, \hat{y}))^t \mathbf{B}^{-1} \mathbf{B}^{-t} \hat{\nabla} \hat{g}(\hat{x}, \hat{y}) d\hat{x} d\hat{y} \end{aligned}$$

1.2 Problema stazionario

Consideriamo un problema di diffusione-convezione-reazione misto di Dirichlet-Neumann sul dominio $\Omega \subset \mathbb{R}^d$ ($d = 1, 2, 3$), con bordo $\partial\Omega$ sufficientemente regolare suddiviso nelle porzioni di bordo Γ_D con condizioni di Dirichlet e Γ_N con condizioni di Neumann, che ne costituiscono una partizione, cioè tali che $\partial\Omega = \Gamma_D \cup \Gamma_N$ e $\Gamma_D \cap \Gamma_N = \emptyset$:

$$\begin{cases} -\nabla \cdot (\nu \nabla u) + \beta \cdot \nabla u + \gamma u = f & \text{in } \Omega \\ u = g_D & \text{su } \Gamma_D \\ \nu \frac{\partial u}{\partial n} = g_N & \text{su } \Gamma_N \end{cases} \quad (1.2)$$

dove $\nu, \beta, \gamma, f, g_D, g_N$ sono funzioni assegnate tali che $\nu, \gamma \in L^\infty(\Omega)$, $\beta \in [L^\infty(\Omega)]^d$, $f \in L^2(\Omega)$, $g_D \in H^{1/2}(\Gamma_D)$, $g_N \in L^2(\Gamma_N)$ con $\nu(x) \geq \nu_0 > 0$ q.o. in Ω , $\nabla \cdot \beta \in L^\infty(\Omega)$, $\beta \cdot \mathbf{n} \geq 0$ su Γ_N , $-\frac{1}{2}\nabla \cdot \beta(x) + \gamma(x) \geq \gamma_0 > 0$ q.o. in Ω .

Data una curva $\Gamma_D \subset \partial\Omega$ e una funzione g a valori su $\partial\Omega$, definiamo in generale il seguente sottospazio di $H^1(\Omega)$

$$H_{g,\Gamma_D}^1(\Omega) = \{v \in H^1(\Omega) \mid v|_{\Gamma_D} = g\}$$

e supponiamo che ogni $v \in H_{g,\Gamma_D}^1(\Omega)$ possa essere scritta come $v = v_g + v_0$ per qualche $v_0 \in H_{0,\Gamma_D}^1(\Omega)$.

Introducendo le seguenti forme bilineare e lineare

$$\begin{aligned} a : H^1(\Omega) \times H^1(\Omega) &\rightarrow \mathbb{R}, & a(u, v) &= \int_{\Omega} \nu \nabla u \cdot \nabla v + \int_{\Omega} (\beta \cdot \nabla u)v + \int_{\Omega} \gamma uv \\ F : H^1(\Omega) &\rightarrow \mathbb{R}, & F(v) &= \int_{\Omega} fv + \int_{\Gamma_N} g_N v, \end{aligned}$$

e scrivendo $u = u_g + u_0$, dove $u_g \in H^1(\Omega)$ è una funzione fissata e nota tale che la sua traccia su Γ_D sia g_D e si dice rilevamento al bordo di g_D , mentre $u_0 \in H_{0,\Gamma_D}^1(\Omega)$ è un'incognita, la formulazione variazionale del problema (1.2) si esprime come

$$[\text{PV}] \quad \begin{cases} \text{trovare } u_0 \in H_{0,\Gamma_D}^1(\Omega) \text{ tale che} \\ a(u_0, v) = F(v) - a(u_g, v) \quad \forall v \in H_{0,\Gamma_D}^1(\Omega) \end{cases}$$

Supponiamo che $\Omega \subset \mathbb{R}^2$ sia un dominio poligonale e che \mathcal{T} sia la mesh triangolare generata con BBTR formata da celle triangolari $E \in \mathcal{T}$. Scegliendo un elemento finito lagrangiano (E, V_E, \mathcal{L}_E) , il problema variazionale [PV] può essere discretizzato con il metodo di Galerkin. Introduciamo il sottospazio $\tilde{V}_h = V_h(\mathcal{T}) \subset H^1(\Omega)$ di dimensione N_p e il sottospazio $V_h = V_h(\mathcal{T}) \subset H_{0,\Gamma_D}^1(\Omega)$ di dimensione $N_{dof} \leq N_p$ pari al numero di gradi di libertà globali della discretizzazione. Denotando e con $\Phi_h = \{\varphi_j\}_{j=1}^{N_{dof}}$ le funzioni della base lagrangiana in V_h , abbiamo che

$$V_h = \text{span} \{ \varphi_j, j = 1, \dots, N_{dof} \}.$$

Definendo $\Phi_h^D = \{\varphi_j^D\}_{j=1}^{N_D}$ l'insieme delle funzioni della base lagrangiana in \tilde{V}_h a meno delle funzioni in Φ_h , si ha

$$\tilde{V}_h = \text{span} \{ \varphi_j, \varphi_i^D, j = 1, \dots, N_{dof}, i = 1, \dots, N_D \}.$$

La formulazione variazione discreta del problema è data da

$$[\text{PV}]_h \quad \begin{cases} \text{trovare } u_{0,h} \in V_h \text{ tale che} \\ a(u_0, v_h) = F(v_h) - a(u_g, v_h) \quad \forall v_h \in V_h \end{cases}.$$

Inoltre fissando la base Φ_h , possiamo sfruttare l'isomorfismo $V_h \simeq \mathbb{R}^{N_{dof}}$ per identificare le funzioni di V_h come vettori in $\mathbb{R}^{N_{dof}}$

$$u_{0,h} = \sum_{j=1}^{N_{dof}} u_{0,j} \varphi_j \in V_h \longleftrightarrow \mathbf{u}_0 = \begin{pmatrix} u_{0,1} \\ \vdots \\ u_{0,N_{dof}} \end{pmatrix} \in \mathbb{R}^{N_{dof}} \quad (1.3)$$

Il problema variazionale discreto $[\text{PV}]_h$ si riduce al seguente sistema algebrico

$$\mathbf{A}\mathbf{u}_0 = \mathbf{f} - \mathbf{A}_D\mathbf{u}_D \quad (1.4)$$

dove $\mathbf{u}_0 \in \mathbb{R}^{N_{dof}}$ è incognita e abbiamo introdotto il vettore $\mathbf{f} = (f_j)_{1 \leq j \leq N_{dof}} \in \mathbb{R}^{N_{dof}}$ e le due matrici $\mathbf{A} = (a_{jk})_{1 \leq j,k \leq N_{dof}} \in \mathbb{R}^{N_{dof} \times N_{dof}}$, $\mathbf{A}_D = (a_{jk}^D)_{1 \leq j \leq N_{dof}, 1 \leq k \leq N_D} \in \mathbb{R}^{N_{dof} \times N_D}$ tali che

$$f_j = F(\varphi_j), \quad a_{jk} = a(\varphi_k, \varphi_j), \quad a_{jk}^D = a(\varphi_k^D, \varphi_j^D).$$

e il vettore $\mathbf{u}_D \in \mathbb{R}^{N_D}$ è associato al rilevamento di g_D in H^1 nel modo seguente

$$u_g \approx \sum_{k_D=1}^{N_D} g_D(\mathbf{a}_{vert(k_D)}) \varphi_j^D \longleftrightarrow \mathbf{u}_D = \begin{pmatrix} g_D(\mathbf{a}_{vert(1)}) \\ \vdots \\ g_D(\mathbf{a}_{vert(N_D)}) \end{pmatrix} \in \mathbb{R}^{N_D} \quad (1.5)$$

dove $k_D \in \{1, \dots, N_D\}$ è un indice che descrive localmente i punti della mesh che si trovano sul bordo di Dirichlet e $vert(k_D) \in \{1, \dots, N_p\}$ è una funzione che lega tale numerazione locale con quella globale.

Per ogni triangolo $E \in \mathcal{T}$ tale che $E \subset \text{supp } \varphi_j$, definiamo $\phi_j := \varphi_j|_E$ la restrizione di φ_j ad E , dove, detto N^E il numero di gradi di libertà su ogni elemento, $\hat{j} = \hat{j}_E(j) \in \{1, \dots, N^E\}$ permette di correlare l'indice globale $j \in \{1, \dots, N_{dof}\}$, associato al grado di libertà globale φ_j , all'indice locale \hat{j} associato al corrispondente grado di libertà locale dell'elemento E . Introducendo la forma bilineare a_E e la forma lineare F_E definite come le forme a, F dove gli integrali su Ω sono sostituiti da integrali su E e l'integrale su Γ_N è sostituito dall'integrale su $\partial E \cap \Gamma_N$, possiamo definire $a_{jk}^E := a_E(\phi_{\hat{j}}, \phi_{\hat{k}})$ e $f_j^E := F_E(\phi_j)$ si ha che

$$\begin{aligned} a_{jk} &= a(\varphi_j, \varphi_k) = \sum_E a_{jk}^E, \\ f_j &= F(\varphi_j) = \sum_E f_j^E. \end{aligned}$$

Richiamiamo alcune stime dell'errore di discretizzazione prodotto dal metodo di Galerkin.

Osservazione. Sia $u \in H^1(\Omega)$ soluzione di [PV], $u_h \in V_h$, se scegliamo un elemento finito di classe C^0 , con k grado dei polinomi di base. Allora esistono $c_1, c_2 > 0$ tali che se $u \in H^{k+1}(\Omega)$ si ha

$$\|u - u_h\|_{0,\Omega} \leq c_1 h^{k+1} |u|_{k+1,\Omega} \quad (1.6)$$

$$\|u - u_h\|_{1,\Omega} \leq c_2 h^k |u|_{k+1,\Omega} \quad (1.7)$$

Inoltre, se $u \in W^{k+1,\infty}(\Omega)$ e $\Omega \subset \mathbb{R}^2$, allora esiste $c > 0$ tale che

$$\|u - u_h\|_{\infty,\Omega} \leq \begin{cases} ch^2 |\log h| |u|_{2,\infty,\Omega} & \text{se } k = 1 \\ ch^{k+1} |u|_{k+1,\infty,\Omega} & \text{se } k \geq 2 \end{cases} \quad (1.8)$$

Capitolo 2

Polinomi di grado 1

2.1 Introduzione teorica

Il primo passo è stato trovare la soluzione numerica del problema stazionario (1.2) tramite l'elemento finito di tipo Lagrangiano $(E, \mathbb{P}_1(E), \mathcal{L}_E^{(1)})$, detto **Elemento di Courant**, su ciascun triangolo della mesh. In questo caso, E è un triangolo non degenere nel piano di vertici \mathbf{a}_i con $i = 1, 2, 3$ ordinati in senso antiorario, $\mathbb{P}_1(E)$ è lo spazio dei polinomi di grado totale minore o uguale a 1 sul triangolo E e infine $\mathcal{L}_E^{(1)}$ è l'insieme dei gradi di libertà corrispondente ai valori assunti dalle funzioni di $\mathbb{P}_1(E)$ nei vertici \mathbf{a}_i , ossia $\mathcal{L}_E^{(1)} = \{v(\mathbf{a}_i) \mid 1 \leq i \leq 3 \wedge v \in \mathbb{P}_1(E)\}$.

Pertanto, seguendo il procedimento esposto nel Paragrafo 1.2, si giunge al sistema algebrico (1.4), le cui componenti nel caso dell'elemento di Courant, dopo aver introdotto una formula di quadratura sul triangolo di riferimento con nodi $\hat{\mathbf{x}}_q$ e pesi ω_q , assumono la seguente forma:

$$\mathbf{A} : \quad a_{jk} = \sum_E a_{jk}^E \simeq \sum_E 2|E| \left(\sum_{q=1}^{N_q} \omega_q \left(\nu(\mathbf{F}_E(\hat{x}_q, \hat{y}_q)) \hat{\nabla} \hat{\phi}_{\hat{k}}^t(\hat{x}_q, \hat{y}_q) \mathbf{B}^{-1} \mathbf{B}^{-t} \hat{\nabla} \hat{\phi}_{\hat{j}}(\hat{x}_q, \hat{y}_q) \right. \right. \\ \left. \left. + \beta(\mathbf{F}_E(\hat{x}_q, \hat{y}_q)) \mathbf{B}^{-t} \hat{\nabla} \hat{\phi}_{\hat{k}}(\hat{x}_q, \hat{y}_q) \hat{\phi}_{\hat{j}}(\hat{x}_q, \hat{y}_q) \right. \right. \\ \left. \left. + \gamma(\mathbf{F}_E(\hat{x}_q, \hat{y}_q)) \hat{\phi}_{\hat{k}}(\hat{x}_q, \hat{y}_q) \hat{\phi}_{\hat{j}}(\hat{x}_q, \hat{y}_q) \right) \right) \quad (2.1)$$

$$\mathbf{A}_D : \quad a_{jk_d}^D = \sum_E (a_{jk_d}^D)^E \simeq \sum_E 2|E| \left(\sum_{q=1}^{N_q} \omega_q \left(\nu(\mathbf{F}_E(\hat{x}_q, \hat{y}_q)) \hat{\nabla} \hat{\phi}_{k_d}^t(\hat{x}_q, \hat{y}_q) \mathbf{B}^{-1} \mathbf{B}^{-t} \hat{\nabla} \hat{\phi}_{\hat{j}}(\hat{x}_q, \hat{y}_q) \right. \right. \\ \left. \left. + \beta(\mathbf{F}_E(\hat{x}_q, \hat{y}_q)) \mathbf{B}^{-t} \hat{\nabla} \hat{\phi}_{k_d}(\hat{x}_q, \hat{y}_q) \hat{\phi}_{\hat{j}}(\hat{x}_q, \hat{y}_q) \right. \right. \\ \left. \left. + \gamma(\mathbf{F}_E(\hat{x}_q, \hat{y}_q)) \hat{\phi}_{k_d}(\hat{x}_q, \hat{y}_q) \hat{\phi}_{\hat{j}}(\hat{x}_q, \hat{y}_q) \right) \right) \quad (2.2)$$

dove N_q è il numero di nodi e pesi di quadratura.

$$\begin{aligned} \mathbf{f} : f_j &= F(\varphi_j) = b_j + (b_N)_j \quad \text{con} \\ b_j &= \sum_E b_j^E \simeq \sum_E 2|E| \left(\sum_{q=1}^{N_q} \omega_q f(\mathbf{F}_E(\hat{x}_q, \hat{y}_q)) \hat{\phi}_{\hat{j}}(\hat{x}_q, \hat{y}_q) \right) \\ (b_N)_j &= \sum_{e \in \Gamma_N} \int_e g_N \phi_j|_e^E d\Gamma \end{aligned} \quad (2.3)$$

dove abbiamo separato per comodità il contributo al termine noto dato dalla forzante e il contributo dato dalla condizione al contorno di Neumann.

Analizzando meglio quest'ultimo, per l'elemento di Courant sappiamo che le funzioni di base ϕ_j sono lineari e possiamo calcolare l'integrale di linea sul lato $e \in \Gamma_N$, infatti parametrizzando e come:

$$\gamma(t) := a_{begin} + (a_{end} - a_{begin})t \quad \text{con } 0 \leq t \leq 1 \quad (2.4)$$

dove a_{begin} e a_{end} rappresentano rispettivamente il vertice iniziale e il vertice finale del lato e , quindi avremo che:

$$\int_e g_N \phi_j|_e^E d\Gamma = \int_0^1 g_N(\gamma(t)) \phi_j^E(\gamma(t)) |e| dt \quad (2.5)$$

A questo punto, detti $g_b := g_N(a_{begin}) \wedge g_e := g_N(a_{end})$ possiamo approssimare g_N con una funzione lineare

$$g_N(\gamma(t)) \simeq g_b(1-t) + g_e t$$

e considerando che

$$j \iff begin \Rightarrow \phi_j^E(\gamma(t)) = 1 - t \quad \wedge \quad j \iff end \Rightarrow \phi_j^E(\gamma(t)) = t$$

si ottengono i seguenti:

$$\begin{aligned} j \iff begin &\Rightarrow \int_0^1 g_N(\gamma(t)) \phi_j^E(\gamma(t)) |e| dt = |e| \int_0^1 (g_b(1-t) + g_e t)(1-t) dt = |e| \frac{2g_b + g_e}{6} \\ j \iff end &\Rightarrow \int_0^1 g_N(\gamma(t)) \phi_j^E(\gamma(t)) |e| dt = |e| \int_0^1 (g_b(1-t) + g_e t) t dt = |e| \frac{g_b + 2g_e}{6} \end{aligned} \quad (2.6)$$

Infine, i termini \mathbf{u}_0 e \mathbf{u}_D mantengono la stessa forma con cui sono espressi in (1.3) e (1.5). Nel paragrafo successivo mostriamo una simulazione che abbiamo effettuato.

2.2 Simulazione

I dati scelti per eseguire la simulazione sono i seguenti:

- $u(x, y) = xy - xy^2 + 1$ (Soluzione esatta);
- $\nu(x, y) = x$ (Coefficiente di diffusione);
- $\beta(x, y) = (x; 1)$ (Coefficiente di convezione);
- $\gamma(x, y) = y$ (Coefficiente di reazione);
- $f(x, y) = 2x^2 - xy + x + y^2 - xy^3$ (Termine forzante);
- $g_D(x, y) = 1$ (Condizione al bordo di Dirichlet);
- $g_N(x, y) = y - y^2$ (Condizione al bordo di Neumann);
- aree = [0.005; 0.001; 0.0006] (Valori del parametro `RefiningArea` del triangolatore usati per la simulazione).

Di seguito riportiamo i risultati che abbiamo ottenuto comprensivi dei grafici della soluzione numerica, della soluzione esatta e dell'errore ottenuto come differenza dei primi due, per tutti i valori di area.

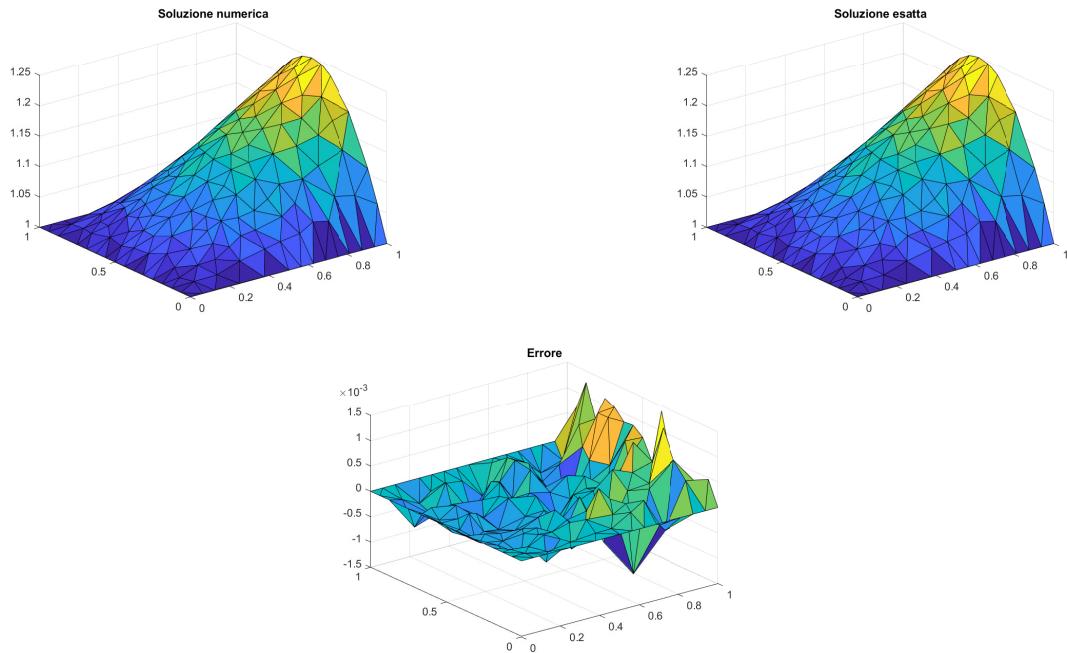


Figura 2.1: Simulazione con `RefiningArea` = 5e-3 e polinomi di base di grado 1

2.2. SIMULAZIONE

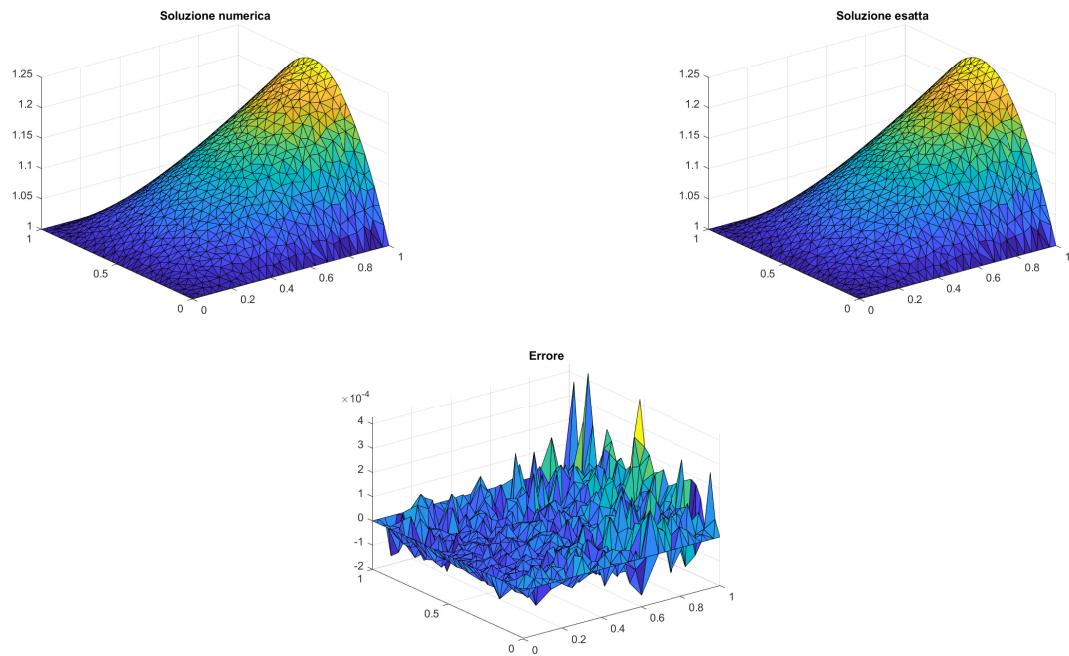


Figura 2.2: Simulazione con RefiningArea = 1e-3 e polinomi di base di grado 1

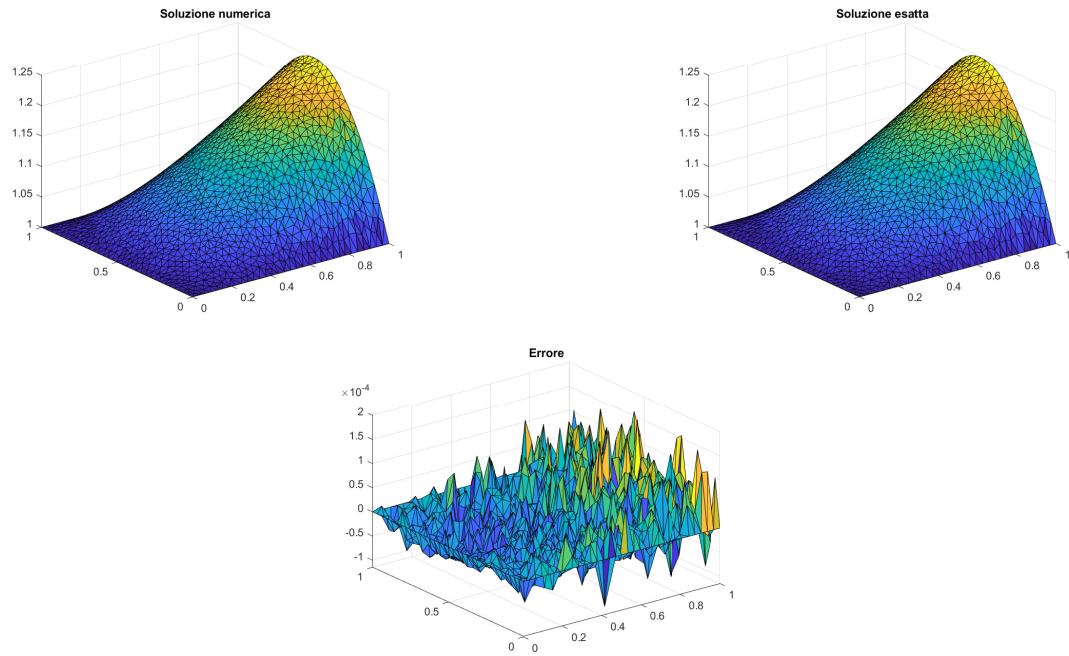


Figura 2.3: Simulazione con RefiningArea = 6e-4 e polinomi di base di grado 1

2.3 Calcolo degli errori e degli ordini di convergenza

Per verificare la correttezza del codice siamo quindi passati al calcolo degli errori e degli ordini di convergenza.

Utilizzando la medesima formula di quadratura sul triangolo di riferimento introdotta ad inizio capitolo, dalla teoria sappiamo che le norme L^2 , H_0^1 e H^1 dell'errore di discretizzazione si calcolano nel seguente modo:

$$\|u - u_h\|_{L^2(\Omega)}^2 \simeq \sum_E 2|E| \sum_{q=1}^{N_q} \omega_q \left(u(\mathbf{F}_E(\hat{x}_q, \hat{y}_q)) - \sum_{\hat{k}=1}^{N^E} u_{G_E(\hat{k})} \hat{\phi}_{\hat{k}}(\hat{x}_q, \hat{y}_q) \right)^2 \quad (2.7)$$

$$\begin{aligned} \|\nabla u - \nabla u_h\|_{L^2(\Omega)}^2 &\simeq \sum_E 2|E| \sum_{q=1}^{N_q} \omega_q \left(\nabla u(\mathbf{F}_E(\hat{x}_q, \hat{y}_q)) - \mathbf{B}^{-t} \sum_{\hat{k}=1}^{N^E} u_{G_E(\hat{k})} \hat{\nabla} \hat{\phi}_{\hat{k}}(\hat{x}_q, \hat{y}_q) \right)^t \\ &\quad \left(\nabla u(\mathbf{F}_E(\hat{x}_q, \hat{y}_q)) - \mathbf{B}^{-t} \sum_{\hat{k}=1}^{N^E} u_{G_E(\hat{k})} \hat{\nabla} \hat{\phi}_{\hat{k}}(\hat{x}_q, \hat{y}_q) \right) \end{aligned} \quad (2.8)$$

$$\|u - u_h\|_{H^1(\Omega)}^2 = \|u - u_h\|_{L^2(\Omega)}^2 + \|\nabla u - \nabla u_h\|_{L^2(\Omega)}^2 \quad (2.9)$$

dove $G_E(\hat{k}) = k$ è la funzione che dato un indice locale \hat{k} , restituisce il corrispettivo indice globale k , con N^E , cioè il numero di gradi di libertà su ogni triangolo, che risulta essere 3 nel caso di elementi \mathbb{P}_1 .

Passiamo quindi agli ordini di convergenza.

Grazie alle stime (1.6, 1.7) deduciamo che l'ordine di convergenza teorico rispetto alla **RefiningArea** in questo caso sia $\frac{k+1}{2} = 1$ per la convergenza in norma L^2 e $\frac{k}{2} = \frac{1}{2}$ per la convergenza in norma H^1 . Per quanto concerne la norma L^∞ , la formula d'interesse è la prima in (1.8): pertanto, la presenza del fattore $|\log h| \sim |\log \sqrt{Area}|$ comporta che l'ordine di convergenza in questo caso sia leggermente minore di 1. Infine, dalla teoria sappiamo che il numero di condizionamento della matrice di rigidezza \mathbf{A} cresce nel seguente modo:

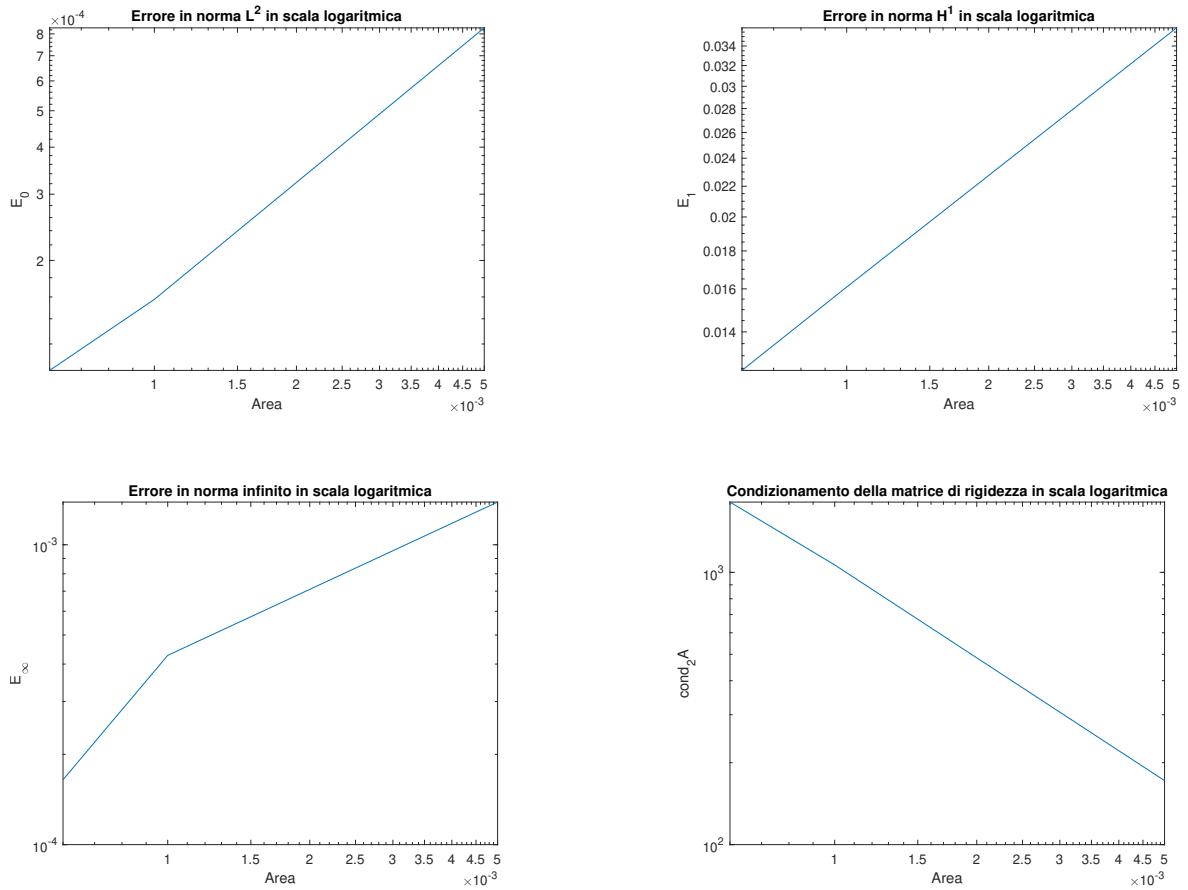
$$cond_2(\mathbf{A}) \sim h^{-2} \sim Area^{-1} \quad per \ h \rightarrow 0 \quad (2.10)$$

Eseguendo regressioni lineari con il metodo dei minimi quadrati, abbiamo ricavato i seguenti ordini di convergenza sperimentali relativi alla simulazione del paragrafo precedente:

2.3. CALCOLO DEGLI ERRORI E DEGLI ORDINI DI CONVERGENZA

- Ordine di convergenza in norma $L^2 = 0.9990$
- Ordine di convergenza in norma $H^1 = 0.5010$
- Ordine di convergenza in norma $L^\infty = 0.9394$
- Potenza della crescita del numero di condizionamento della matrice di rigidezza = -1.1165

Concludiamo questo paragrafo con i seguenti grafici: essi rappresentano in scala logaritmica (log-log) l'andamento degli errori e l'andamento del numero di condizionamento della matrice di rigidezza in funzione dei valori del parametro `RefiningArea`. Possiamo notare che secondo quanto previsto dalle stime (1.6), (1.7) e (2.10), dopo aver applicato il logaritmo ad ambo i membri in ciascuna di esse, i grafici delle norme L^2 e H^1 dell'errore di discretizzazione e il grafico del condizionamento di \mathbf{A} hanno un andamento lineare. Invece, l'andamento non lineare del grafico dell'errore in norma infinito è causato dalla presenza del fattore $|\log h|$, già menzionato in precedenza.



2.4 Codice Matlab

In questo paragrafo si trova il codice con cui abbiamo risolto il problema in analisi, insieme al calcolo degli errori. In particolare, abbiamo deciso di spezzare il codice in modo tale da evidenziare l'implementazione di ciascuna delle componenti del sistema algebrico.

In primis, i dati del problema

```

1 % DATI DEL PROBLEMA
2
3 global U_esatta csi eta omega geom gradU_esatto
4
5 U_esatta = @(x,y) x.*y-x.*y.^2+1;
6 gradU_esatto = @(x,y) [y-y.^2, x-2.*x.*y];
7
8 nu = @(x,y) x;
9 betha = @(x,y) [x,1];
10 gamma = @(x,y) y;
11 f = @(x,y) 2*x.^2 - x.*y + x + y.^2 - x.*y.^3;
12
13 gD = @(x,y) 1;
14 gN = @(x,y) y - y.^2;
15
16 aree = [0.005; 0.001; 0.0006];

```

In seguito, la costruzione delle matrici \mathbf{A}_D e \mathbf{A} e del vettore \mathbf{b} (contributo della forzante al termine noto) secondo le formule (2.1), (2.2), (2.3)

```

1 % COSTRUZIONE DI A,A_D E b
2
3 coordinates = geom.elements.coordinates;
4 triangles = geom.elements.triangles;
5 nPunti = size(coordinates,1);
6 nTriangoli = geom.nelements.nTriangles;
7 pivot = geom.pivot.pivot;
8 Ndof = max(pivot);
9 ND = abs(min(pivot));
10
11 A = zeros(Ndof, Ndof);
12 AD = zeros(Ndof, ND);
13 b = zeros(Ndof,1);
14
15 Hat_Phi = @(x,y) [x;y;1-x-y]; % Phi cappuccio
16 Hat_Phi_Gradient = [1,0,-1; 0,1,-1]; % Nabla cappuccio
17
18 Hat_Phi_Matrix = zeros(3,length(csi));
19
20 % Memorizzo la valutazione di Hat_Phi nei nodi di quadratura
21

```

```

22 for q=1:length(csi)
23     Hat_Phi_Matrix(:,q) = Hat_Phi(csi(q),eta(q));
24 end
25
26 %Creo delle strutture per memorizzare anche i delta dato che li devo
27 %usare piu volte
28
29 deltaX = zeros(3,nTriangoli);
30 deltaY = zeros(3,nTriangoli);
31 B = zeros(2,2,nTriangoli);
32 B_inv = zeros(2,2,nTriangoli);
33
34 for e = 1:nTriangoli
35     Area = geom.support.TInfo(e).Area;
36
37 %% Calcolo dei delta
38 deltaX(1,e) = coordinates(triangles(e,3),1) - ...
39     coordinates(triangles(e,2),1);
40 deltaX(2,e) = coordinates(triangles(e,1),1) - ...
41     coordinates(triangles(e,3),1);
42 deltaX(3,e) = coordinates(triangles(e,2),1) - ...
43     coordinates(triangles(e,1),1);
44
45 deltaY(1,e) = coordinates(triangles(e,2),2) - ...
46     coordinates(triangles(e,3),2);
47 deltaY(2,e) = coordinates(triangles(e,3),2) - ...
48     coordinates(triangles(e,1),2);
49 deltaY(3,e) = coordinates(triangles(e,1),2) - ...
50     coordinates(triangles(e,2),2);
51
52 B (:,:,e) = [deltaX(2,e) -deltaX(1,e); -deltaY(2,e) deltaY(1,e)];
53 B_inv (:,:,e) = (1/(2*Area)) * [deltaY(1,e), deltaX(1,e); ...
54     deltaY(2,e), deltaX(2,e)];
55
56 for j=1:3
57     jj = pivot(triangles(e,j));
58     if jj>0
59         for k=1:3
60             kk = pivot(triangles(e,k));
61             if kk>0
62                 for q=1:length(omega)
63                     F_e_vector = coordinates(triangles(e,3),:)';
64                     + B (:,:,e)*[csi(q);eta(q)];
65
66                     A(jj,kk) = A(jj,kk) + omega(q) * 2 * Area ...
67                     *(nu(F_e_vector(1),F_e_vector(2))*Hat_Phi_Gradient(:,k)'...
68                     * B_inv (:,:,e)*(B_inv (:,:,e))'*Hat_Phi_Gradient(:,j) ...
69                     + betha(F_e_vector(1),F_e_vector(2))*(B_inv (:,:,e))' ...
70                     * Hat_Phi_Gradient(:,k)*Hat_Phi_Matrix(j,q) ...
71                     +gamma(F_e_vector(1),F_e_vector(2))*Hat_Phi_Matrix(k,q) ...
72                     * Hat_Phi_Matrix(j,q) );
73                 end
74             else
75             end
76         end
77     end
78 end

```

```

68     for q=1:length(omega)
69         F_e_vector = coordinates(triangles(e,3),:) ' ...
70             + B(:,:,e)*[csi(q);eta(q)];
71
72         AD(jj,-kk) = AD(jj,-kk) + omega(q) * 2 * Area ...
73             *(nu(F_e_vector(1),F_e_vector(2))*Hat_Phi_Gradient(:,k)' ...
74             * B_inv(:,:,e)*(B_inv(:,:,e))'*Hat_Phi_Gradient(:,j) ...
75             + betha(F_e_vector(1),F_e_vector(2))*(B_inv(:,:,e))' ...
76             * Hat_Phi_Gradient(:,k)*Hat_Phi_Matrix(j,q) ...
77             +gamma(F_e_vector(1),F_e_vector(2))*Hat_Phi_Matrix(k,q) ...
78             * Hat_Phi_Matrix(j,q) ) ;
79     end
80 end
81
82 for q=1:length(omega)
83     F_e_vector = coordinates(triangles(e,3),:) ' ...
84         + B(:,:,e)*[csi(q);eta(q)];
85     b(jj) = b(jj) + omega(q) * 2 * Area ...
86         * f(F_e_vector(1),F_e_vector(2))*Hat_Phi_Matrix(j,q);
87 end
88 end
89 end
90 end
91 cond_number2 = cond(A,2);

```

Poi, la costruzione del vettore \mathbf{b}_N (contributo della condizione al contorno di Neumann al termine noto) secondo le formule (2.6)

```

1 %% COSTRUZIONE DI bN
2 bN = zeros(Ndof,1);
3 for e = 1:size(geom.pivot.Ne,1)
4     l = geom.pivot.Ne(e,1);
5     ib = geom.elements.borders(l,1);
6     ie = geom.elements.borders(l,2);
7     iib = geom.pivot.pivot(ib);
8     iie = geom.pivot.pivot(ie);
9
10    bordo = norm(coordinates(ib,:)-coordinates(ie,:));
11
12    if iib > 0
13        bN(iib) = bN(iib) ...
14            + bordo*(2 * gN(coordinates(ib,1),coordinates(ib,2)) ...
15            + gN(coordinates(ie,1),coordinates(ie,2)))/6;
16    end
17
18    if iie > 0
19        bN(iie) = bN(iie) ...
20            + bordo*(gN(coordinates(ib,1),coordinates(ib,2)) ...
21            + 2*gN(coordinates(ie,1),coordinates(ie,2)))/6;
22    end
23 end

```

Infine, la costruzione del vettore \mathbf{u}_D (soluzione sul bordo di Dirichlet, nota a priori), la risoluzione del sistema lineare e la costruzione della soluzione

```

1 %% COSTRUZIONE DI uD
2 uD = zeros(ND,1);
3
4 for i = 1:length(geom.pivot.Di)
5     x_D = coordinates(geom.pivot.Di(i,1),1);
6     y_D = coordinates(geom.pivot.Di(i,1),2);
7     uD(i) = gD(x_D, y_D);
8 end
9
10 % RISOLUZIONE DEL SISTEMA LINEARE E COSTRUZIONE DELLA SOLUZIONE
11 u0 = A\b(b-AD*uD+bN);
12 u = zeros(1,nPunti);
13
14 for j = 1:nPunti
15     jj = pivot(j);
16     if jj > 0
17         u(j) = u0(jj);
18     else
19         u(j) = uD(-jj);
20     end
21 end

```

Concludiamo il paragrafo con il calcolo degli errori, secondo le formule (2.7), (2.8), (2.9)

```

1 %% CALCOLO DEGLI ERRORI
2
3 normaL2 = 0 % Inizializzazione errori
4 normaL2grad = 0;
5 normaH1 = 0;
6
7 normaLInf = norm((U_esatta(coordinates(:,1),coordinates(:,2))-u'),inf);
8 % errore in norma L^inf
9
10 for e = 1:nTriangoli
11     Area = geom.support.TInfo(e).Area;
12     for j = 1:length(csi)
13         F_e_vector = coordinates(triangles(e,3),:) ' ...
14             + B(:,:,e)*[csi(j);eta(j)];
15         normaL2 = normaL2 + 2*Area*omega(j) ...
16             *(U_esatta(F_e_vector(1),F_e_vector(2)) ...
17                 - u(triangles(e,:))*Hat_Phi_Matrix(:,j))^2;
18         normaL2grad = normaL2grad + ...
19             2*Area*omega(j)*(gradU_esatto(F_e_vector(1),F_e_vector(2)) ...
20                 - u(triangles(e,:))*Hat_Phi_Gradient'*...
21                     (B_inv(:,:,e))*(gradU_esatto(F_e_vector(1),F_e_vector(2)) ...
22                         - u(triangles(e,:))*Hat_Phi_Gradient'*(B_inv(:,:,e)))');
23     end
24 end

```

```
24 % normaL2 e normal2grad appena calcolati sono rispettivamente il ...
    quadrato della norma L^2 e il quadrato della seminorma H^1 ...
    dell'errore
25
26 normaH1 = sqrt(normaL2 + normal2grad); % errore in norma H^1
27 normaL2 = sqrt(normaL2); % errore in norma L^2
```

Capitolo 3

Polinomi di grado 2

3.1 Introduzione teorica

Una possibile alternativa alla scelta dell'elemento finito di Courant è data dall'elemento lagrangiano $(E, \mathbb{P}_2(E), \mathcal{L}_E)$ per ciascun triangolo $E \in \mathcal{T}$, dove $\mathbb{P}_2(E)$ è lo spazio dei polinomi di secondo grado tale che $\dim \mathbb{P}_2(E) = 6$ e l'insieme dei gradi di libertà \mathcal{L}_E è dato dai valori di $v \in \mathbb{P}_2(E)$ sui vertici del triangolo e sui punti medio di ogni lato.

Richiamando la definizione (1.1) delle funzioni della base di Lagrange $\{\hat{N}_k(\hat{x})\}_{k=1}^3$ in $\mathbb{P}_1(\hat{E})$, possiamo costruire le funzioni della base di Lagrange in $\mathbb{P}_2(\hat{E})$ come segue

$$\begin{aligned}\hat{\phi}_1 &= 2\hat{N}_1 \left(\hat{N}_1 - \frac{1}{2} \right), & \hat{\phi}_4 &= \hat{N}_1 \hat{N}_3, \\ \hat{\phi}_2 &= 2\hat{N}_2 \left(\hat{N}_2 - \frac{1}{2} \right), & \hat{\phi}_5 &= \hat{N}_2 \hat{N}_1, \\ \hat{\phi}_3 &= 2\hat{N}_3 \left(\hat{N}_3 - \frac{1}{2} \right), & \hat{\phi}_6 &= 4\hat{N}_2 \hat{N}_3.\end{aligned}$$

In modo del tutto simile al caso in cui abbiamo scelto l'elemento di Courant, giungiamo al sistema lineare (1.4), dove \mathbf{u}_D è dato dalla (1.5) e introducendo una formula di quadratura sul triangolo \mathbf{A} , \mathbf{A}_D , \mathbf{f} assumono le forme espresse in (2.1, 2.2, 2.3) con le nuove funzioni di base.

In particolare per il contributo di \mathbf{b}_N al vettore \mathbf{f} procediamo in modo simile. Introduciamo per ogni $e \in \Gamma_N$ la parametrizzazione (2.4) e definiamo \mathbf{a}_{medium} il punto medio del segmento e tale che $\mathbf{a}_{medium} = \gamma(\frac{1}{2})$. Ponendo $g_m := g_N(\mathbf{a}_{medium})$ e mantenendo le definizioni di g_b , g_e , dalla (2.5) avremo che

$$g_N(\gamma(t)) = g_b 2(t-1)(t-0.5) + g_m (-4)t(t-1) + g_e 2t(t-0.5)$$

e considerando che

$$\begin{aligned}j \iff begin &\Rightarrow \phi_j^E(\gamma(t)) = 2(t-1)(t-0.5) \\ j \iff medium &\Rightarrow \phi_j^E(\gamma(t)) = -4t(t-1) \\ j \iff end &\Rightarrow \phi_j^E(\gamma(t)) = 2t(t-0.5)\end{aligned}$$

si ottengono i seguenti:

$$j \iff \begin{aligned} & \Rightarrow \int_0^1 g_N(\gamma(t)) \phi_j^E(\gamma(t)) |e| dt = |e| \frac{4g_b + 2g_m - g_e}{30} \end{aligned} \quad (3.1)$$

$$j \iff \text{medium} \Rightarrow \int_0^1 g_N(\gamma(t)) \phi_j^E(\gamma(t)) |e| dt = |e| \frac{-g_b + 2g_m + 4g_e}{30} \quad (3.2)$$

$$j \iff \text{end} \Rightarrow \int_0^1 g_N(\gamma(t)) \phi_j^E(\gamma(t)) |e| dt = |e| \frac{g_b + 8g_m + g_e}{15} \quad (3.3)$$

3.2 Simulazione

Consideriamo il medesimo problema modello del Paragrafo 2.2. Considerando gli stessi parametri del problema e la stessa soluzione, osserviamo che a parità di `RefiningArea` l'utilizzo dei polinomi di base in \mathbb{P}_2 permette la riduzione dell'errore assoluto di due ordini di grandezza rispetto al caso in cui si usi una base lineare \mathbb{P}_1 .

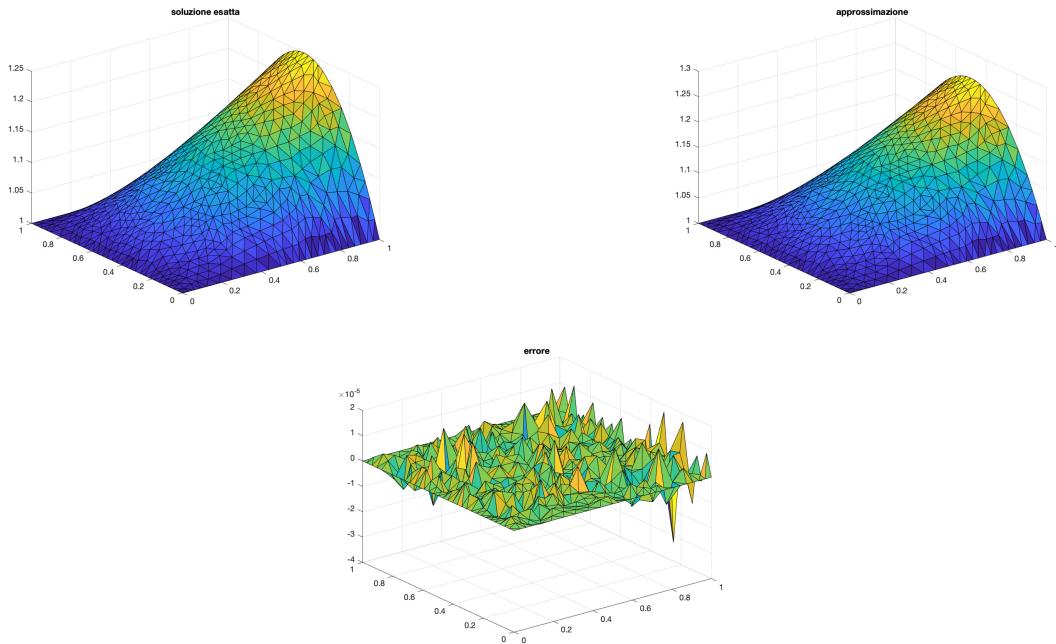


Figura 3.1: Simulazione con `RefiningArea` = 5e-3 e polinomi di base di grado 2

3.2. SIMULAZIONE

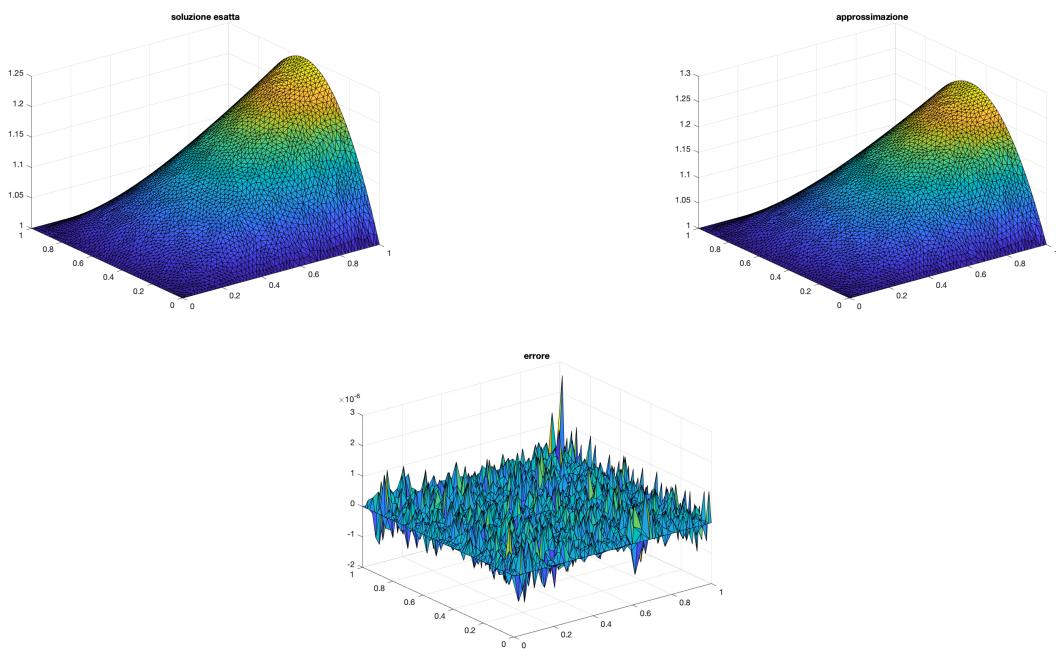


Figura 3.2: Simulazione con `RefiningArea` = $1\text{e-}3$ e polinomi di base di grado 2

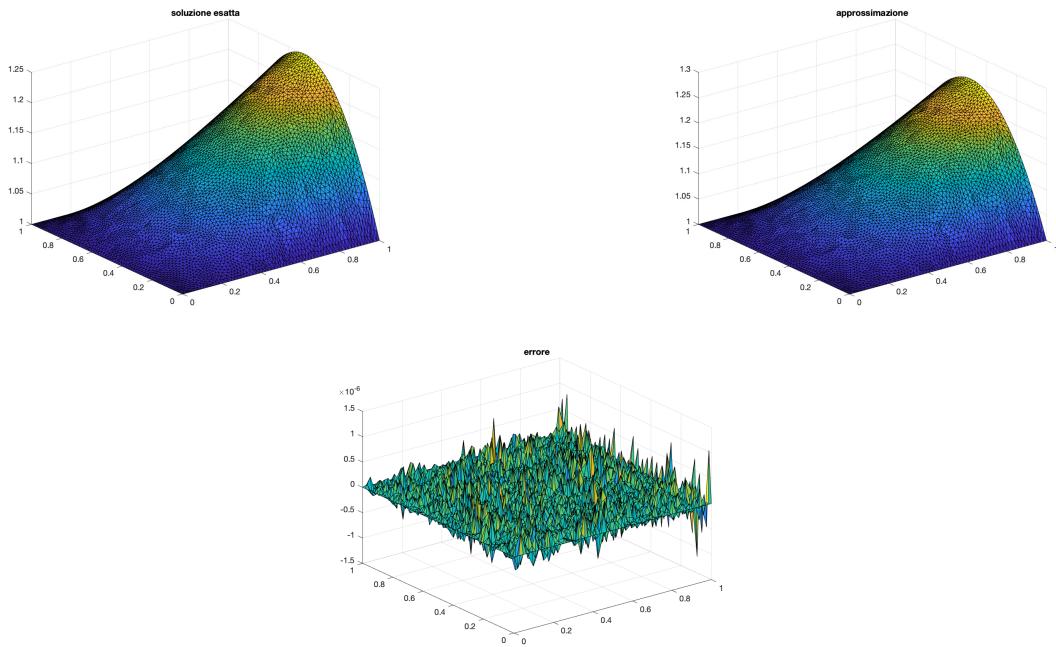


Figura 3.3: Simulazione con `RefiningArea` = $6\text{e-}4$ e polinomi di base di grado 2

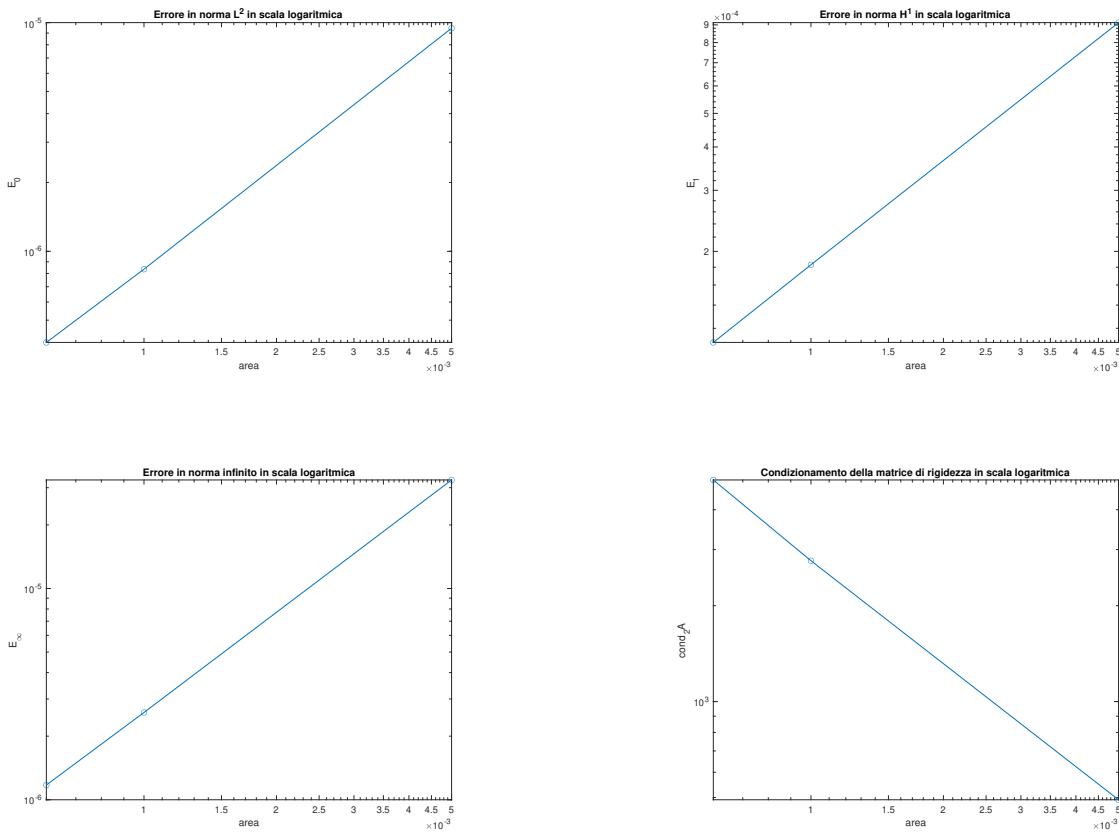
3.2. SIMULAZIONE

In effetti il miglioramento della qualità della soluzione è atteso grazie alle stime (1.6, 1.7, 1.8), attraverso cui deduciamo che l'ordine di convergenza teorico rispetto alla RefiningArea in questo caso sia $\frac{k+1}{2} = \frac{3}{2}$ per la convergenza in norma L^2 , $\frac{k}{2} = 1$ per la convergenza in norma H^1 e $\frac{k+1}{2} = \frac{3}{2}$ per la convergenza in norma L^∞ . Eseguendo regressione lineare con il metodo dei minimi quadrati possiamo calcolare gli ordini di convergenza sperimentali seguenti:

- ordine di convergenza in norma L^2 : 1.4957
- ordine di convergenza in norma H^1 : 1.0010
- ordine di convergenza in norma L^∞ : 1.5697
- Potenza della crescita del numero di condizionamento della matrice di rigidezza = -1.0859

che osserviamo essere vicini agli ordini attesi a meno della seconda cifra decimale e ciò corrobora l'esattezza dei metodi risolutivi applicati.

Concludiamo questo paragrafo rappresentando in scala logaritmica (log-log) l'andamento degli errori e l'andamento del numero di condizionamento della matrice di rigidezza in funzione dei valori del parametro RefiningArea e osserviamo che hanno tutti andamento lineare come previsto dalle stime (2.1, 2.2, 2.3)



3.3 Codice Matlab

In questa sezione riportiamo il codice per la risoluzione del problema stazionario usando funzioni polinomiali di base in \mathbb{P}_2 . Recuperando la struttura di base del codice descritto nella Sezione 2.4, è sufficiente apportare solo alcune modifiche per implementare il nuovo metodo risolutivo.

Mantenendo le stesse strutture di memorizzazione dei dati del problema, introduciamo le nuove funzioni di base con le rispettive derivate parziali e le valutiamo nei nodi di quadratura.

```

1 Hat_Phi = @(x,y) [
2     2*x.*(x-0.5);
3     2*y.*(y-0.5);
4     2*(1-x-y).* (0.5-x-y);
5     4*x.* (1-x-y); 4*x.*y;
6     4*y.* (1-x-y) ]; % Phi cappuccio
7
8 Hat_Phi_Gradient = @(x,y) [
9     4*x-1, 0, -3+4*x+4*y, 4-8*x-4*y, 4*y, -4*y;
10    0, 4*y-1, -3+4*x+4*y, -4*x, 4*x, 4-8*y-4*x ]; % Nabla cappuccio
11
12 Hat_Phi_Matrix = zeros(6,length(omega));
13 % Memorizzo la valutazione di Hat_Phi nei nodi di quadratura
14 Hat_Phi_Gradient_Tensor = zeros(2,6,length(omega));
15 % La riga rappresenta la variabile x,y rispetto a cui si deriva ...
16 % nel gradiente
16 % la colonna l'i-esima phi_cappuccio
17 % la terza dimensione il nodo di quadratura su cui si vuole ...
18 % valutare il gradiente
18 for q=1:length(omega)
19     Hat_Phi_Matrix(:,q) = Hat_Phi(csi(q),eta(q));
20     Hat_Phi_Gradient_Tensor(:,:,q) = Hat_Phi_Gradient(csi(q),eta(q));
21 end
22
23 Hat_Phi_xx = [ 4, 0, 4, -8, 0, 0 ];
24 Hat_Phi_yy = [ 0, 4, 4, 0, 0, -8 ];
25 Hat_Phi_xy = [ 0, 0, 4, -4, 4, -4 ];

```

Quindi in modo simile al caso precedente costruiamo le matrici \mathbf{A} , \mathbf{A}_D e il vettore \mathbf{b} , con le opportune sostituzioni delle funzioni di base.

```

1 for e = 1:nTriangoli
2
3     % [ ... ] si faccia riferimento al codice nel caso dei P1
4
5     for j=1:6
6         jj = pivot(triangles(e,j));
7         if jj>0
8             for k=1:6
9                 kk = pivot(triangles(e,k));

```

```

10      if kk>0
11          for q=1:length(omega)
12              F_e_vector = coordinates(triangles(e,3),:) ' ...
13                  + B(:,:,e)*[csi(q);eta(q)];
14              A(jj,kk) = A(jj,kk) + omega(q) * 2*Area ...
15                  * ( nu(F_e_vector(1),F_e_vector(2)) ...
16                      *Hat_Phi_Gradient_Tensor(:,k,q)' ...
17                      *B_inv(:,:,e)*(B_inv(:,:,e))' ...
18                      *Hat_Phi_Gradient_Tensor(:,j,q) ...
19                      + betha(F_e_vector(1),F_e_vector(2))* (B_inv(:,:,e))' ...
20                      *Hat_Phi_Gradient_Tensor(:,k,q)*Hat_Phi_Matrix(j,q) ...
21                      + gamma(F_e_vector(1),F_e_vector(2))* ...
22                      Hat_Phi_Matrix(k,q)*Hat_Phi_Matrix(j,q) ) ;
23      end
24  else
25      for q=1:length(omega)
26          F_e_vector = coordinates(triangles(e,3),:) ' ...
27              + B(:,:,e)*[csi(q);eta(q)];
28          AD(jj,-kk) = AD(jj,-kk) + omega(q) * 2*Area ...
29              * ( nu(F_e_vector(1),F_e_vector(2)) ...
30                  *Hat_Phi_Gradient_Tensor(:,k,q)'*B_inv(:,:,e) ...
31                  *(B_inv(:,:,e))'*Hat_Phi_Gradient_Tensor(:,j,q) ...
32                  + betha(F_e_vector(1),F_e_vector(2))* (B_inv(:,:,e))' ...
33                  *Hat_Phi_Gradient_Tensor(:,k,q)*Hat_Phi_Matrix(j,q) ...
34                  + gamma(F_e_vector(1),F_e_vector(2)) ...
35                  *Hat_Phi_Matrix(k,q)*Hat_Phi_Matrix(j,q) ) ;
36      end
37  end
38 end
39 for q=1:length(omega)
40     F_e_vector = coordinates(triangles(e,3),:) ' ...
41         + B(:,:,e)*[csi(q);eta(q)];
42     b(jj) = b(jj) + omega(q) * 2*Area ...
43         * f(F_e_vector(1),F_e_vector(2))*Hat_Phi_Matrix(j,q);
44 end
45 end
46 end
47 end

```

La costruzione del vettore \mathbf{u}_D rimane del tutto invariata, mentre il termine \mathbf{b}_N viene calcolato come riportato in (3.1, 3.2, 3.3)

```

1 %% Costruzione di bN
2 bN = zeros(Ndof,1);
3 for e = 1:size(geom.pivot.Ne,1)
4     l = geom.pivot.Ne(e,1);
5     ib = geom.elements.borders(l,1);
6     ie = geom.elements.borders(l,2);
7     im = geom.elements.borders(l,5);
8
9     iib = pivot(ib);
10    iie = pivot(ie);

```

```

11 iim = pivot(im);
12
13 bordo = norm(coordinates(ib,:) - coordinates(ie,:));
14
15 if iib > 0
16     bN(iib) = bN(iib) + ...
17         bordo*( 4*gN(coordinates(ib,1),coordinates(ib,2)) ...
18             - gN(coordinates(ie,1),coordinates(ie,2)) ...
19                 + 2*gN(coordinates(im,1),coordinates(im,2)) )/30;
20 end
21 if iie > 0
22     bN(iie) = bN(iie) + ...
23         bordo*(-gN(coordinates(ib,1),coordinates(ib,2)) ...
24             + 4*gN(coordinates(ie,1),coordinates(ie,2)) ...
25                 + 2*gN(coordinates(im,1),coordinates(im,2)) )/30;
26 end
27 if iim > 0
28     bN(iim) = bN(iim) + ...
29         bordo*(gN(coordinates(ib,1),coordinates(ib,2)) ...
30             + gN(coordinates(ie,1),coordinates(ie,2)) ...
31                 + 8*gN(geom.elements.coordinates(im,1),coordinates(im,2)) )/15;
32 end
33 end

```

Noti tutti i termini del sistema lineare (1.4), la sua risoluzione e la costruzione della soluzione è analoga al caso \mathbb{P}_1 . Per quanto riguarda il calcolo dell'errore è sufficiente procedere in modo simile alla costruzione della matrice \mathbf{A} con le opportune sostituzioni delle nuove funzioni di base.

```

1 normaL2 = 0;
2 normaL2grad = 0;
3 normaH1 = 0
4
5 for e = 1:nTriangoli
6     Area = geom.support.TInfo(e).Area;
7     Fe = @(x,y) coordinates(triangles(e,3),:)'+B(:,:,e)*[x; y];
8
9     for q = 1:length(csi)
10        inp = Fe(csi(q),eta(q));
11        normaL2 = normaL2 + 2*Area*omega(q) * ( U_esatta(inp(1),inp(2)) ...
12            - u(triangles(e,:))*Hat_Phi_Matrix(:,q) )^2;
13        normaL2grad = normaL2grad + 2*Area*omega(q) ...
14            * ( gradU_esatto(inp(1),inp(2)) - u(triangles(e,:)) ...
15                *Hat_Phi_Gradient_Tensor(:,:,q)'*(B_inv(:,:,e)) ) ...
16                * ( gradU_esatto(inp(1),inp(2)) - u(triangles(e,:)) ...
17                    *Hat_Phi_Gradient_Tensor(:,:,q)'*(B_inv(:,:,e)) );
18    end
19 end
20
21 normaL2 = sqrt(normaL2);
22 normaL2grad = sqrt(normaL2grad);
23 normaH1 = sqrt(normaL2^2 + normaL2grad^2)

```

Capitolo 4

Problema di diffusion-convezione

4.1 Problema di diffusion-convezione

In questa sezione discuteremo di un problema di stabilità della soluzione che si può osservare nel problema di convezione-diffusione. Tale problematica risulta piuttosto comune poiché si presenta quando il coefficiente di diffusione risulta di molti ordini di grandezza inferiore rispetto al coefficiente di convezione, ed è noto che in generale i processi di diffusione (almeno quelli fickiani) risultino essere molto lenti. In questi casi è quindi sufficiente che sia presente un forte “vento” convettivo perché insorga questa instabilità.

4.2 Problema in 1D

Consideriamo il seguente problema di convezione-diffusione in 1D:

$$\begin{cases} -\nu \frac{\partial^2 u}{\partial x^2} + \beta \frac{\partial u}{\partial x} = 0 & \text{in } (0, L) \\ u(0) = u_0 \\ u(L) = u_L \end{cases} \quad (4.1)$$

Facendo una discretizzazione ad elementi finiti lineari del problema su una partizione uniforme dell’intervallo $(0, L)$ in N intervalli, si ottiene il seguente problema discreto:

$$\begin{aligned} -\nu \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + \beta \frac{u_{i+1} - u_{i-1}}{2h} &= 0, \quad i = 1, \dots, N-1 \\ u_0 &= u_0 \\ u_N &= u_L \end{aligned} \quad (4.2)$$

Introduciamo inoltre la seguente quantità:

$$Pe_E = \frac{\beta h}{2\nu} \quad (4.3)$$

che chiamiamo “numero di Peclet di griglia”.

La soluzione esatta del problema discreto si dimostra essere

$$u_i = u_0 + (u_L - u_0) \frac{\xi^i - 1}{\xi^N - 1}$$

dove

$$\xi = \frac{1 + Pe_E}{1 - Pe_E}$$

Quindi, notiamo che se $Pe_E > 1 \Rightarrow \xi < -1$. Questo comporta che $\xi^i > 0$ per valori pari di i e $\xi^i < 0$ per valori dispari di i . Inoltre, a prescindere dal segno, il valore assoluto aumenta all'aumentare di i . Ne segue che la soluzione del problema discreto ha un comportamento oscillatorio molto marcato che compromette la bontà dell'approssimazione.

Per ovviare a questo problema, che sorge solamente quando $Pe_E > 1$, dobbiamo introdurre il concetto di “viscosità artificiale”, cioè una perturbazione del coefficiente di diffusione “naturale” ν tale che:

$$\frac{\beta h}{\nu + \tau} = 1 \quad (4.4)$$

dove τ è proprio la viscosità artificiale. Questo tipo di stabilizzazione non compromette la convergenza del metodo in 1D.

Si può anche dimostrare che esiste un valore ottimo della viscosità artificiale che permette alla soluzione approssimata di ottenere nei nodi equispaziati lo stesso valore della soluzione esatta:

$$\tau^{opt} = \frac{h\beta}{2 \tanh Pe_E} - \nu \quad (4.5)$$

Nel seguente paragrafo approfondiremo lo stesso concetto per quanto riguarda gli elementi finiti in 2D.

4.3 Problema in 2D

L'idea della stabilizzazione in 2D ha bisogno di un'accortezza in più: infatti la viscosità artificiale, che si deve aggiungere, dovrebbe agire solamente nella direzione del “vento” convettivo e non ortogonalmente ad esso. Tenendo conto di ciò il termine da aggiungere alla diffusione è

$$-\int_{\Omega} \nabla \cdot (\boldsymbol{\tau} \nabla u) = \int_{\Omega} \frac{\tau}{\beta^2} \frac{\partial u}{\partial \beta} \frac{\partial v}{\partial \beta} d\Omega - \int_{\partial\Omega} (\boldsymbol{\tau} \nabla u) \cdot \hat{\mathbf{n}} v d\Omega \quad (4.6)$$

dove $\beta = \beta \hat{\mathbf{t}}$ è la componente tangenziale della convezione e i vettori $\hat{\mathbf{t}}$ e $\hat{\mathbf{n}}$ sono i versori tangente e normale alla direzione del “vento” convettivo e τ è la viscosità artificiale. Inoltre per mantenere intatto l’ordine di convergenza del metodo di Galerkin è necessario operare delle ulteriori modifiche che coinvolgono tutti gli altri termini dell’equazione. In questo modo il problema variazionale deve essere riscritto nel modo seguente:

$$\begin{aligned} \int_{\Omega} \nu \nabla u_h \nabla \phi_j d\Omega + \int_{\Omega} \boldsymbol{\beta} \cdot \nabla u_h \phi_j d\Omega + \sum_{E \in \mathcal{T}} \tau_E \int_E (\nabla \cdot (\nu \nabla u_h) + \boldsymbol{\beta}^t \nabla u_h) \cdot \boldsymbol{\beta}^t \nabla \phi_j d\Omega \\ = \int_{\Omega} f \phi_j d\Omega + \sum_{E \in \mathcal{T}} \int_E f \boldsymbol{\beta}^t \nabla \phi_j d\Omega, \quad \forall j = 1, \dots, N_{dof} \end{aligned} \quad (4.7)$$

dove la scelta del parametro τ_E dipende dal numero di Peclét del triangolo in cui lo si calcola secondo le formule:

$$Pe_E = m_k \frac{\|\boldsymbol{\beta}\|_E h_E}{2\nu} \quad (4.8)$$

$$\tau_E = \begin{cases} m_k \frac{h_E^2}{4\nu}, & \text{se } 0 \leq Pe_E < 1 \\ \frac{h_E}{2\|\boldsymbol{\beta}\|_E} & \text{se } Pe_E \geq 1 \end{cases} \quad (4.9)$$

dove m_k è un valore che dipende dal grado dei polinomi della base. Noi useremo i polinomi di grado 2 per cui $m_2 = 24$. Significa che, per ogni triangolo, dovremo calcolare Pe_E e considerare il tipo di correzione da effettuare sulla matrice di rigidezza. Inoltre per calcolare Pe_E considereremo $|E| = h_E^2$.

Esempio 1. Abbiamo preso come modello un problema di convezione-diffusione con i seguenti dati in modo da avere $Pe_E > 1$

- $u(x, y) = 16xy(1 - x)(1 - y)$ (Soluzione esatta);
- $\nu(x, y) = 3 \cdot 10^{-8}$ (Coefficiente di diffusione);
- $\boldsymbol{\beta}(x, y) = (1, 1)$ (Coefficiente di convezione);
- $\gamma(x, y) = 0$ (Coefficiente di reazione);
- $f(x, y) = -96 \cdot 10^{-8}(-y + y^2 - x + x^2) + 16(-4xy + y - y^2 + x - x^2 + 2xy^2 + 2x^2y)$ (Termine forzante);
- $g_D(x, y) = 0$ (Condizione al bordo di Dirichlet) su $\partial\Omega$;
- $\Omega = [0, 1] \times [0, 1]$

Come si può notare, la differenza in ordine di grandezza tra le componenti di β e ν è molto grande. Bisogna però tenere in considerazione anche l'area del triangolo sul quale va verificato il tipo di stabilizzazione da fare, ricordando che la norma scritta in (4.8) è una norma funzionale. Prima di applicare la stabilizzazione abbiamo deciso di verificare qualitativamente l'instabilità del metodo base, provando ad approssimare la soluzione per alcuni valori di area:

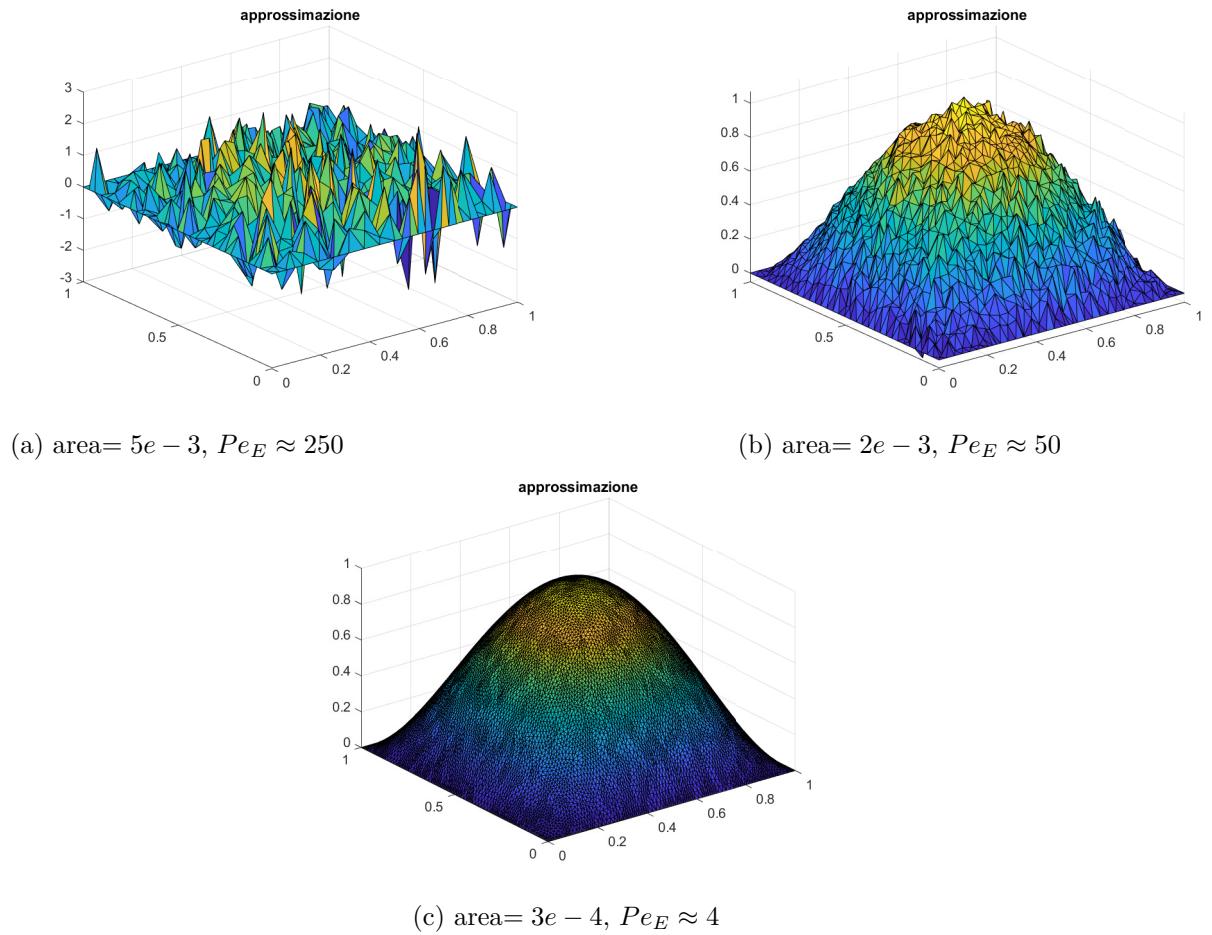


Figura 4.1: Grafici dell'approssimazione ottenuta mediante polinomi di base di grado 2 relativi al problema di sopra per diversi valori di area.

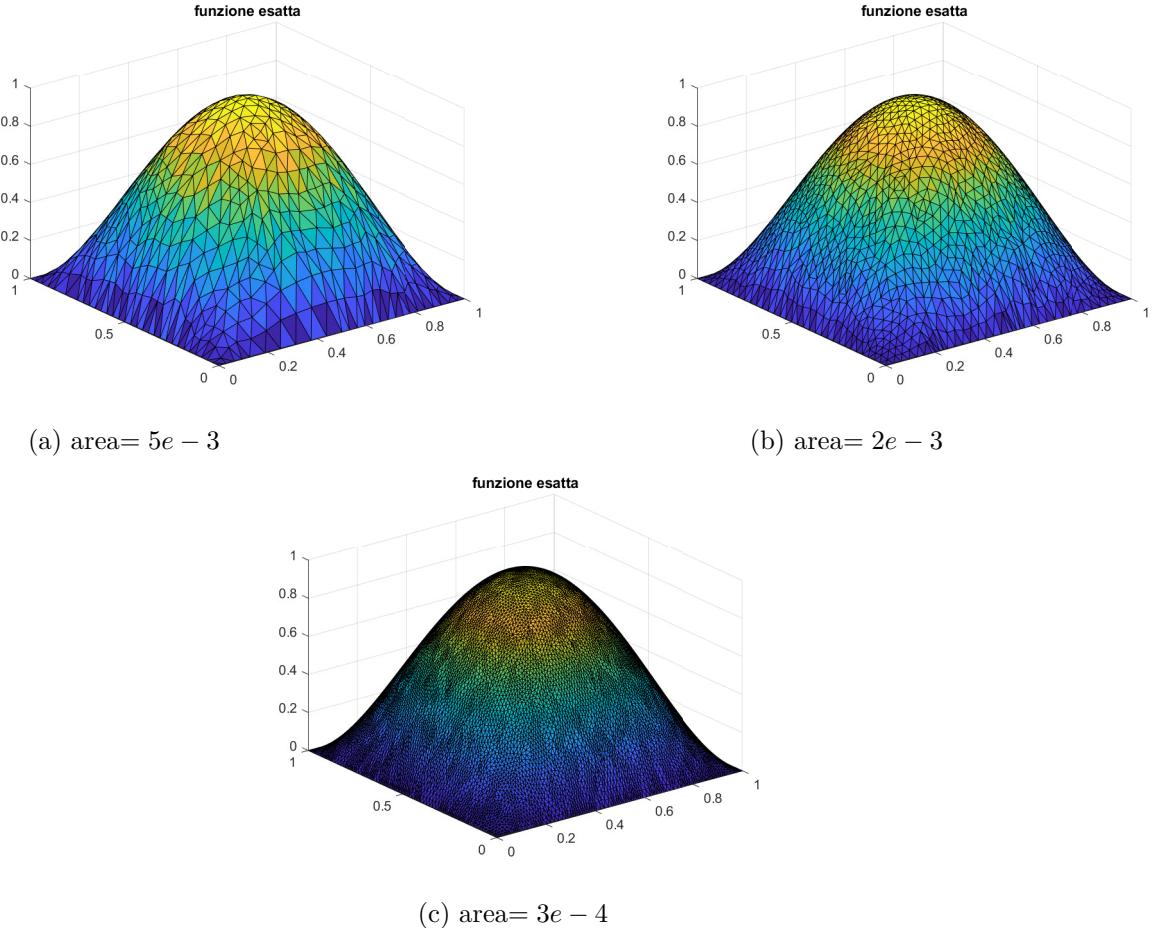


Figura 4.2: Grafici della soluzione esatta relativa al problema di sopra per gli stessi valori di area delle approssimazioni mostrate sopra

A livello qualitativo già si riesce a osservare la grande instabilità data da un Pe_E molto alto e il fatto che diminuendo l'area si possa correggere “naturalmente” l'instabilità. Tuttavia ciò non è sempre praticabile poiché per aree molto basse il costo computazionale diventa eccessivamente grande. Di seguito i risultati che abbiamo ottenuto facendo la correzione che abbiamo scritto in (4.7), usando gli stessi valori di area:

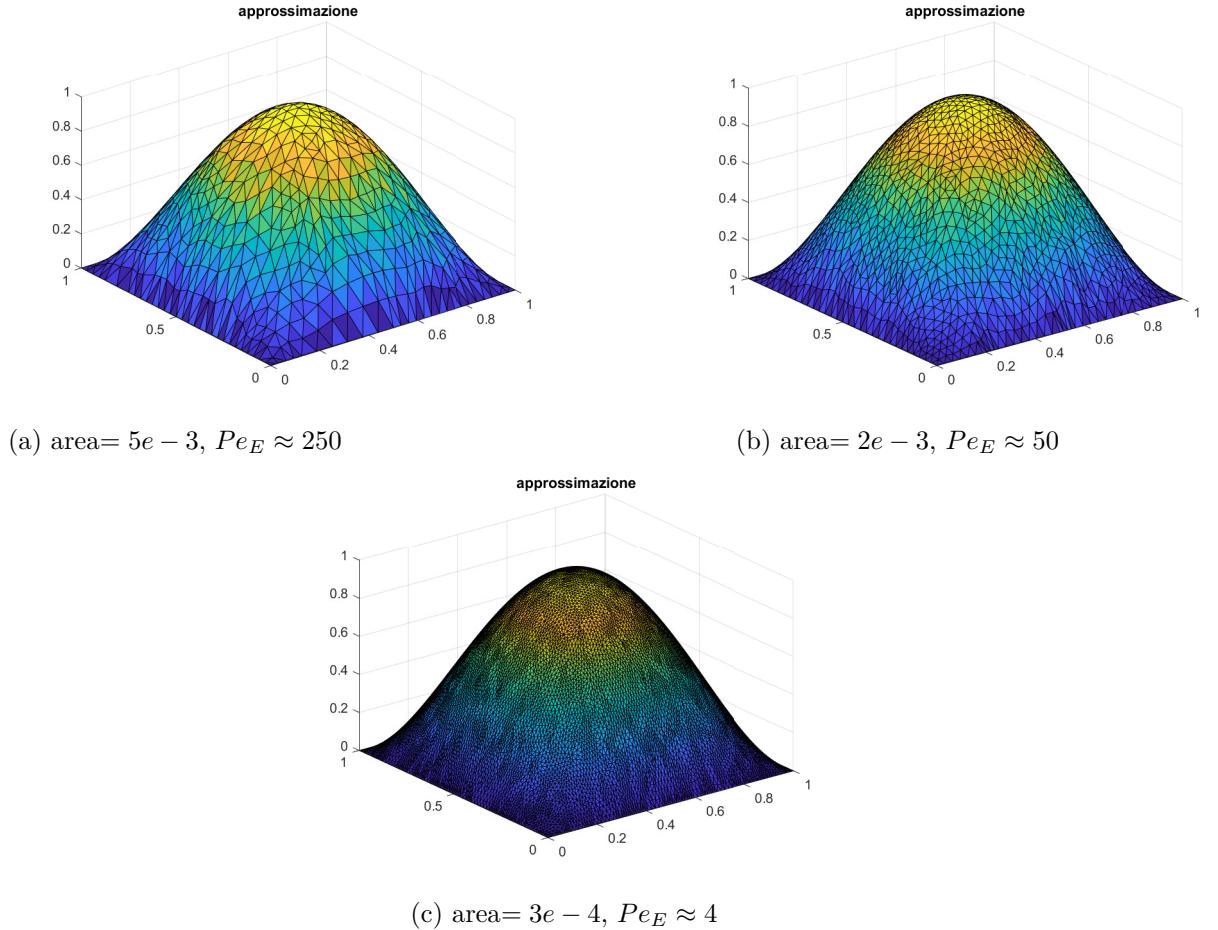


Figura 4.3: Grafici dell'approssimazione ottenuta mediante polinomi di base di grado 2 relativi al problema di sopra per diversi valori di area, applicando la correzione in (4.7).

Gli ordini di convergenza, ottenuti dal problema stabilizzato, vengono rispettati: infatti, tenendo conto che abbiamo usato le aree, abbiamo ottenuto:

- ordine di convergenza in norma L^2 : 1.4
- ordine di convergenza in norma H^1 : 0.96
- ordine di convergenza in norma L^∞ : 1.23

4.4 Dettagli di implementazione

Riguardo all'implementazione della formula (4.7) abbiamo riscontrato un problema che nelle altre implementazioni non si è mai verificato, ovvero il cambio di coordinate dell'operatore laplaciano. Considerando che il cambio di coordinate dell'operatore nabla può essere scritto come segue:

$$\nabla = \mathbf{B}^{-t} \hat{\nabla} = \mathbf{B}^{-t} \left(\frac{\partial}{\partial \hat{x}}, \frac{\partial}{\partial \hat{y}} \right)^t \quad (4.10)$$

possiamo riscrivere anche il laplaciano ricordandoci che può essere scritto come la divergenza del gradiente:

$$\Delta = \nabla \cdot \nabla = \mathbf{B}^{-t} \hat{\nabla} \cdot \mathbf{B}^{-t} \hat{\nabla} = \hat{\nabla}^t \mathbf{B}^{-1} \mathbf{B}^{-t} \hat{\nabla} \quad (4.11)$$

definiamo ora la matrice $\tilde{\mathbf{B}} := \mathbf{B}^{-1} \mathbf{B}^{-t}$ e le sue entrate come $\tilde{\mathbf{B}} = (b_{jk})$ $1 \leq j, k \leq 2$. Esplicitando l'operatore $\hat{\nabla}$ e svolgendo le moltiplicazioni matrice-vettore:

$$\begin{aligned} \Delta &= \left(\frac{\partial}{\partial \hat{x}}, \frac{\partial}{\partial \hat{y}} \right) \tilde{\mathbf{B}} \left(\frac{\partial}{\partial \hat{x}}, \frac{\partial}{\partial \hat{y}} \right)^t = \\ &= b_{11} \frac{\partial^2}{\partial \hat{x}^2} + (b_{12} + b_{21}) \frac{\partial^2}{\partial \hat{x} \partial \hat{y}} + b_{22} \frac{\partial^2}{\partial \hat{y}^2} \end{aligned} \quad (4.12)$$

In questo modo siamo riusciti a scrivere l'operatore laplaciano e abbiamo implementato proprio questa formula nel codice Matlab

4.5 Codice Matlab

Di seguito presentiamo il codice che abbiamo implementato di cui mostriamo solamente la parte interessata in cui costruiamo la matrice di rigidezza e il termine noto con la correzione. In particolare, per questioni di comodità, abbiamo diviso la parte “standard” dalla correzione in modo da poterla aggiungere a piacimento tramite l'utilizzo di una variabile booleana. Prima di tutto abbiamo calcolato il numero di Peclet e la viscosità artificiale

```

1   m = 1/24;
2   CG = geom.support.TInfo(e).CG;
3   normBeta = norm(betha(CG(1), CG(2)));
4   Area = geom.support.TInfo(e).Area;
5   Peclet = (m/2)*sqrt(Area^3)*normBeta/eps(CG(1), CG(2));
6
7   if Peclet < 1 && Peclet >= 0
8       viscosita_artificiale = (m/4)*Area/eps(CG(1), CG(2));
9   end
10  if Peclet >= 1
11      viscosita_artificiale = 1/(2*sqrt(Area)*normBeta);
12  end

```

e dopo aver calcolato la matrice B , calcoliamo le matrici del sistema e il termine noto, con l'aggiunta del termine di correzione nel caso che la variabile “upwind” sia uguale a 1.

```

1 %Stesso codice dei P2
2 A(jj,kk) = A(jj,kk)+...

```

```

3 ...
4
5 if upwind == 1
6 tmp = B_inv(:,:,e)*(B_inv(:,:,e))';
7     A(jj,kk) = A(jj,kk)...
8     +viscosita_artificiale*omega(q)*2*Area*...
9     (eps(CG(1),CG(2))*tmp(1,1)*Hat_Phi_xx(k)+...
10    (tmp(1,2)+tmp(2,1))*Hat_Phi_xy(k)+tmp(2,2)*Hat_Phi_yy(k))*...
11    (betha(F_e_vector(1),F_e_vector(2))*...
12    (B_inv(:,:,e))'*Hat_Phi_Gradient_Tensor(:,j,q))...
13    +betha(F_e_vector(1),F_e_vector(2))*...
14    (B_inv(:,:,e))'*Hat_Phi_Gradient_Tensor(:,k,q)*...
15    *betha(F_e_vector(1),F_e_vector(2))*(B_inv(:,:,e))'*...
16    Hat_Phi_Gradient_Tensor(:,j,q));
17 end

```

La correzione della matrice \mathbf{A}_D è analoga alla precedente, mentre la correzione del termine noto è

```

1 %Stesso codice dei P2
2 b(jj) = b(jj) + ...
3
4 if upwind == 1
5     b(jj) = b(jj) + ...
6     omega(q)*2*Area*viscosita_artificiale*...
7     f(F_e_vector(1),F_e_vector(2))*...
8     betha(F_e_vector(1),F_e_vector(2))*...
9     *(B_inv(:,:,e))'*Hat_Phi_Gradient_Tensor(:,j,q);
10 end

```

Capitolo 5

Problema di diffusion-reazione

5.1 Problema di diffusion-reazione

Consideriamo la seguente quantità adimensionale

$$Se_E = \frac{\gamma h^2}{6\nu} \quad (5.1)$$

relativa a un triangolo della mesh, dove h è l'ampiezza dell'intervallo di cui si vuole calcolare l'indice. Come nel caso del problema di diffusion-convezione ci aspettiamo un comportamento oscillatorio nel caso in cui $Se_E > 1$. In particolare tali oscillazioni si dovrebbero osservare vicino al bordo di Dirichlet. Il motivo per cui compaiono è che, nel metodo degli elementi finiti, il termine di reazione coinvolge tutti i nodi vicini ad esso. Nel metodo delle differenze finite tale coinvolgimento non è presente, ed infatti non presenta comportamenti oscillatori per $Se_E > 1$. Per vederlo più chiaramente consideriamo il seguente problema monodimensionale:

$$\begin{cases} -\nu \frac{\partial^2 u}{\partial x^2} + \gamma u = 0 & \text{in } [0, L] \\ u(0) = u_0 \\ u(1) = u_L \end{cases} \quad (5.2)$$

Facendo una discretizzazione ad elementi finiti lineari del problema usando nodi equispaziati si ottiene

$$\begin{aligned} -\nu \frac{u_{i+1} - 2u_i + u_{i-1}}{h} + \frac{\gamma h}{6}(u_{i+1} + 4u_i + u_{i-1}) &= 0, \quad i = 1, \dots, N-1 \\ u_0 &= u_0 \\ u_N &= u_L \end{aligned} \quad (5.3)$$

dove è evidente che per ogni indice i sono coinvolte tre incognite diverse legate alla discettizzazione del termine di reazione. Con il metodo delle differenze finite si ottiene invece

$$\begin{aligned} -\nu \frac{u_{i+1} - 2u_i + u_{i-1}}{h} + \gamma h u_i &= 0, \quad i = 1, \dots, N-1 \\ u_0 &= u_0 \\ u_N &= u_L \end{aligned} \tag{5.4}$$

in cui è coinvolta solamente un'incognita legata al termine di reazione per ogni valore dell'indice i . Da qui segue l'intuizione di “accoppare” i termini della reazione tutti sulla diagonale della matrice di rigidezza, anche nel caso bidimensionale. Tale procedimento si chiama “mass lumping” ed è una correzione del primo ordine che si può apportare nel caso si usino polinomi di base di grado 1.

Esempio 2. Prendiamo in considerazioni i seguenti dati relativi al problema di diffusione-reazione stazionario

- $u(x, y) = 16xy(1-x)(1-y)e^{5x}$ (Soluzione esatta);
- $\nu(x, y) = 1 \cdot 10^{-8}$ (Coefficiente di diffusione);
- $\beta(x, y) = (0,0)$ (Coefficiente di convezione);
- $\gamma(x, y) = 3$ (Coefficiente di reazione);
- $f(x, y) = -\nu \Delta u(x, y) + \gamma(x, y)u(x, y)$ (Termine forzante);
- $g_D(x, y) = 0$ (Condizione al bordo di Dirichlet) su $\partial\Omega$;
- $\Omega = [0,1] \times [0,1]$

Abbiamo approssimato con il metodo degli elementi finiti con diversi valori di area, di seguito i risultati che abbiamo ottenuto qualitativamente, prima i grafici della funzione esatta

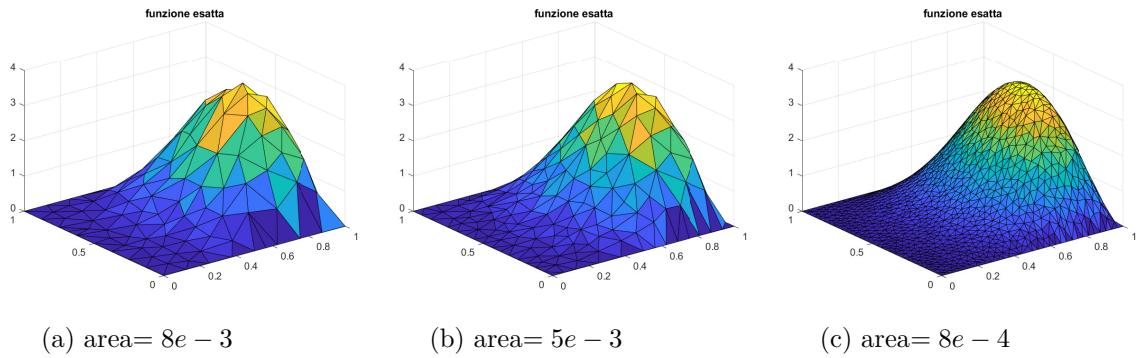


Figura 5.1: Grafici della soluzione esatta del problema di sopra per tre valori di area

e poi i grafici degli errori commessi con e senza “mass lumping”

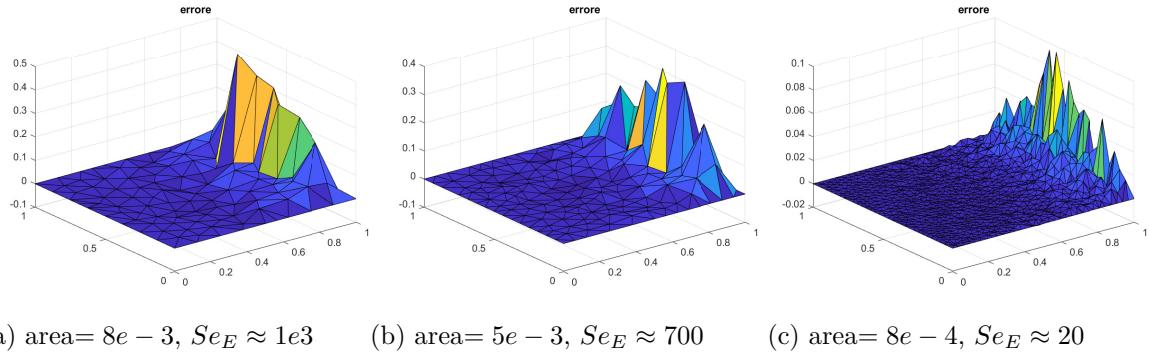


Figura 5.2: Grafici degli errori $\text{err} = \mathbf{u} - \mathbf{u}_h$ relativi al problema di sopra per tre diversi valori di area ottenuti senza effettuare il “mass lumping”

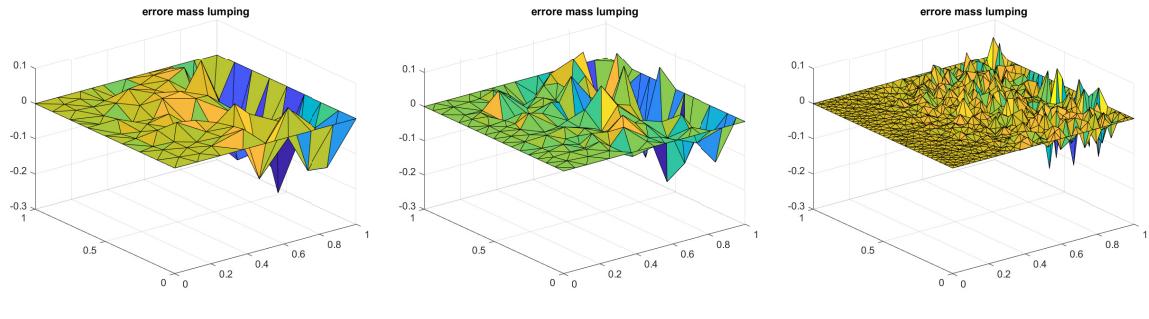


Figura 5.3: Grafici degli errori $\text{err} = \mathbf{u} - \mathbf{u}_h$ relativi al problema di sopra per tre diversi valori di area ottenuti effettuando il “mass lumping”

Abbiamo deciso di non mostrare i grafici delle approssimazioni poiché qualitativamente non è possibile fare delle osservazioni significative.

Dai grafici dell'errore si vede come il “mass lumping” risulti utile per valori di area sufficientemente grandi, poiché per il primo e per il secondo valore di area l'errore diminuisce. Per il terzo valore invece l'errore aumenta di circa tre volte. Da queste osservazioni possiamo concludere che, in generale, il “mass lumping” risulti utile nei casi in cui il valore dell'area sia abbastanza grande. Infatti per aree piccole ($Se_E \approx 1$) la correzione si rende meno necessaria.

Operando il “mass lumping” gli ordini di convergenza vengono inoltre errati rispetto ai risultati teorici.

5.2 Codice Matlab

Di seguito presentiamo il codice Matlab con cui abbiamo effettuato il “mass lumping”, analogamente al caso del problema di convezione-diffusione abbiamo impostato una variabile booleana chiamata “reazione” per decidere a piacimento se effettuare o meno la correzione

```

1 %Stesso codice dei P1
2 ...
3 if reazione == 1
4 A(jj,kk) = A(jj,kk) + omega(q) * 2*Area * ( ...
5     eps(F_e_vector(1),F_e_vector(2))*...
6         Hat_Phi_Gradient(:,k)'*...
7             B_inv(:,:,e)*(B_inv(:,:,e))'*Hat_Phi_Gradient(:,j)...
8             + betha(F_e_vector(1),F_e_vector(2))*(B_inv(:,:,e))'*...
9                 Hat_Phi_Gradient(:,k)*Hat_Phi_Matrix(j,q));
10    A(jj,jj) = A(jj,jj) + ...
11        omega(q)*2*Area*gamma(F_e_vector(1),F_e_vector(2))*...
12            Hat_Phi_Matrix(k,q)*Hat_Phi_Matrix(j,q);
13 ...
14 if reazione == 1
15 AD(jj,-kk) = AD(jj,-kk) + omega(q) * 2*Area * ( ...
16     eps(F_e_vector(1),F_e_vector(2))*...
17         Hat_Phi_Gradient(:,k)'*...
18             B_inv(:,:,e)*(B_inv(:,:,e))'*Hat_Phi_Gradient(:,j)...
19             + betha(F_e_vector(1),F_e_vector(2))*(B_inv(:,:,e))'*...
20                 Hat_Phi_Gradient(:,k)*Hat_Phi_Matrix(j,q));
21 A(jj,jj) = A(jj,jj) + ...
22     omega(q)*2*Area*gamma(F_e_vector(1),F_e_vector(2))...
23             *Hat_Phi_Matrix(k,q)*Hat_Phi_Matrix(j,q);
24 ...

```

Capitolo 6

Problema parabolico

6.1 Problema Parabolico

Nel caso di fenomeni evolutivi, caratterizzati da variazioni nel tempo oltre che nello spazio, considereremo il seguente problema prototipo descritto in $\Omega \times [0, T]$, dove $T > 0$ è una costante.

$$\begin{cases} \frac{\partial u}{\partial t} - \nabla \cdot (\nu \nabla u) + \beta \cdot \nabla u + \gamma u = f & \text{in } \Omega, \text{ per } 0 < t \leq T \\ u = g_D & \text{su } \Gamma_D, \text{ per } 0 < t \leq T \\ \nu \frac{\partial u}{\partial n} = g_N & \text{su } \Gamma_N, \text{ per } 0 < t \leq T \\ u = u_0 & \text{in } \Omega, \text{ per } t = 0 \end{cases} \quad (6.1)$$

I coefficienti ν , β , γ soddisfano le stesse ipotesi del problema di diffusione-trasporto-reazione, mentre f , g_N sono funzioni tali che $f \in L^2(0, T; L^2(\Omega))$, $g_N \in L^2(0, T; L^2(\Gamma_N))$, $u_0 \in L^2(\Omega)$.

La strategia nella risoluzione di questo problema consiste nell'applicare il metodo degli elementi finiti per ogni passo di discretizzazione in tempo.

Dopo aver scritto il problema in forma variazionale, la semi-discretizzazione in spazio del problema porta al seguente sistema di equazioni:

$$\begin{cases} Bu' + Au = f & \text{in } (0, T] \\ \mathbf{u}(0) = \mathbf{u}_0 \end{cases} \quad (6.2)$$

dove A è la solita matrice di rigidezza, \mathbf{f} è il solito vettore del termine noto, mentre B prende il nome di matrice di massa ed è definita come segue

$$B = (b_{jk}) = b(\phi_k, \phi_j) = \int_E \phi_j \phi_k \quad (6.3)$$

dove le ϕ_i sono le funzioni di base. La soluzione e la sua derivata sono indicati come vettori poiché contengono il valore di \mathbf{u} in ogni punto della triangolazione del dominio Ω . A partire da questa formulazione, applicando la discretizzazione in tempo, si può sviluppare un metodo per approssimare la soluzione. Tra le varie formule che si possono

usare useremo quella di Crank-Nicholson che è basata sulla formula di quadratura dei trapezi. Esplicitando la soluzione si ottiene

$$\left(B + \frac{\Delta t}{2} A \right) \mathbf{u}^{n+1} = \left(B - \frac{\Delta t}{2} A \right) \mathbf{u}^n + \frac{\Delta t}{2} (\mathbf{b}^n + \mathbf{b}^{n+1}) \quad (6.4)$$

dove la notazione all'apice indica la valutazione al tempo $t_n \in [0, T]$. Considerando anche la condizione al bordo mista Dirichlet-Neumann in Matlab si implementa la seguente formula:

$$\begin{aligned} \left(B_0 + \frac{\Delta t}{2} A_0 \right) \mathbf{u}_0^{n+1} &= M_0 \mathbf{u}_0^n - M_D \left(\mathbf{u}_D^{n+1} - \mathbf{u}_D^n \right) - \frac{\Delta t}{2} A_0 \mathbf{u}_0^n - \frac{\Delta t}{2} A_D \left(\mathbf{u}_D^n + \mathbf{u}_D^{n+1} \right) + \\ &\quad \frac{\Delta t}{2} \left(\mathbf{b}^n + \mathbf{b}^{n+1} \right) + \frac{\Delta t}{2} \left(\mathbf{G}_N^n + \mathbf{G}_N^{n+1} \right) \end{aligned} \quad (6.5)$$

dove i vettori \mathbf{G}_N^n e \mathbf{u}_D^n fanno riferimento alle correzioni da apportare al sistema lineare, come per gli altri capitoli. Invece il pedice 0 si riferisce all'oggetto costruito a partire dal problema di Dirichlet omogeneo. Questo significa che una volta trovata la matrice della soluzione omogena (che sulle righe ha i gradi di libertà e sulle colonne gli istanti di tempo), si devono aggiungere i punti il cui valore è noto dalla condizione al bordo di Dirichlet.

Esempio 3. Vediamo un primo esempio di applicazione. Abbiamo considerato la soluzione $u(x, y, t) = t(x^2 + y^3)$ relativa al problema parabolico:

- $u(x, y, t) = t(x^2 + y^3)$ (Soluzione esatta);
- $\nu(x, y, t) = x$ (Coefficiente di diffusione);
- $\beta(x, y, t) = (3y, 0)$ (Coefficiente di convezione);
- $\gamma(x, y, t) = 0$ (Coefficiente di reazione);
- $f(x, y, t) = x^2 + y^3 - 4xt$ (Termine forzante);
- $g_D(x, y, t) = u(x, y, t)$ su $\partial\Omega$ (Condizione al bordo di Dirichlet, l'abbiamo posta uguale alla soluzione reale per semplicità di implementazione);
- $u(x, y, 0) = 0$ (soluzione iniziale);
- $Q_T = \Omega \times [0, T]$, con $\Omega = [0, 1] \times [0, 1]$ e $T = 1$;

Abbiamo applicato il metodo degli elementi finiti con area pari a $5e - 3$ e passo di discretizzazione in tempo pari a 0.25. L'errore è stato calcolato per semplicità in $t = 1$ poiché, essendo l'ultimo istante, dà un'idea dell'errore complessivo in ogni istante. Di seguito i grafici di evoluzione nel tempo e il grafico dell'errore:

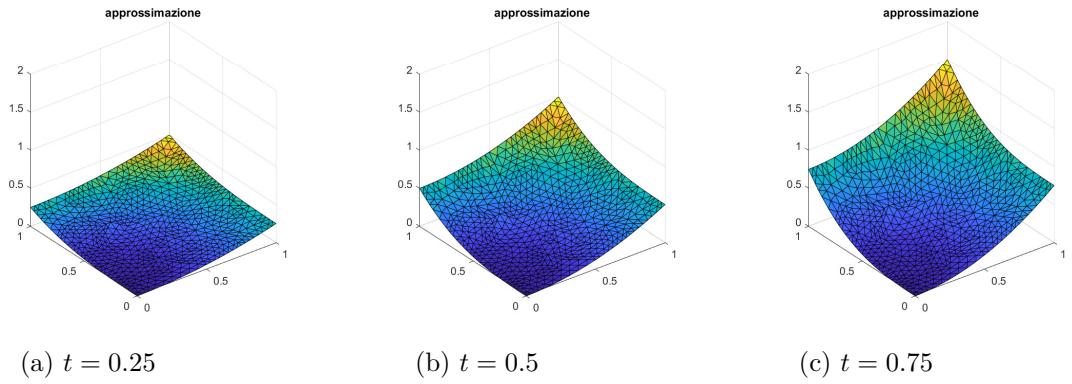


Figura 6.1: Grafici dell’evoluzione in tempo della soluzione del problema di sopra, calcolato mediante elementi finiti con area pari a $5e - 3$ e passo di discretizzazione in tempo pari a 0.25

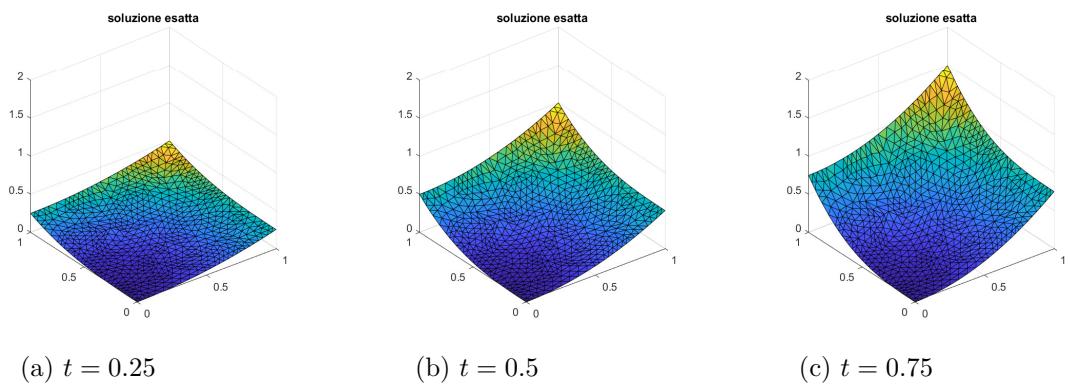


Figura 6.2: Grafici della funzione esatta in tre istanti di tempo diversi

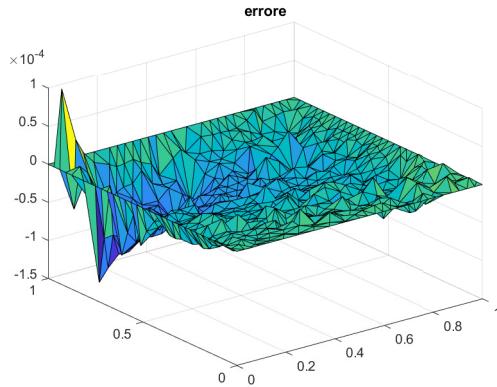


Figura 6.3: Grafico della funzione $\text{err} = \mathbf{u} - \mathbf{u}_h$ al tempo $t = 1$, relativo al problema di sopra

Visivamente non si riesce a cogliere la differenza tra la soluzione e l'approssimazione, ciò suggerisce la bontà dell'algoritmo, che verificheremo tecnicamente nel prossimo paragrafo dedicato allo studio dell'ordine di convergenza.

6.2 Ordine di convergenza

La verifica dell'ordine di convergenza di questa applicazione del metodo degli elementi finiti necessita di una maggiore accortezza. Infatti in questo caso l'errore dipende da due passi di discretizzazione: quello in spazio e quello in tempo. Prendendo come riferimento di errore quello che si ottiene al tempo T , si può dimostrare la seguente diseguaglianza:

$$\|u(x, y, T) - u_{h,\Delta t}\|_{L^2(\Omega)} \leq C_{\Delta t} \Delta t^2 + C_h h^{k+1} \quad (6.6)$$

che vale solamente nel caso si usi il metodo di Crank-Nicholson, mentre l'indice k indica il grado dei polinomi usati come funzioni di base. Per poter verificare che la convergenza sia rispettata sia in spazio che in tempo, è necessario considerare problemi con soluzioni che rendano estremamente basso l'errore nella dimensione che non si vuole verificare. Per al convergenza in spazio useremo le aree come nei capitoli precedenti, per la convergenza in tempo non ce ne sarà bisogno, poiché possiamo direttamente usare Δt . Gli esempi che abbiamo considerato sono i seguenti:

- $u(x, y, t) = t + e^{-(x+y)}$ per verificare la convergenza in spazio
- $u(x, y, t) = t^3 + x + y$ per verificare la convergenza in tempo

Esempio 4. Il primo esempio riguarda la verifica della convergenza in spazio. Abbiamo considerato i seguenti dati relativi al problema parabolico:

- $u(x, y, t) = t + e^{-(x+y)}$ (Soluzione esatta);

- $\nu(x, y, t) = 1$ (Coefficiente di diffusione);
- $\beta(x, y, t) = (0,0)$ (Coefficiente di convezione);
- $\gamma(x, y, t) = y$ (Coefficiente di reazione);
- $f(x, y, t) = 1 + yt + (y - 2)e^{-(x+y)}$ (Termine forzante);
- $g_D(x, y, t) = u(x, y, t)$ su Γ_D (Condizione al bordo di Dirichlet, l'abbiamo posta uguale alla soluzione reale per semplicità di implementazione);
- $g_N(x, y, t) = -e^{-(x+y)}$ su Γ_N
- $u(x, y, 0) = e^{-(x+y)}$ (soluzione iniziale);
- $Q_T = \Omega \times [0, T]$, con $\Omega = [0,1] \times [0,1]$ e $T = 1$;
- $\Gamma_N = \{(x, y) \in \Omega \mid y = 1\}$, e $\Gamma_D = \partial\Omega \setminus \Gamma_N$

Abbiamo applicato il metodo degli elementi finiti considerando tre diversi valori di area $[5e - 3 \cdot 1e - 3 \cdot 7e - 4]$ e un passo di discretizzazione in tempo pari a 0.1. Quest'ultimo è stato scelto appositamente piccolo in modo da rendere il più basso possibile il contributo del tempo nell'ordine di convergenza. Per questo metodo abbiamo usato i polinomi di secondo grado, dunque $k = 2$, ma siccome l'ordine è stato calcolato a partire dai valori di area, ci aspettiamo che l'ordine di convergenza sia pari a $\frac{k+1}{2} = 1.5$. Sperimentalmente abbiamo ottenuto 1.528.

I grafici della funzione sono riportati nelle figure [6.4a](#) e [6.4b](#).

Esempio 5. Il secondo esempio riguarda la verifica della convergenza in tempo. Abbiamo considerato i seguenti dati relativi al problema parabolico:

- $u(x, y, t) = t^3 + x + y$ (Soluzione esatta);
- $\nu(x, y, t) = 1$ (Coefficiente di diffusione);
- $\beta(x, y, t) = (y, 0)$ (Coefficiente di convezione);
- $\gamma(x, y, t) = 1$ (Coefficiente di reazione);
- $f(x, y, t) = -x - t^3 + 3t^2$ (Termine forzante);
- $g_D(x, y, t) = u(x, y, t)$ su $\partial\Omega$ (Condizione al bordo di Dirichlet, l'abbiamo posta uguale alla soluzione reale per semplicità di implementazione);
- $u(x, y, 0) = x + y$ (soluzione iniziale);
- $Q_T = \Omega \times [0, T]$, con $\Omega = [0,1] \times [0,1]$ e $T = 1$;

Abbiamo applicato il metodo degli elementi finiti considerando tre diversi valori di discretizzazione in tempo $[0.25 \ 0.2 \ 0.1]$ e un'area pari a $7e - 4$. Quest'ultimo è stato scelto appositamente piccolo in modo da rendere il più basso possibile il contributo dello spazio nell'ordine di convergenza. Abbiamo sempre usato i polinomi di base di secondo grado, in questo modo, poiché la parte della soluzione in spazio è di primo grado, il corrispettivo errore di approssimazione sarà ulteriormente ridotto. Come abbiamo già detto abbiamo implementato la formula di Crank-Nicholson che, in teoria, ci garantisce un ordine di convergenza in tempo pari a 2. Sperimentalmente abbiamo ottenuto 1.977. I grafici della funzione sono riportati nelle figure 6.5a e 6.5b.

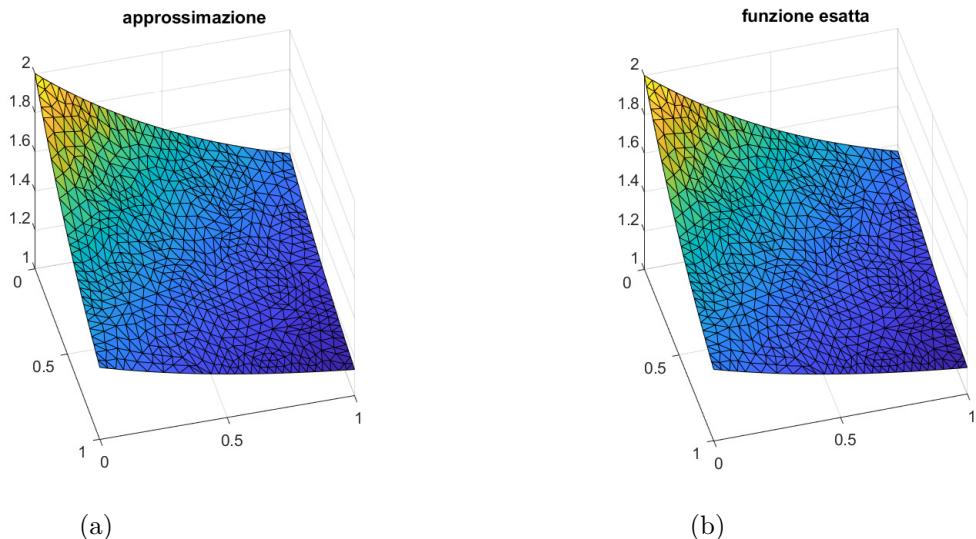


Figura 6.4: Grafico dell'approssimazione della funzione $u(x, y, t) = t + e^{-(x+y)}$ in $t = 1$ tramite elementi finiti con valore di area pari a $5e - 3$, passo di discretizzazione in tempo 0.1 e polinomi di base di grado $k = 2$ (a sinistra), e grafico della soluzione esatta (a destra)

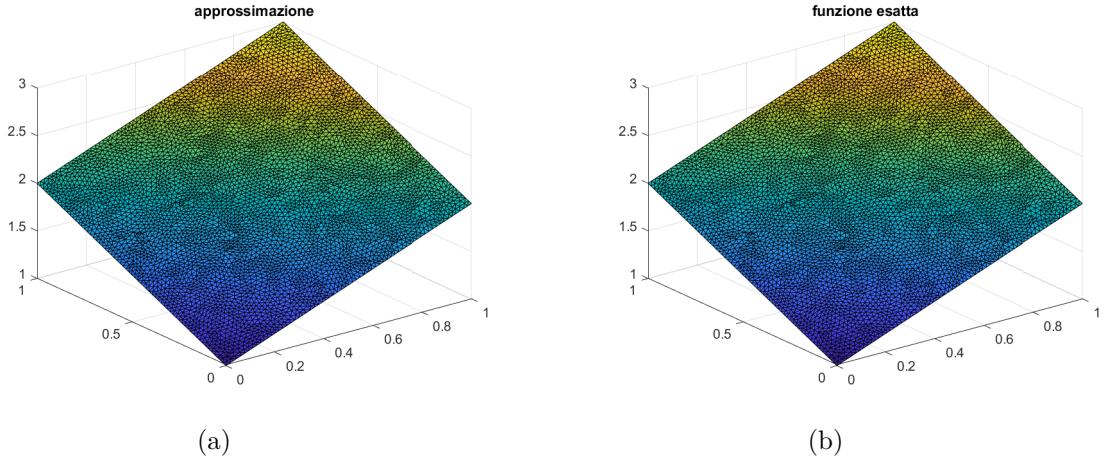


Figura 6.5: Grafico dell'approssimazione della funzione $u(x, y, t) = t^3 + x + y$ in $t = 1$ tramite elementi finiti con valore di area pari a $7e - 4$, passo di discretizzazione in tempo 0.1 e polinomi di base di grado $k = 2$, (a sinistra), e grafico della soluzione esatta (a destra)

Per riassumere, gli ordini di convergenza sperimentali che abbiamo ottenuto sono:

- ordine di convergenza in spazio in norma L^2 : 1.528
- ordine di convergenza in tempo in norma L^2 : 1.977

Per entrambi gli esempi abbiamo notato che la verifica dell'ordine di convergenza ha richiesto molto più tempo rispetto agli esempi precedenti. In particolare per la verifica della convergenza in tempo. Questo comportamento è giustificabile dal fatto che, con l'aggiunta della dimensione del tempo, i calcoli richiesti per generare la soluzione siano molti di più.

6.3 Dettagli di implementazione

L'implementazione del metodo è basata sempre sul metodo degli elementi finiti, con l'aggiunta del tempo. In particolare, escludendo l'implementazione della matrice di massa e degli oggetti algebrici aggiuntivi, la parte più importante riguarda proprio la gestione del tempo. Abbiamo deciso di generare prima una matrice di dati che contenga per ogni istante di tempo la soluzione omogenea del problema e di risolvere completamente il problema in tal senso, richiamando svariate volte il metodo degli elementi finiti (con la funzione `calcoloUPar` che è basata sul codice dei \mathbb{P}_2). Successivamente abbiamo usato un “ciclo for” annidato per aggiungere alla soluzione omogenea la parte di soluzione legata al bordo di Dirichlet, in modo analogo ai casi precedenti senza il dominio del tempo.

6.4 Codice Matlab

In questa sezione vedremo il codice con cui abbiamo risolto il problema parabolico. Per prima cosa abbiamo inizializzato le variabili

```

1 nT = T/Dt + 1;
2 Ndof = max(geom.pivot.pivot);
3 t = linspace(0,T,nT);
4nPunti = size(geom.elements.coordinates,1);
5nTriangoli = size(geom.elements.triangles,1);

```

poi abbiamo inizializzato la matrice della soluzione omogenea inserendo anche la soluzione iniziale data dal problema e abbiamo calcolato completamente la soluzione omogenea

```

1 global U0
2 U0 = zeros(Ndof,nT);
3
4 for j = 1:nPunti
5     jj = geom.pivot.pivot(j);
6     if jj>0
7         U0(jj,1) = u0(geom.elements.coordinates(j,1), ...
8                         geom.elements.coordinates(j,2));
9     end
10    end
11    for r=1:length(t)-1
12        calcoloUPar(f, eps, betha, gamma, gN, gD, t, r, upwind);
13    end

```

Successivamente abbiamo inserito nell'omogenea la soluzione al bordo in ogni istante

```

1 U = zeros(nPunti,nT);
2 for r = 1:length(t)
3     %Costruisco il vettore di valori al bordo di Dirichlet
4     uD = zeros(length(geom.pivot.Di),1);
5     for i = 1:length(geom.pivot.Di)
6         x_D = geom.elements.coordinates(geom.pivot.Di(i,1),1);
7         y_D = geom.elements.coordinates(geom.pivot.Di(i,1),2);
8         uD(i) = gD(x_D, y_D, t(r));
9     end
10    %Costruisco la vera matrice delle soluzioni
11    for j = 1:nPunti
12        jj = geom.pivot.pivot(j);
13        if jj>0
14            U(j,r) = U0(jj,r);
15        else
16            U(j,r) = uD(-jj);
17        end
18    end
19 end

```