POLITECNICO DI MILANO

Facoltà di Ingegneria dei Sistemi

Corso di Laurea Specialistica in Ingegneria Matematica

Orientamento *Scienze computazionali per l'ingegneria*

Elementi Finiti *Discontinuous Galerkin* Ibridizzabili per Problemi Ellittici in 3D

Candidato:
ANDREA SACCONI
721764

Relatore: Prof. **RICCARDO SACCO**
Correlatore: Dr. Ing. **MARCO RESTELLI**

A.A. 2010/2011

Indice

Elenco delle figure

Elenco del codice listato

Ringraziamenti

Aequales quoque, devexi iam veris imago,
optati plausus fuerunt et laudis avari
in comitem, is quamvis omnes in pectore ferret.
Omnia quapropter quondam sibi cara perosus,
solus et ingratae curvus sub pondere vitae
exsul uti fugiens, petiit deserta locorum.

(IOANNES BAPTISTA PIGATUS c.r.s., *Sacerdos moriens*, 72–77)

Così come ho fatto per la laurea triennale, esordisco nei miei ringraziamenti con sei esametri dattilici di Padre Giovanni Battista Pigato (1910 – 1976), sacerdote cattolico dell'Ordine dei Padri Somaschi, insigne latinista, nonché valoroso Cappellano degli Alpini della *Julia* sui fronti greco-albanese e russo. Rispetto ai versi citati nella prima laurea, tratti dal suo *Pax in bello*, qui ho inserito invece alcuni versi del suo testamento spirituale, che si avvicinano molto bene ad alcuni pensieri che ho maturato durante questi anni di studio.

Sono finalmente giunto alla fine di questa specialistica, di questa Ingegneria matematica che mi ha visto arrivare come una meteora, "trapiantato" da un altro corso di laurea che – chissà, per qualche strano motivo – mi appariva troppo povero di numeri.
La fatica e il logorio per arrivare a questo punto sono stati notevoli. Appena calcate le aule della Nave, in quel lontano giorno del mio ventiduesimo compleanno, i problemi non sono tardati ad arrivare: come scrivevo ad una cara amica, mi sentivo come un topolino di fronte ad un'alta montagna, o come un soldato inerme che solleva sconsolato lo sguardo verso una fortezza inespugnabile. Senza strumenti adeguati, e con tanti debiti di frequenza da sanare, mi pareva di aver fatto il passo più lungo della gamba, e di sentirmi così inadeguato, sommerso da formule e materie diverse, nonché dalla memoria di quanto invece avevo lasciato. Faticando, con lo spirito brontolone e mai intimorito dalle difficoltà che hanno sempre contraddistinto la mia giovane vita, sono ormai arrivato (e direi anche bene) al tanto sospirato traguardo.

Ringrazio innanzitutto la mia famiglia, che per due anni ho visto solamente nel fine settimana, di ritorno dalla trasferta milanese. Riuscire a vedere dal treno la basilica di Sant'Abbondio – raro e meraviglioso esempio di romanico lariano con due campanili, di chiara ispirazione nordeuropea – era il segnale che il lago e le montagne erano lì ad attendermi, e insieme a loro mamma, papà, mia sorella maggiore Silvia (detta *Sissina*) e i miei nonni.

Ringrazio i Padri Somaschi dell'Istituto Usuelli di Milano, presso i quali ho risieduto durante i due anni di frequenza dei corsi; a loro va un sincero grazie per l'attenzione, la discrezione e l'ospitalità concessami.

Ringrazio il prof. Riccardo Sacco, relatore di questa tesi, per la precisa spiegazione dei metodi e i preziosi suggerimenti nella stesura finale.
Ringrazio l'ing. Marco Restelli, che con infinita pazienza mi ha aiutato, ad inizio febbraio, a revisionare il codice riga per riga. Grazie alla sua disponibilità siamo riusciti a scovare e risolvere tutti i piccoli bachi che si annidavano qua e là. Non so davvero come ripagare tutte le ore che gli ho fatto perdere nel controllare le funzioni di base, l'assemblaggio delle matrici e la visualizzazione delle soluzioni!
Ringrazio l'ing. Carlo de Falco, per il quale *Octave* e C++ non hanno segreti.

Ringrazio poi Simone, amico e fedele compagno di viaggio di questa specialistica. A lui, "profugo" come me di Ingegneria gestionale, alla sua dettagliata conoscenza degli orari, delle aule, dei programmi dei corsi, che mi hanno spinto a seguirlo nell'avventura della specialistica, un rinnovato grazie per la stima che mi ha sempre dimostrato, per l'impegno profuso nei progetti condivisi (Statistica, Fluidodinamica, Programmazione avanzata) e per le telefonate-fiume sempre animate. Spero che anche lui possa concludere presto il suo ciclo di studi e trovare la gratificazione che merita già da ora, nell'avventura ad Abu Dhabi in cui si sta per lanciare. Grazie per tutte le riflessioni condivise sul lavoro o sul dottorato, riguardo ai quali – anche qui, c'era da immaginarselo – abbiamo spesso parlato.
Ringrazio poi tutti i miei compagni di corso, quasi tutti ormai già laureati: a loro va il mio in bocca al lupo per le carriere o gli studi superiori che hanno intrapreso.
Ringrazio anche tutte le persone che mi hanno sempre sostenuto, soprattutto in quei mesi in cui – per vari motivi – non sono riuscito a rendere come in passato.

Ringrazio, *last but not least*, la Provvidenza, presenza silenziosa sempre pronta a vigilare sul mio cammino. Non ho il minimo dubbio che saprà indicarmi – nei tempi e nei modi a Lei sola noti – come, dove e con chi impiegare a fondo le mie energie e il mio spirito brontolone perchè anche io possa finalmente trovare, come dice la S. Scrittura, "gioia piena"!

Andrea

Riassunto della tesi

L'obiettivo di questa tesi di laurea è lo studio approfondito dei metodi *Discontinuous Galerkin*, con particolare attenzione ai cosiddetti metodi *ibridizzabili*, e la realizzazione di un codice di calcolo per la risoluzione delle equazioni di diffusione-trasporto-reazione, basato sulla filosofia di **programmazione ad oggetti** in C++ e all'uso del *software* gratuito **Octave** per l'assemblaggio e la risoluzione numerica dei sistemi lineari.
Attenzione particolare è stata data anche alla visualizzazione delle soluzioni: dopo averle salvate in opportuni *files* in formato XML secondo le specifiche di VTK (*Visualization Toolkit*), sono poi state visualizzate grazie a *Paraview*.

Di seguito verrà presentato lo sviluppo del lavoro di tesi, attraverso una breve ma significativa presentazione di ogni capitolo e del suo contributo al lavoro nel suo complesso.

Il **capitolo 1** delinea il contesto matematico in cui la tesi s'inserisce: **l'Analisi numerica e le Equazioni a Derivate Parziali**.
Lo studio di queste materie costituisce l'ossatura portante del corso di Laurea Specialistica e, in particolare, dell'indirizzo *Maior* da me seguito; si vuole sottolineare quanto sia importante conoscere la modellistica matematica per saper descrivere accuratamente diverse realtà fisiche, rappresentabili e simulabili opportunamente attraverso codici di calcolo numerico. Viene inoltre presentato lo schema concettuale da seguire per poter passare dalle equazioni descriventi il fenomeno in analisi alle tecniche di risoluzione numerico-informatiche più opportune.

Il **capitolo 2** presenta con molto dettaglio i **metodi ad elementi finiti *Discontinuous Galerkin***, attraverso la loro nascita ed evoluzione storica e il confronto con i più tradizionali e comunemente insegnati metodi *Continuous Galerkin*. In particolare, dopo un'accurata disamina di questi metodi in termini di età di comparsa in letteratura, dimensione del problema, significato dei gradi di libertà, gestione delle condizioni al contorno e scelta del grado polinomiale, si è proceduto con lo studio di tali metodi per le **equazioni ellittiche del second'ordine**, già introdotte nel capitolo 1.
La trattazione prosegue con lo studio del processo di **ibridizzazione**, con l'obiettivo di chiarire innanzitutto la comprensione della sua portata così come recepita nella letteratura di riferimento. A questa sezione si vuole dare particolare risalto, in quanto è importante

cogliere che **questi metodi misti ibridi**, ancora poco diffusi e utilizzati, **sono a tutti gli effetti metodi di Galerkin**, caratterizzati da una **formulazione agli spostamenti generalizzata** per una **nuova incognita**, λ_h, **definita sulle frontiere degli elementi**, da cui è poi possibile calcolare il valore delle incognite interne, definite in ciascun tetraedro.

Il **capitolo 3** presenta la struttura del **codice ad oggetti scritto in linguaggio C++**, utile a precalcolare tutte le quantità matematiche necessarie alla costruzione delle matrici locali, alla risoluzione dei sistemi lineari per il calcolo delle incognite e alla conseguente visualizzazione delle soluzioni tramite *Paraview*.
Sono state ampiamente sfruttate le librerie API C++ di *Octave*, utilizzato non semplicemente come clone gratuito di *Matlab*®, bensì come supporto per la presenza di tipi già sviluppati e ampiamente testati da programmatori ed esperti del calcolo scientifico.
La trattazione si articola rispetto ai diversi **namespace** in cui si è voluta suddividere la programmazione: il primo fa riferimento alla **griglia di calcolo** e a tutte le informazioni di natura topologica e geometrica; il secondo gestisce gli **spazi approssimanti ad elementi finiti**, ponendo in evidenza la presenza di incognite con diverso numero di componenti (scalari vs. vettoriali) e diverso dominio di definizione (tridimensionale vs. bidimensionale); il terzo manipola nodi e pesi di quadratura per l'**integrazione numerica**, necessaria per il calcolo delle matrici locali; il quarto ed ultimo calcola tutte le informazioni necessarie per **visualizzare** le soluzioni numeriche, secondo le specifiche del *software* VTK.

Il **capitolo 4** si occupa di descrivere il **codice *Octave*** utilizzato per **l'assemblaggio**, la **risoluzione numerica**, il ***post-processing***, il **calcolo dell'errore** e la **visualizzazione**. Il codice di calcolo è strutturato come *package*, in modo da poter estendere le funzionalità già presenti gratuitamente in *Octave* e, ipoteticamente, in futuro, renderlo disponibile ad uso pubblico sul *web*.

Il **capitolo 5** presenta i **risultati delle simulazioni numeriche**. Il primo caso test è volto a studiare gli ordini di convergenza ottenibili sperimentalmente all'aumentare del grado polinomiale e al raffinarsi della griglia di calcolo. Il secondo caso presentato, invece, descrive un flusso di Darcy in un mezzo poroso caratterizzato da una frattura e da permeabilità discontinua; la scelta dei parametri e le quantità studiate fanno riferimento alla recente pubblicazione interna del MOX [CDASC2010], rispetto alla quale è possibile confrontare direttamente la fisica del problema e le conclusioni ottenute.

Il **capitolo 6** raccoglie le considerazioni conclusive di questa tesi e propone possibili direttrici di sviluppo per il futuro.

L'**appendice A** presenta con maggiore rigore alcuni aspetti analitici e geometrici non introdotti esplicitamente nei capitoli della tesi, ove già sufficientemente ricchi di formule

matematiche.
Un'attenzione particolare viene prestata alla gestione di entità geometriche definite su domini differenti (elementi vs. facce di bordo), alla trasformazione di Piola (necessaria per la corretta manipolazione dei campi vettoriali definiti su ciascun elemento), alla procedura di ricostruzione del flusso e del potenziale (con l'obiettivo di un profilo asintotico di convergenza migliore, a parità di griglia di calcolo) e alla generazione della griglia con il *software* GMSH.

L'**appendice B** vuole essere un sintetico manuale di utilizzo del codice di calcolo, che guidi l'utente ad una corretta scelta dei parametri di ingresso necessari alle simulazioni.

Abstract

The aim of this Master thesis work is a detailed analysis of *Hybridizable Discontinuous Galerkin methods* (HDG) for convection-diffusion-reaction equations and its implementation via object-oriented and *Octave* programming.
The thesis is organized in six chapters whose contents are shortly described below.
Chapter 1 focuses on the importance of numerical approximation of partial differential equations; in addition, particular attention is paid to the conceptual path to be followed in the resolution of a problem of physical interest.
Chapter 2 analyizes in depth *Discontinuous Galerkin* (DG) methods, in terms of age of the method, size of problem, meaning of degrees of freedom, and other important features.
A particular class of DG methods, called *hybridizable* DG methods, is then studied in detail.
Chapters 3 and 4 are devoted to a thorough description of, respectively, C++ classes and *Octave* code, written for solving convection-diffusion-reaction equations using HDG methods.
Chapter 5 is divided in two parts: the first one deals with experimental rate of convergence of L^2 errors for five different unknowns; the second one is devoted to simulate a Darcy flow in a porous bulk with heterogeneous permeability.
Chapter 6 sums up the main ideas developed in the thesis and indicates potential extensions.
Appendix A focuses on analytical and geometrical aspects (reference element, Piola transformation, post-processing, and mesh generation) non directly explained in the previous chapters.
Appendix B is a brief tutorial to be read before using the code.

Capitolo 1

Analisi numerica di *PDEs*

1.1 Equazioni a Derivate Parziali e Analisi Numerica

L'**approssimazione numerica delle equazioni a derivate parziali** (nel seguito si userà spesso, per brevità, l'acronimo *PDEs*, dall'inglese *Partial Differential Equations*) costituisce una delle più importanti branche dell'Analisi Numerica. Considerato l'ampio spettro di applicabilità (dalla Meccanica delle Strutture all'Emodinamica, dalla Fisica dei Semiconduttori alla Dinamica delle Popolazioni, etc.), al risolutore è richiesto di possedere conoscenze multidisciplinari che siano in grado di abbracciare la notevole varietà di aspetti presenti.

In primo luogo, è fondamentale avere un'**adeguata conoscenza della fisica del problema**: in questo modo è più semplice comprendere in anticipo quale sarà il comportamento esibito delle soluzioni incognite; nondimeno, tale conoscenza può condurre ad una scelta appropriata dei metodi numerici di risoluzione. Si pensi, a titolo di esempio, alle equazioni che governano la Fisica dei Semiconduttori: nel celebre **modello *Drift-Diffusion***[1] due delle tre incognite coinvolte sono le concentrazioni di elettroni e lacune che, per loro natura, sono strettamente positive. Nell'approssimazione numerica e nella manipolazione del sistema lineare ad essa associato sarà dunque obiettivo del risolutore cercare di preservare questa positività[2]. Ottenere soluzioni approssimate che violano palesemente le leggi della Fisica è spesso spia di una non adeguata selezione del metodo numerico o della finezza dei parametri che in quest'ultimo devono essere tarati.
In secondo luogo, le moderne formulazioni dei problemi differenziali sono basate su **tecniche variazionali**, che conducono alla ricerca di una soluzione in **spazi di Hilbert** (o eventualmente di Banach), per la cui comprensione sono necessarie alcune conoscenze

[1] Questo modello, nella sua formulazione classica, è al giorno d'oggi un po' superato, vista la miniaturizzazione sempre più spinta nella costruzione dei dispositivi elettronici; solo un'adeguata correzione di tipo *quantistico* può mettere in luce i fenomeni tipici della scala dei nanometri.

[2] Interessante, a questo proposito, il caso dei sistemi lineari per i quali la matrice dei coefficienti è una M-matrice: non semplicemente una proprietà astratta, tipica di un manuale di Algebra lineare, bensì un'avveduta scelta di risoluzione che sappia conservare da subito le caratteristiche fisiche delle incognite del problema.

di Analisi Funzionale. Obiettivo non trascurabile dell'analista numerico è saper fornire stime *a priori* per la soluzione, nella norma più "adeguata" al problema in esame. Inoltre, eventuali conoscenze analitiche sulla regolarità della soluzione esatta[3] possono indirizzare al meglio la scelta del metodo numerico, e di conseguenza determinare l'accuratezza raggiungibile nel rispetto delle risorse computazionali a disposizione.

1.2 La struttura concettuale dell'approssimazione numerica

Per l'approssimazione di un problema ai limiti, il processo decisionale da seguire è ben schematizzato dalla figura 1.1, tratta da [QV2008]:

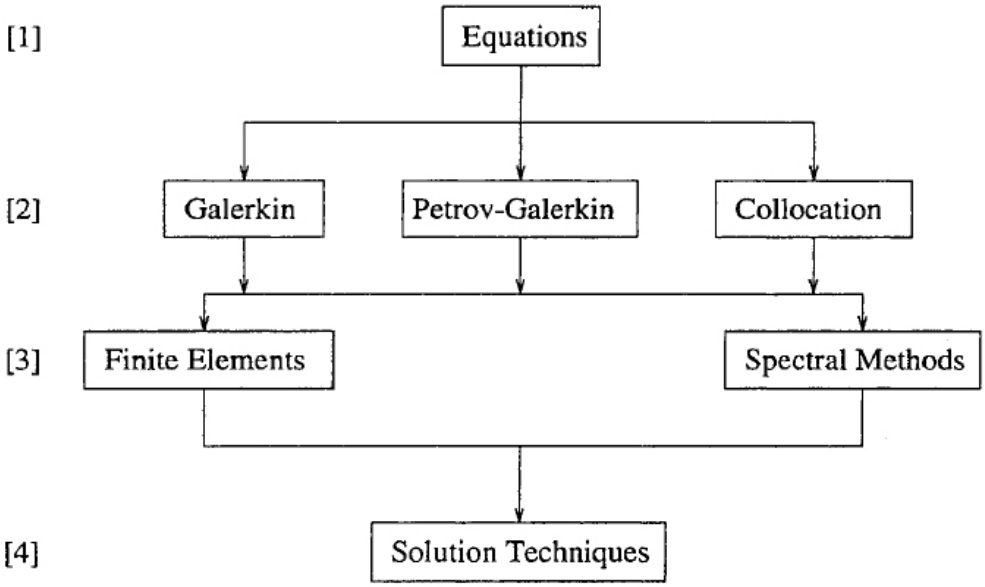

Figura 1.1: Struttura concettuale dell'approssimazione numerica.

Il livello [1] è costituito dal **problema ai limiti** in esame, espresso nella sua *formulazione debole* contenente le opportune condizioni al bordo. Essa discende dalle equazioni scritte nella cosiddetta *forma forte*, e ha l'obiettivo di ampliare l'insieme di ricerca delle soluzioni oltre i consueti spazi di funzioni sufficientemente regolari che possiedono derivate continue almeno fino al massimo ordine presente nell'equazione. Per una trattazione più approfondita delle diverse soluzioni di un problema differenziale e della loro maggiore o minore adattabilità alla risoluzione numerica, si rimanda a [SAL2007], Cap. 9, § 2.
Il livello [2] indica la **tipologia di discretizzazione** (o metodo numerico) da porre in atto per ricondurre il problema assegnato ad un altro avente *dimensione finita*. Come lecito attendersi, la particolare strategia adottata determinerà la struttura del problema

[3] A titolo di esempio, si considerino le stime di abbattimento dell'errore per Elementi Finiti lagrangiani ([QUA2008], Cap. 3): diminuire il passo di griglia h o aumentare il grado r dei polinomi approssimanti non garantisce sempre un miglioramento dell'ordine di convergenza. La scelta ottimale per l'adattività dipende, in ultima analisi, dalla regolarità di Sobolev posseduta dalla soluzione.

algebrico.
Come indicato in figura 1.1, sono due le principali tipologie di discretizzazione:

- **metodo di Galerkin**: è basato su una *formulazione integrale* del problema differenziale e ammette, fra le sue varianti più degne di nota, il cosiddetto **metodo di Petrov-Galerkin**.
  Tali metodi sono basati sulla volontà di prescrivere il soddisfacimento dell'equazione differenziale non più puntualmente, nel senso usuale dell'Analisi matematica, ma *in senso integrale*: si vuole cioè che il residuo dell'equazione (la differenza fra il primo membro contenente le incognite e il termine noto), moltiplicato scalarmente per qualsivoglia funzione test di un ben preciso spazio, si annulli. Si passa dunque ad una *formulazione integrale* dove si richiede alle soluzioni un'inferiore regolarità; così facendo si riesce a mantenere uno stretto collegamento con la fisica sottostante e a descrivere fenomeni particolari[4] con adeguato rigore analitico (si veda, ad esempio, [QUA2008], Cap. 2).

- **metodo di collocazione**: è basato invece sul soddisfacimento della *PDE* (dove un'opportuna controparte discreta prende il posto dell'operatore differenziale di partenza) in nodi ben precisi del dominio computazionale. Esso può essere tuttavia interpretato come una *formulazione di Galerkin generalizzata*, ottenuta combinando l'approccio alla Galerkin con l'integrazione numerica operata con formule di quadratura gaussiane.

Il livello [3] specifica la **natura dei sottospazi** usati per approssimare gli spazi di Hilbert infinito-dimensionali nei quali sono ambientate le soluzioni esatte. Le scelte più comuni prevedono di utilizzare funzioni polinomiali a tratti di basso ordine nel caso degli elementi finiti e polinomi algebrici di grado più elevato nel caso dei metodi spettrali.
Al livello [4] è invece deputata la **scelta degli algoritmi migliori per i sistemi lineari** derivanti dalla discretizzazione operata. E' sempre bene tenere in considerazione le struttura topologica e le proprietà possedute dalle matrici (sparsità, simmetria, definita positività, etc.) che vengono manipolate: in casi particolari è infatti possibile sfruttare procedure di calcolo implementate *ad hoc*, così da ridurre l'onere computazionale.

1.2.1 Livelli dell'approssimazione numerica considerati

Nel seguito verranno esaminati i cosiddetti **Elementi finiti *Discontinuous Galerkin***, con ampio dettaglio implementativo in più nella loro versone **ibridizzabile** (livello [2] della figura 1.1), e l'attenzione cadrà in particolar modo sulle loro caratteristiche **analitiche** (livello [3]: scelta degli spazi finito-dimensionali, ordini di convergenza ottenibili) e **algebriche** (dimensione del problema matriciale, gradi di libertà globalmente accoppiati, utili

[4] Tra essi ricadono, a titolo di esempio, un impulso elettrico o una discontinuità di *shock* nel moto di un fluido, che trovano una naturale rappresentazione matematica con la *delta di Dirac* o la ricerca di una soluzione *discontinua* ma *essenzialmente limitata*.

per il livello [4]). La fisica sottostante al problema (livello [1]), invece, s'inserisce nella ben consolidata teoria delle **equazioni ellittiche del second'ordine**, per la cui trattazione – non necessaria in questa tesi – si rimanderà alla bibliografia (ad esempio, [SAL2007] Cap. 2).

Capitolo 2

I metodi *Discontinuous Galerkin*

Il capitolo precedente ha fornito una sintetica panoramica dei vari aspetti presenti nel processo di approssimazione numerica delle *PDEs*.
I paragrafi che seguono descriveranno in dettaglio gli **elementi *Discontinuous Galerkin*** e una loro possibile versione, cosiddetta **ibridizzata**. La trattazione delineerà in primo luogo la comparsa dei metodi DG nella letteratura scientifica, con particolare attenzione ad **analogie e differenze con i metodi *Continuous Galerkin***; da ultimo, si darà adeguato spazio al concetto di *ibridizzazione* di un metodo agli elementi finiti.

2.1 I metodi *Discontinuous Galerkin*: storia e caratteristiche

2.1.1 Convezione e soluzioni discontinue

Come evidenziato in [CKS2000], esiste una notevole varietà di problemi di interesse pratico in cui la *convezione* riveste un ruolo importante. Si possono ricordare, ad esempio: meteorologia, oceanografia, dinamica dei gas, acustica, flussi turbolenti, estrazione di idrocarburi, problemi in acque basse, trasporto di inquinanti in mezzi porosi, simulazioni di dispositivi a semiconduttore, etc. Da questo lungo elenco emergono l'importanza e la necessità di sviluppare **metodi robusti, accurati ed efficienti** per la risoluzione numerica; come lecito attendersi, quest'obiettivo ha catalizzato l'attenzione di molti ricercatori ed esperti nelle scienze applicate.

Non mancano, tuttavia, difficoltà di realizzazione, principalmente per due motivi.
In primo luogo, è ben noto che la soluzione esatta di problemi puramente convettivi (non lineari) **sviluppa singolarità in tempo finito**; in secondo luogo, tali soluzioni possono esibire una **struttura molto ricca e complicata nell'intorno di tali discontinuità**[1].

[1] Si pensi alla risoluzione del problema di Riemann per le *equazioni di Eulero*: pur partendo da due soli stati costanti, a sinistra e a destra del punto spaziale di discontinuità per $t = 0$, la soluzione esatta esibisce una straordinaria ricchezza di comportamento: è autosimile su raggi uscenti da tale punto e si compone di onde di rarefazione, *shock* e discontinuità di contatto, a seconda degli stati iniziali assegnati.

Pertanto, quando si vogliono costruire nuovi metodi numerici per problemi di questo genere, bisogna garantire che **le discontinuità simulate dal metodo di calcolo corrispondano proprio a quelle di interesse fisico**. Nondimeno, bisogna assicurare che la comparsa di singolarità non sia accompagnata da oscillazioni spurie tali da inquinare la bontà dell'approssimazione[2]; tuttavia, pur soddisfacendo questo vincolo, il metodo deve idealmente rimanere il più accurato possibile nei pressi delle singolarità allo scopo di catturare la struttura, possibilmente molto ricca, esibita dalla soluzione esatta.
A queste difficoltà si riesce a far fronte con successo grazie al considerevole sviluppo degli schemi *alle differenze finite con alta risoluzione* e *ai volumi finiti*, applicati a sistemi iperbolici non lineari, con l'uso di **flussi numerici** e **limitatori di pendenza**.
Dato che i metodi *Discontinuous Galerkin* (DG) ammettono soluzioni approssimate discontinue, **essi possono a buon diritto essere considerati una generalizzazione dei metodi ai volumi finiti**: infatti, anziché far evolvere nel tempo la media di cella (che altro non è se non un polinomio di grado zero), **la soluzione spaziale** (a istante temporale fissato) **sarà espressa attraverso polinomi di grado superiore**, e i coefficienti dello sviluppo (variabili in tempo) costituiranno le nuove incognite del problema.
Per valutare il flusso all'interfaccia di separazione fra celle contigue, si ricorre ad opportuni flussi numerici e – per ottenere una diminuzione della variazione totale del profilo spaziale – si costruiscono opportuni limitatori di pendenza per ogni cella di calcolo.
Da queste considerazioni, appare chiaro che i metodi DG incorporano – in maniera del tutto naturale – nel *framework* degli elementi finiti i concetti chiave della fluidodinamica comprimibile; essi sono altresì in grado di catturare le singolarità fisicamente rilevanti senza produrre eccessive oscillazioni spurie in prossimità di queste ultime.

2.1.2 Caratteristiche fondamentali dei metodi DG

In virtù della loro natura di elementi finiti, i metodi DG presentano alcuni vantaggi rispetto ai "concorrenti" ai volumi finiti e alle differenze finite. In [CKS2000] sono ben evidenziati:

- l'ordine reale di accuratezza dipende solamente dalla soluzione esatta; metodi DG di ordine (formale) arbitrariamente alto possono essere ottenuti attraverso un'opportuna scelta del grado dei polinomi approssimanti;

- i metodi DG sono altamente parallelizzabili: infatti, essendo elementi *discontinui*, **la matrice di massa è diagonale a blocchi** e, dato che la dimensione dei blocchi è uguale al numero di gradi di libertà del corrispondente elemento, i blocchi possono essere invertiti a mano (oppure attraverso un manipolatore simbolico) una volta per tutte;

[2] I metodi numerici applicati ai problemi della fluidodinamica comprimibile possono presentare caratteristiche diverse, in termini di monotonia, contrattività rispetto ad una norma, diminuzione della variazione totale, ordine di convergenza. Esiste sempre un *trade-off* tra queste proprietà, in quanto il soddisfacimento di alcune impedisce quello di altre. Per una trattazione più ampia, si veda a tal proposito [LEV2004].

- i metodi DG sono in grado di gestire geometrie anche complicate e richiedono un trattamento semplice delle condizioni al contorno;

- con i metodi DG possono essere facilmente implementate strategie di adattività, in quanto il raffinamento o deraffinamento della griglia possono essere ottenuti senza dover tenere in conto il vincolo di continuità tra un elemento e l'altro, come invece accade nel caso di metodi agli elementi finiti conformi. Nondimeno, il grado dei polinomi approssimanti può essere facilmente modificato da un elemento all'altro.

2.1.3 Confronto fra formulazioni DG e CG

Seguendo l'ottimo approccio di [RIV2008], si pongono a confronto i metodi CG (*Continuous Galerkin*) e DG (*Discontinuous Galerkin*) da un punto di vista pratico. E' bene ricordare che la soluzione CG è, *per costruzione*, formata da funzioni che sono polinomi su ogni singolo elemento, **globalmente continui** sul dominio computazionale. I metodi DG sono invece caratterizzati da una soluzione alla quale, *a priori*, **non è imposto alcun vincolo di continuità** all'interfaccia di separazione fra elementi adiacenti.
Ecco che cosa si può dire comparando i due metodi:

- **età del metodo**: la formulazione CG è in uso da più di 60 anni, e ciò ha condotto a centinaia di pubblicazioni che ne hanno analizzato in dettaglio i vari aspetti. I metodi DG, invece, sono **decisamente più recenti** e hanno quindi riscosso solo da minor tempo l'attenzione della comunità scientifica internazionale. Molte filosofie seguite nel caso CG (ad esempio, l'adattività) possono essere trasferite anche ai metodi DG;

- **dimensione del problema**: per un generico metodo **DG**, **il numero totale di gradi di libertà è direttamente proporzionale al numero di elementi** che formano la griglia computazionale; la costante di proporzionalità è funzione del grado che si va scegliere per i polinomi approssimanti. Nel caso **CG**, invece, **il numero di gradi di libertà dipende dal numero di vertici e possibilmente dal numero di vertici ed elementi della *mesh***. Per esempio, si consideri una *mesh* strutturata composta da 5×5 elementi rettangolari. I gradi di libertà per un'approssimazione DG di grado $1, 2, 3, 4$ sono rispettivamente $75, 150, 250, 375$, mentre nel caso CG (ferma restando la medesima scelta sui polinomi) le incognite da determinare sono $36, 121, 256, 441$. Di conseguenza, già su una griglia così piccola, se si vanno ad impiegare polinomi cubici o superiori (cioè, $k \geq 3$), l'approccio CG è più costoso rispetto a quello DG. La ragione risiede nel fatto che si è obbligati ad usare lo spazio $\mathbb{Q}_k$ su griglie rettangolari per assicurare la continuità tipica dei metodi CG, mentre è possibile continuare ad impiegare $\mathbb{P}_k$ sulle medesime griglie per i metodi DG. La figura 2.1 mostra il rapporto fra il numero totale di gradi di libertà per il metodo CG e il numero totale di gradi di libertà per il metodo DG, definiti su una griglia uniforme di $N \times N$ rettangoli, con N variabile tra 10^2 e 10^{16}. L'approccio CG è meno costoso

rispetto al DG se il grado polinomiale è minore o uguale a 3; i rapporti tendono ai valori limite $1/3, 2/3, 9/10, 16/15, 25/21$ per grado $1, 2, 3, 4, 5$, rispettivamente. Su *mesh* triangolari, invece, il metodo DG è più costoso rispetto alla controparte CG: ad esempio, su griglia uniforme di $N \times N \times 2$ elementi triangolari, il rapporto tra gradi di libertà del metodo CG rispetto al metodo DG tende a $1/6, 1/3, 9/20, 14/30, 25/42$ per grado $1, 2, 3, 4, 5$, rispettivamente, per N tendente ad infinito. Si può notare, inoltre, come tale rapporto aumenti al crescere del grado polinomiale utilizzato.

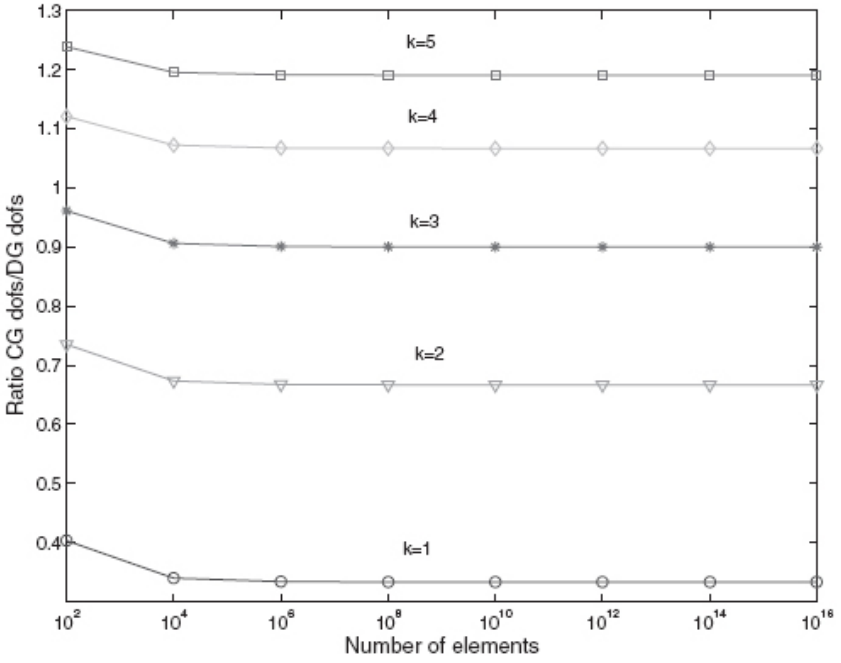

Figura 2.1: Rapporto fra il numero di gradi di libertà nei due casi, CG e DG, per una *mesh* uniforme con elementi rettangolari. Al tendere ad infinito del numero di elementi si raggiunge un asintoto orizzontale, funzione del grado polinomiale scelto.

- **significato dei gradi di libertà**: gli utilizzatori che hanno meno confidenza con il metodo degli elementi finiti fanno uso, quasi sempre, delle sole funzioni lineari a tratti. Grazie alla caratteristica forma "a capanna" delle funzioni di base, i gradi di libertà dell'approccio CG corrispondono al valore della soluzione nei vertici della *mesh*; è questa una proprietà molto interessante, soprattutto quando **si desidera visualizzare la soluzione** calcolata per verificarne il significato fisico. Al contrario, **i gradi di libertà di un metodo DG non hanno un particolare significato oltre a quello di essere i coefficienti dell'espansione** della soluzione incognita rispetto all'insieme delle funzioni di base. Ciò significa che, per ottenere la soluzione DG in un particolare punto del dominio, è necessario calcolare la suddetta espansione, vale a dire calcolare ogni singola funzione di base, moltiplicarla per il corrispondente coefficiente e sommare rispetto a tutte le funzioni di base dello spazio lineare in cui la soluzione approssimante viene cercata. In un particolare vertice, dunque, sono

presenti più valori della soluzione numerica[3].

Nulla vieta, nell'approssimazione DG, di utilizzare le medesime funzioni di base del caso CG, la cui struttura a supporto compatto favorisce sempre una certa sparsità della matrice, da cui la possibilità di applicare tecniche di Algebra lineare numerica più veloci ed efficienti.

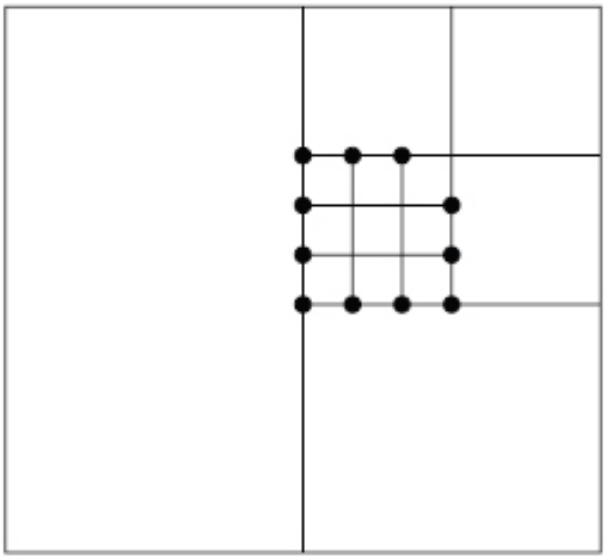

Figura 2.2: *Hanging nodes*, evidenziati con un punto nero spesso.

- **nodi *hanging***[4]: tale nome deriva dalla nomenclatura dei metodi CG per indicare vertici della *mesh* corrispondenti a gradi di libertà. Con abuso di notazione (introdotto volontariamente in [RIV2008]), si indica qui come nodo *hanging* un qualsivoglia vertice della *mesh* posizionato sull'interno di un lato (o di una faccia, a seconda se il problema è ambientato in $\mathbb{R}^2$ o in $\mathbb{R}^3$). La figura 2.2 contiene una *mesh* con 11 nodi *hanging*. **Tale griglia computazionale *non conforme* può essere gestita da un metodo DG di ordine arbitrario, mentre ciò non è possibile per un metodo CG**: infatti, il risolutore DG può posizionare a piacimento il numero di nodi *hanging*, dal momento che non sussistono vincoli di continuità tra un elemento e quelli ad esso adiacenti.

[3] Si noti che in questa trattazione si fa sempre implicito riferimento ad elementi finiti *lagrangiani*, che permettono di **associare ad ogni grado di libertà un ben preciso punto del dominio**, in cui la funzione di base corrispondente vale 1, mentre tutte le rimanenti sono nulle. Nulla, *a priori*, vieta di scegliere lo **stesso spazio lineare**, ma di descriverlo *attraverso un diverso insieme di funzioni di base* (ad esempio, la base dei monomi, che ha nel famoso triangolo di Pascal una comoda rappresentazione dei suoi coefficienti). La soluzione incognita (che si suppone unica grazie ad opportuni teoremi di Analisi funzionale), *sarà la stessa*, ma saranno diversi sia la **matrice del sistema lineare** da risolvere, sia **i valori numerici di ogni componente del vettore incognito**. La scelta di elementi lagrangiani è dettata, pertanto, sia da una **maggiore facilità di risoluzione** matriciale (dovuta alla località dei supporti delle funzioni di base, da cui discende una certa sparsità nella matrice), sia da **una corrispondenza visiva** che non sempre è possibile ottenere.

[4] Letteralmente, dall'inglese: *appesi, penzolanti.*

- **grado dei polinomi e funzioni di base**: è relativamente semplice cambiare il grado di una soluzione approssimata DG, andando a modificare limitate porzioni di un *software* di risoluzione: si tratta solamente di modificare le *routines* che calcolano le funzioni di base.
  Nel caso CG è necessaria un'attenzione maggiore, dato che è necessario andare ad "incollare" adeguatamente funzioni di base locali (che hanno supporto limitato) tra un elemento e i contigui; nel caso DG, è invece sufficiente una semplice estensione a zero oltre l'elemento di appartenza, senza vincoli di continuità che, *a priori*, non sono imposti;

- **accuratezza**: entrambi i metodi convergono (in opportune norme) al raffinarsi della griglia e/o all'aumentare del grado polinomiale scelto. Per il caso CG, si può fare riferimento a [QUA2008], Cap. 3; nel caso DG, invece, bisogna analizzare in dettaglio *quale particolare approssimazione discontinua* si va a scegliere. In questa tesi si verificherà sperimentalmente la bontà di particolari metodi DG in termini di abbattimento dell'errore.

- **condizioni al contorno**: le condizioni di Dirichlet sono imposte tipicamente in senso debole nel caso DG, mentre lo sono in senso forte nel caso CG[5]; in linea di principio, tuttavia, è anche possibile sviluppare metodi DG in cui le condizioni al bordo sono imposte in senso forte[6];

- **conservazione della massa**: come si vedrà più compiutamente nel seguito, i metodi DG esibiscono un bilancio locale di massa, mentre l'approccio CG lo conserva solamente se si prende in esame l'intero dominio computazionale. Tale questione è di non secondaria importanza soprattutto nei problemi di trasporto nei mezzi porosi, in cui la fisica del problema impone (o suggerisce) la scelta di un preciso metodo matematico rispetto ad un altro. Nel seguito verrà descritta in dettaglio l'importanza

[5] Si ricordi che le condizioni di Dirichlet vengono anche dette *essenziali*, dato che vengono prescritte all'atto stesso di scelta dello spazio vettoriale in cui si va a cercare l'approssimazione. Si pensi, per semplicità, al Laplaciano con forzante non nulla e condizioni di Dirichlet omogeneo sul bordo. In questo caso la formulazione debole viene scritta nello spazio H_0^1, in cui è imposto che le funzioni siano nulle (nel senso delle tracce) sul bordo del dominio in esame. Scritto il metodo di Galerkin, **tutte le funzioni di base soddisfano il vincolo al bordo**, in quanto esso è uno dei requisiti che ha condotto alla scelta di quel particolare spazio approssimante e non di un altro.

[6] E' sempre bene ricordare che le condizioni al bordo giocano un ruolo importante nella modellistica numerica del problema differenziale in esame, sia nella scrittura del problema variazionale astratto, sia nell'implementazione informatica della risoluzione. Dal primo punto di vista, va sottolineato che la scelta di uno spazio di funzioni test piuttosto che un altro porta alla nascita di diverse questioni. Ad esempio, nel caso di condizioni di Dirichlet non omogenee per l'equazione di Laplace, come non notare che le funzioni test in H_0^1 non soddisferebbero le condizioni al bordo, e quindi il problema non potrebbe essere analizzato ricorrendo alle tecniche di Analisi funzionale più note (il Lemma di Lax-Milgram su tutti)? In questo caso, grazie alla linearità del problema e alla tecnica del *rilevamento del dato al bordo*, l'analisi di buona positura è comunque semplice. In linea generale, la scelta di uno spazio di test rispetto ad un altro fa sorgere la seguente domanda: è possibile in questo framework **dimostrare esistenza e unicità della soluzione**? Nondimeno, dal punto di vista **algebrico-computazionale**, è semplice **costruire e gestire** funzioni di base che – a livello *discreto* – **riproducano** il comportamento esibito nel *continuo* (si pensi alla difficoltà di costruire spazi finito-dimensionali contenenti funzioni a divergenza nulla per il problema di Stokes)?

della cosiddetta ***conservativity condition***, vero elemento fondante dei metodi DG *ibridizzabili*.

2.1.4 Nota storica: il primo esempio di metodo DG

Il primo metodo DG venne presentato nel 1973 da Reed and Hill (citato in [CKS2000]) per la risoluzione dell'equazione di trasporto di neutroni

$$\sigma\, u + \nabla \cdot (\boldsymbol{a}\, u) = f \qquad \text{in } \Omega$$

dove σ è un numero reale e $\boldsymbol{a}$ un vettore costante. L'importanza del metodo fu subito riconosciuta da LeSaint e Raviart, che nel 1974 pubblicarono il primo studio analitico ad esso dedicato.

Pur essendo nati nell'ambito *convettivo*, i metodi *Discontinuous Galerkin* furono presto estesi ad una pluralità di applicazioni fisiche e ingegneristiche, lineari e non lineari, iperboliche, paraboliche ed ellittiche.
Lo sviluppo storico degli elementi finiti DG va oltre lo scopo di questa tesi; si rimanda pertanto all'analisi in [CKS2000] per tutti i dettagli del caso; ivi è menzionato, naturalmente, anche **il caso dei metodi DG applicati alla risoluzione di problemi ellittici**, che saranno oggetto delle simulazioni numeriche operate con il codice di calcolo appositamente sviluppato in questa tesi.
Da segnalare, inoltre, altri due interessanti riferimenti bibliografici: per una panoramica storica più semplice e concisa, si può fare riferimento alla pubblicazione didattica [COCK2003]; [LI2006] è invece dedicato specificatamente alle applicazioni di termofluidodinamica.

2.2 Metodi DG ed equazioni ellittiche del second'ordine

Come visto in precedenza, i metodi *Discontinuous Galerkin* nacquero nell'alveo delle equazioni di trasporto, ma furono presto proposti anche per le equazioni ellittiche del second'ordine. Nel 2002 [ABCM2002] **proposero uno studio unificante dei metodi DG applicabili a problemi ellittici**.
Senza pretesa di completezza (per cui si rimanda all'articolo), si indicheranno qui i principali risultati di tale studio e gli elementi utili per il prosieguo.
Per semplicità di trattazione, in [ABCM2002] gli autori analizzano il seguente problema modello:

$$
\begin{aligned}
-\triangle u &= f \qquad \text{in } \Omega, && \text{(2.1a)}\\
u &= 0 \qquad \text{su } \partial\Omega && \text{(2.1b)}
\end{aligned}
$$

dove Ω è un dominio poligonale convesso e f un'assegnata funzione di $L^2(\Omega)$. Per incasellare tutti i metodi DG in un *framework* comune, per prima cosa si riscrive il problema come un sistema del primo ordine:

$$
\begin{aligned}
\sigma &= \nabla u && \text{(2.2a)}\\
-\nabla\cdot\sigma &= f, \qquad \text{in } \Omega, && \text{(2.2b)}\\
u &= 0 \qquad \text{su } \partial\Omega && \text{(2.2c)}
\end{aligned}
$$

Moltiplicando le due equazioni, rispettivamente, per le funzioni test τ e v, e integrando formalmente su un sottoinsieme K di Ω, si ottiene:

$$
\begin{aligned}
\int_K \sigma\cdot\tau &= -\int_K u\,\nabla\cdot\tau + \int_{\partial K} u\, n_K\cdot\tau\\
\int_K \sigma\cdot\nabla v &= \int_K f\, v + \int_{\partial K} \sigma\cdot n_K\, v
\end{aligned}
$$

dove n_K è la normale uscente a ∂K. Questa è la formulazione debole da cui gli autori partono per lo studio unificato dei metodi DG.
Introducendo ora un'opportuna triangolazione $\Omega_h = \{K\}$ e i seguenti spazi ad elementi finiti:

$$
\begin{aligned}
\Sigma_h &:= \{\tau\in[L^2(\Omega)]^d : \tau|_K\in\Sigma(K)\ \forall K\in\Omega_h\}\\
V_h &:= \{v\in L^2(\Omega) : v|_K\in P(K)\ \forall K\in\Omega_h\}
\end{aligned}
$$

dove $P(K)=\mathscr{P}_k(K)$ e $\Sigma(K)=[\mathscr{P}_k(K)]^d$ sono gli usuali spazi di funzioni polinomiali di grado k e d la dimensione del dominio; si perviene alla seguente formulazione generale:

trovare u_h e σ_h t.c., $\forall v$ e $\forall \tau$

$$\int_K \sigma_h \cdot \tau = -\int_K u_h \nabla \cdot \tau + \int_{\partial K} \hat{u}_K \, n_K \cdot \tau \tag{2.3a}$$

$$\int_K \sigma_h \cdot \nabla v = \int_K f \, v + \int_{\partial K} \hat{\sigma}_K \cdot n_K \, v \tag{2.3b}$$

dove i flussi numerici $\hat{\sigma}_K$ e $\hat{u}_K$ sono approssimazioni di $\sigma = \nabla u$ e di u, rispettivamente, sulla frontiera di ciascun elemento K della reticolazione. **Per completare la definizione di un particolare metodo DG, è dunque necessario esprimere compiutamente i flussi numerici $\hat{\sigma}_K$ e $\hat{u}_K$ in funzione delle incognite σ_h e u_h ed eventualmente delle condizioni al contorno**.

Saltando ora la trattazione storica e la derivazione nei singoli casi, di scarso interesse per la tesi, si può presentare direttamente il risultato dello studio unificante, sintetizzato nella figura 2.3.

Method	$\widehat{u}_K$	$\widehat{\sigma}_K$
Bassi–Rebay [10]	$\{u_h\}$	$\{\sigma_h\}$
Brezzi et al. [22]	$\{u_h\}$	$\{\sigma_h\} - \alpha_{\mathrm{r}}(\llbracket u_h \rrbracket)$
LDG [41]	$\{u_h\} - \beta \cdot \llbracket u_h \rrbracket$	$\{\sigma_h\} + \beta \llbracket \sigma_h \rrbracket - \alpha_{\mathrm{j}}(\llbracket u_h \rrbracket)$
IP [50]	$\{u_h\}$	$\{\nabla_h u_h\} - \alpha_{\mathrm{j}}(\llbracket u_h \rrbracket)$
Bassi et al. [13]	$\{u_h\}$	$\{\nabla_h u_h\} - \alpha_{\mathrm{r}}(\llbracket u_h \rrbracket)$
Baumann–Oden [15]	$\{u_h\} + n_K \cdot \llbracket u_h \rrbracket$	$\{\nabla_h u_h\}$
NIPG [64]	$\{u_h\} + n_K \cdot \llbracket u_h \rrbracket$	$\{\nabla_h u_h\} - \alpha_{\mathrm{j}}(\llbracket u_h \rrbracket)$
Babuška–Zlámal [7]	$(u_h\|_K)\|_{\partial K}$	$-\alpha_{\mathrm{j}}(\llbracket u_h \rrbracket)$
Brezzi et al. [23]	$(u_h\|_K)\|_{\partial K}$	$-\alpha_{\mathrm{r}}(\llbracket u_h \rrbracket)$

Figura 2.3: Lista dei principali metodi DG analizzati nello studio unificante [ABCM2002].

La simbologia è quella tradizionalmente in uso per indicare l'operatore di salto e di media; si rimanda all'articolo in questione per la bibliografia ivi citata nell'analisi generale dei diversi metodi e per l'espressione di tutti i coefficienti α_j, α_r e β necessari alla definizione completa dei flussi numerici.

Sommando su tutti i triangoli della reticolazione, si perviene alla seguente formulazione: trovare u_h e σ_h t.c., $\forall v$ e $\forall \tau$

$$\int_{\Omega_h} \sigma_h \cdot \tau = -\int_{\Omega_h} u_h \nabla \cdot \tau + \sum_{K \in \Omega_h} \int_{\partial K} \hat{u}_K \, n_K \cdot \tau \tag{2.4a}$$

$$\int_{\Omega_h} \sigma_h \cdot \nabla v = \int_{\Omega_h} f \, v + \sum_{K \in \Omega_h} \int_{\partial K} \hat{\sigma}_K \cdot n_K \, v \tag{2.4b}$$

Per la formulazione appena presentata è bene notare che:

- **ci sono due incognite** u_h e σ_h, e non più una soltanto: si passa dunque da una sola equazione ad un **sistema di equazioni del primo ordine**. La variabile σ_h (che è un campo vettoriale) assume spesso un significato fisico ben preciso (in termini di *flusso*, ad esempio), e assurge qui a variabile indipendente del problema;

- vista la scelta di polinomi discontinui, nulla si può dire sul valore di u_h sul bordo di interfaccia tra un elemento e l'altro. *A priori*, ci saranno **due valori**, ciascuno corrispondente al triangolo (risp. tetraedro) che condivide il lato (risp. la faccia).

- ciò che distingue tra loro i vari metodi è, come già detto in precedenza, la **particolare scelta dei flussi numerici**. Gli autori in [ABCM2002] individuano due principali famiglie in cui racchiudere i vari metodi passati in esame: da un lato, i metodi *Interior Penalties*, in cui si impone un'opportuna penalità alle interfacce fra elementi contigui, in modo da "forzare" una qualche condizione di continuità che gli spazi scelti non possiedono intrinsecamente; dall'altro lato, i metodi ispirati alle tecniche *a volumi finiti*, eredità della fluidodinamica comprimibile e dei flussi numerici per leggi di conservazione.
  Essendo inoltre (2.4) un problema a due campi, la scelta dei flussi numerici è cruciale nella **dimostrazione dell'esistenza e unicità della soluzione**.

2.3 Un caso particolare: i metodi *ibridizzabili*

Lo studio condotto fino ad ora ha messo in luce la **grande varietà di metodi DG** presenti, nonché l'importanza di un'**adeguata scelta dei flussi numerici all'interfaccia**. Proprio dall'analisi di che cosa succede a contatto fra un elemento e i suoi adiacenti, si prende ora in esame una particolare e molto interessante classe di metodi DG per le equazioni ellittiche: i cosiddetti metodi *ibridizzabili*.

Nota sulla nomenclatura Nel paragrafo precedente è stata utilizzata la variabile σ_h, campo vettoriale, per indicare l'incognita corrispondente al flusso, in perfetta analogia con la trattazione in [ABCM2002] e con la figura 2.3 ivi contenuta. Nel seguito la nomenclatura verrà leggermente modificata, per coerenza con quanto usato in [CGL2009]: la variabile campo vettoriale sarà indicata con $\boldsymbol{q}$, in grassetto per indicare che non è uno scalare bensì un vettore. Il corrispondente flusso numerico sarà dunque indicato con $\hat{\boldsymbol{q}}_h$ in luogo di $\hat{\sigma}_h$. La variabile scalare manterrà invece la medesima espressione.

2.3.1 Il processo di ibridizzazione

Seguendo lo schema condotto in [CGL2009], si può sottolineare subito una definizione: tutti i metodi che verranno presentati forniscono un'approssimazione per $(\boldsymbol{q}, u)$ nell'interno degli elementi $K \in \Omega_h$, detta $(\boldsymbol{q}_h, u_h)$, così come un'approssimazione per u sulle frontiere degli elementi, detta λ_h; è questo il motivo per il quale vengono detti ***ibridi***. Ciò risulta in perfetto accordo con la definizione di *metodi ibridi* data in un testo classico dell'Analisi numerica, [CIA1978]: **"possiamo definire con maggiore generalità come *ibrido* un qualsivoglia metodo agli elementi finiti basato su una formulazione in cui un'incognita è una funzione, o qualcuna delle sue derivate, definita sull'intero dominio Ω, e l'altra incognita è la traccia di qualcuna delle derivate di tale funzione, o la traccia della funzione medesima, lungo le frontiere dell'insieme K"**.
Naturalmente, *non tutti* i metodi agli elementi finiti possono essere riscritti usando la medesima struttura concettuale propria dell'ibridizzazione; si può dire che si può ibridizzare un assegnato metodo agli elementi finiti se si è in grado di trovare un metodo *ibrido* (parte del)la cui soluzione **coincide** con la soluzione del metodo inizialmente assegnato. Quest'ultimo viene dunque detto *ibridizzabile*, e il metodo ibrido viene considerato come *ibridizzazione* del metodo originario.

La prima ibridizzazione di un metodo agli elementi finiti fu proposta nel 1965 da Fraejis De Veubeke (si veda la referenza citata in [CGL2009]), nello studio di risoluzione numerica di un problema di elasticità lineare. Forse per il fatto che fu allora intesa come tecnica implementativa, la distinzione fra ibridizzazione e *condensazione statica* (una ben nota manipolazione algebrica per la riduzione dimensionale di matrici *già* assemblate) compare

raramente nella letteratura ingegneristica. Tuttavia, in [AB1985], **gli autori mostrarono che l'ibridizzazione era molto di più di un semplice artificio implementativo: si riuscì a dimostrare, infatti, che la nuova incognita λ_h (interpretabile anche come *moltiplicatore di Lagrange* associato ad un vincolo di continuità sul flusso approssimato) contiene preziose informazioni *aggiuntive* riguardo alla soluzione esatta**. Ciò fu utilizzato per migliorare l'accuratezza dell'approssimazione attraverso un opportuno *post-processing* locale.
Dopo circa due decenni emerse nella letteratura un nuova prospettiva sull'ibridizzazione (si veda [CG2004]), in cui veniva introdotta la **caratterizzazione della traccia approssimata λ_h come soluzione di una specifica formulazione debole**; tale studio venne condotto andando ad ibridizzare i metodi misti di Raviart-Thomas (RT) e Brezzi-Douglas-Marini (BDM) di grado polinomiale arbitrario. Si vuole ora mostrare che cosa significhi ***ibridizzare* un metodo *Discontinuous Galerkin***.
Si noti che che gli autori in [CGL2009] analizzano un problema modello leggermente modificato rispetto a quello in [ABCM2002], vale a dire un'equazione ellittica in cui compare anche il contributo di reazione. Ciò non intacca in alcun modo la comprensione dello spirito unificante proprio dei due articoli, ma semplicemente prepara il campo ad una formulazione più completa, con la presenza del trasporto, come sarà nei capitoli a seguire. Si preferisce tuttavia, per fedeltà a [CGL2009], seguire il medesimo problema modello, scritto in forma mista:

$$\begin{aligned} \boldsymbol{q} + \boldsymbol{a}\,\nabla u &= 0 \qquad \text{in } \Omega && \text{(2.5a)} \\ \nabla\cdot\boldsymbol{q} + d\,u &= f \qquad \text{in } \Omega && \text{(2.5b)} \\ u &= g \qquad \text{su } \partial\Omega && \text{(2.5c)} \end{aligned}$$

Sia Ω_h una reticolazione del dominio fisico Ω formata da elementi K di forma regolare, aventi diametro h_K; si ponga $\partial\Omega_h := \{\partial K : K \in \Omega_h\}$. Si associ a questa reticolazione l'insieme delle facce interne $\mathscr{E}_h^i$ e l'insieme delle facce di frontiera $\mathscr{E}_h^\partial$; si dice che $e \in \mathscr{E}_h^i$ se esistono due elementi K^+ e K^- in Ω_h tali che $e = \partial K^+ \cap \partial K^-$, mentre si dice che $e \in \mathscr{E}_h^\partial$ se esiste un elemento in Ω_h tale che $e = \partial K \cap \partial\Omega$. Si pone $\mathscr{E}_h := \mathscr{E}_h^i \cup \mathscr{E}_h^\partial$.
Si va alla ricerca di un'approssimazione $(\boldsymbol{q}_h, u_h, \lambda_h)$ della soluzione esatta $(\boldsymbol{q}|_\Omega, u|_\Omega, \lambda|_\Omega)$ in uno spazio finito-dimensionale $\boldsymbol{V}_h \times W_h \times M_h$ della forma

$$\begin{aligned} \boldsymbol{V}_h &:= \{\boldsymbol{v} \in \boldsymbol{L}^2(\Omega_h) : \boldsymbol{v} \in \boldsymbol{V}(K)\ \forall K \in \Omega_h\} \\ W_h &:= \{\omega \in L^2(\Omega_h) : \omega \in W(K)\ \forall K \in \Omega_h\} \\ M_h &:= \{\mu \in L^2(\mathscr{E}_h) : \mu|_e \in M(e)\ \forall e \in \mathscr{E}_h,\ \mu|_{\partial\Omega_D} = 0\} \end{aligned}$$

con $\boldsymbol{a}$ tensore uniformemente definito positivo in Ω, $d \geq 0$ q.o. in Ω e $\boldsymbol{V}(K), W(K), M(e)$ opportuni spazi polinomiali da precisarsi. Nelle equazioni i prodotti scalari verranno

indicati come segue:

$$\begin{aligned}(\zeta,\omega)_{\Omega_h} &:= \sum_{K\in\Omega_h}\int_K \zeta(x)\,\omega(x)\,\mathrm{d}x\\ \langle\zeta,\boldsymbol{v}\cdot\boldsymbol{n}\rangle_{\partial\Omega_h} &:= \sum_{K\in\Omega_h}\int_{\partial K}\zeta(\gamma)\,\boldsymbol{v}(\gamma)\cdot\boldsymbol{n}\,\mathrm{d}\gamma\\ (\boldsymbol{\sigma},\boldsymbol{v})_{\Omega_h} &:= \sum_{j=1}^{d}(\sigma_j,v_j)_{\Omega_h}\end{aligned}$$

per ogni coppia $\boldsymbol{\sigma}$, $\boldsymbol{v}$ di campi vettoriali di $\boldsymbol{H}^1(\Omega_h) := [H^1(\Omega_h)]^d$ e ogni coppia ζ, $\omega \in H^1(\Omega_h)$. Il versore normale esterno a ∂K è indicato con $\boldsymbol{n}$; $d = 2, 3$ rappresenta la dimensione del problema.
Procedendo con le usuali integrazioni per parti e ricordando che le variabili incognite sono $(\boldsymbol{q}_h, u_h, \lambda_h)$, approssimazioni del flusso, del potenziale e del **valore del potenziale alle interfacce**, si perviene alla seguente formulazione debole:

$$\begin{aligned}(\boldsymbol{c}\,\boldsymbol{q}_h,\boldsymbol{v})_{\Omega_h} - (u_h,\nabla\cdot\boldsymbol{v})_{\Omega_h} + \sum_{K\in\Omega_h}\langle\lambda_h,\boldsymbol{v}\cdot\boldsymbol{n}\rangle_{\partial K\setminus\partial\Omega} &= -\langle g_h,\boldsymbol{v}\cdot\boldsymbol{n}\rangle_{\partial\Omega} &\text{(2.6a)}\\ -(\boldsymbol{q}_h,\nabla\omega)_{\Omega_h} + \langle\hat{\boldsymbol{q}}_h\cdot\boldsymbol{n},\omega\rangle_{\partial\Omega_h} + (d\,u_h,\omega)_{\Omega_h} &= (f,\omega)_{\Omega_h} &\text{(2.6b)}\\ \sum_{K\in\Omega_h}\langle\mu,\hat{\boldsymbol{q}}_h\cdot\boldsymbol{n}\rangle_{\partial K} &= 0 &\text{(2.6c)}\end{aligned}$$

per ogni tripletta $(\boldsymbol{v},\omega,\mu) \in \boldsymbol{V}_h \times W_h \times M_h$. Si noti che $\boldsymbol{c} = \boldsymbol{a}^{-1}$.
Alcuni approcci DG possono essere letti in questa forma, anche se non tutti sono da considerarsi ibridizzabili: infatti, lo schema considerato nell'analisi unificante [ABCM2002] può essere scritto[7] nella nostra notazione come

$$\begin{aligned}(\boldsymbol{c}\,\boldsymbol{q}_h,\boldsymbol{v})_{\Omega_h} - (u_h,\nabla\cdot\boldsymbol{v})_{\Omega_h} + \sum_{K\in\Omega_h}\langle\hat{u}_h,\boldsymbol{v}\cdot\boldsymbol{n}\rangle_{\partial K\setminus\partial\Omega} &= -\langle g_h,\boldsymbol{v}\cdot\boldsymbol{n}\rangle_{\partial\Omega} &\text{(2.7a)}\\ -(\boldsymbol{q}_h,\nabla\omega)_{\Omega_h} + \sum_{K\in\Omega_h}\langle\hat{\boldsymbol{q}}_h\cdot\boldsymbol{n},\omega\rangle_{\partial K} + (d\,u_h,\omega)_{\Omega_h} &= (f,\omega)_{\Omega_h} &\text{(2.7b)}\end{aligned}$$

dove $\hat{\boldsymbol{q}}_h$ e $\hat{u}_h$ sono le cosiddette *tracce numeriche* del metodo *Discontinuous Galerkin*. Mettendo a confronto le (2.7a), (2.7b) con le (2.6a), (2.6b), (2.6c), ci si rende immediatamente conto che $u_h = \lambda_h$ su $\mathscr{E}_h^i$. Possiamo dunque notare che, affinchè un metodo agli elementi finiti sia ibridizzabile, **la traccia numerica $\hat{u}_h$ *deve essere ad un solo valore*. Questa rappresenta la principale differenza fra un generico metodo DG e un metodo DG ibridizzabile**. Si noti, inoltre, che la formulazione *ibridizzata* presenta una terza equazione, la (2.6c), detta *condizione di trasmissione* (o, in inglese, *conservativity condition*, per motivi che saranno chiari più avanti), che **dà informazioni in più su**

[7] Si ricordi che [ABCM2002] analizzavano il caso tradizionale del Laplaciano; è comunque sufficiente fare una piccola modifica alla parte diffusiva, inserendo un generico tensore al posto del tensore identità, nonché il contributo di reazione, per arrivare alla formulazione generale.

come possa essere approssimata u sul bordo degli elementi K.

Inseriamo qui alcuni dettagli utili alla comprensione della struttura della soluzione: dato il seguente spazio

$$\mathsf{M}_h := \{\mathsf{m} \in L^2(\mathscr{E}_h) : \mathsf{m}|_e \in M(e)\ \forall e \in \mathscr{E}_h\}$$

con $M(e)$ opportuno spazio a dimensione finita per ogni lato (risp. faccia) della *mesh*, s'introducono di seguito i cosiddetti **solutori locali**, che saranno utili a breve per individuare una decomposizione della soluzione $(\boldsymbol{q}_h, u_h, \lambda_h)$ in opportuni addendi[8].
La trattazione in questo capitolo fa riferimento al caso di diffusione-reazione con sole condizioni di Dirichlet; per una estensione al caso in cui siano presenti anche il trasporto e le condizioni di Neumann, si veda in Appendice, A.1.2.

Primo solutore locale:

$$(\boldsymbol{c}\,\mathcal{Q}\mathsf{m}, \boldsymbol{v})_K - (\mathcal{U}\mathsf{m}, \nabla\cdot\boldsymbol{v})_K = -\langle \mathsf{m}, \boldsymbol{v}\cdot\boldsymbol{n}\rangle_{\partial K} \tag{2.8a}$$

$$-(\mathcal{Q}\mathsf{m}, \nabla\omega)_K + \langle \hat{\mathcal{Q}}\mathsf{m}\cdot\boldsymbol{n}, \omega\rangle_{\partial K} + (d\,\mathcal{U}\mathsf{m}, \omega)_K = 0 \tag{2.8b}$$

Qui $\hat{\mathcal{Q}}\mathsf{m}$ rappresenta la traccia numerica del flusso che, in linea generale, è una funzione a due valori su $\mathscr{E}_h^i$. Nei prodotti scalari che vedono coinvolta $\hat{\mathcal{Q}}\mathsf{m}$ sul bordo ∂K di un generico elemento, l'integranda da utilizzarsi sarà quella definita relativamente per il corrispondente elemento K.
Negli esempi che verranno forniti nel seguito il flusso numerico $\hat{\mathcal{Q}}\mathsf{m}$ è espresso esplicitamente rispetto a $(\mathcal{Q}\mathsf{m},\mathcal{U}\mathsf{m})$; in altri casi, invece, va considerato come una funzione incognita. In tal caso sarebbe necessario introdurre *ad hoc* lo spazio per l'incognita $\hat{\mathcal{Q}}\mathsf{m}$ nonché un'equazione addizionale, tale da garantire che il solutore locale sia univocamente risolubile. Dal punto di vista teorico, tuttavia, nella presentazione di questo *framework* metodologico non è essenziale sapere quale sia la definizione precisa di di $\hat{\mathcal{Q}}\mathsf{m}$; l'unica richiesta "formale" necessaria è che $\mathsf{m} \mapsto (\mathcal{Q}\mathsf{m}, \hat{\mathcal{Q}}\mathsf{m},\mathcal{U}\mathsf{m})$ sia una mappa lineare ben definita.

Secondo solutore locale:

$$(\boldsymbol{c}\,\mathcal{Q}f, \boldsymbol{v})_K - (\mathcal{U}f, \nabla\cdot\boldsymbol{v})_K = 0 \tag{2.9a}$$

$$-(\mathcal{Q}f, \nabla\omega)_K + \langle \hat{\mathcal{Q}}f\cdot\boldsymbol{n}, \omega\rangle_{\partial K} + (d\,\mathcal{U}f, \omega)_K = (f, \omega)_K \tag{2.9b}$$

[8] I risolutori locali non sono altro che problemi differenziali identici, nelle equazioni, al problema fisico generale, ma definiti sul singolo elemento K della reticolazione; tali solutori consentono precisamente di collegare adeguatamente grandezze diverse, quali funzioni che vivono nell'intero elemento, con quelle che vivono sulla frontiera dell'elemento medesimo. Non sfugga la perfetta analogia che c'è tra i risolutori locali e le estensioni armoniche descritte in [QUA2008], Cap. 14, per la *Decomposizione di Domini*: l'obiettivo è sempre quello di decomporre la soluzione generale del problema in componenti diverse, ciascuna associata ad una particolare condizione (risp., variabile, forzante) di bordo (risp., sull'intero dominio). Ciò che rende lecita questa decomposizione è, naturalmente, la **linearità** del problema differenziale e la conseguente applicabilità del *Principio di sovrapposizione delle cause e degli effetti.*

Analogamente al caso del primo risolutore locale, si lascia non definita la traccia numerica $\hat{\mathcal{Q}}f$.

Assunzione 1. *Per ogni* $\mathsf{m} \in \mathsf{M}_h$ *esiste una sola tripletta di funzioni* $(\mathcal{Q}\mathsf{m}, \hat{\mathcal{Q}}\mathsf{m}, \mathcal{U}\mathsf{m})$*, che dipendono linearmente da* m*, soddisfacenti il sistema 2.8. Inoltre, per ogni* $f \in L^2(\Omega)$ *esiste una sola tripletta di funzioni* $(\mathcal{Q}f, \hat{\mathcal{Q}}f, \mathcal{U}f)$*, che dipendono linearmente da* f*, soddisfacenti il sistema 2.9.*

Ponendo $g_h \in \mathsf{M}_h$ come opportuna interpolazione del dato di Dirichlet g, per cui si possono operare le estensioni locali come nel caso della funzione m, un generico metodo ibridizzabile definisce un'approssimazione di $(\boldsymbol{q}, u)$ così esprimibile:

$$(\boldsymbol{q}_h, u_h) = (\mathcal{Q}\lambda_h + \mathcal{Q}g_h + \mathcal{Q}f, \mathcal{U}\lambda_h + \mathcal{U}g_h + \mathcal{U}f) \in \boldsymbol{V}_h \times W_h \tag{2.10}$$

dove *si assume che* λ_h *sia determinata* dalla seguente versione discreta della cosiddetta **condizione di trasmissione**:

$$\langle \mu, [[\hat{\mathcal{Q}}\lambda_h + \hat{\mathcal{Q}}g_h + \hat{\mathcal{Q}}f]] \rangle_{\mathscr{E}_h} = 0 \qquad \forall \mu \in M_h \tag{2.11}$$

Se si definisce il flusso numerico come

$$\hat{\boldsymbol{q}}_h := \hat{\mathcal{Q}}\lambda_h + \hat{\mathcal{Q}}g_h + \hat{\mathcal{Q}}f$$

e se la (estensione a zero su $\mathscr{E}_h$ della) funzione $[[\hat{\boldsymbol{q}}_h]]|_{\mathscr{E}_h^i}$ appartiene alla spazio M_h, allora la condizione (2.11) ci sta semplicemente dicendo che $[[\hat{\boldsymbol{q}}_h]]|_{\mathscr{E}_h^i} = 0$ puntualmente, vale a dire, la componente normale della traccia numerica $\hat{\boldsymbol{q}}_h$ è a un solo valore, o – volendo adottare la terminologia usata in [ABCM2002] – la funzione $\hat{\boldsymbol{q}}_h$ è un flusso numerico ***conservativo***. E' per questo motivo che la (2.11) viene chiamata ***conservativity condition***.

2.3.2 La caratterizzazione della variabile λ_h

Come ormai appare chiaro dalle considerazioni precedenti, la rilevanza dei metodi che si adattano alla struttura generale dell'ibridizzazione risiede nel fatto che **la variabile ibrida** λ_h **può essere caratterizzata attraverso una semplice formulazione debole, nella quale non compare nessuna delle altre variabili incognite**.
Ciò permette di leggere un metodo ibrido come un **vero e proprio metodo di Galerkin**, caratterizzato da una **formulazione agli spostamenti generalizzata**, in cui **gli spostamenti incogniti sono definiti sulle frontiere degli elementi**.

Teorema 1. *Si supponga che valga l'Assunzione 1 riguardo all'esistenza e unicità dei solutori locali. Allora* $\lambda_h \in M_h$ *soddisfa la 2.11 se e solo se soddisfa la seguente:*

$$a_h(\lambda_h, \mu) = b_h(\mu) \qquad \forall \mu \in M_h \tag{2.12}$$

dove

$$
\begin{aligned}
a_h(\eta,\mu) &= (\boldsymbol{c}\,\mathcal{Q}\eta,\mathcal{Q}\mu)_{\Omega_h} + (d\,\mathcal{U}\eta,\mathcal{U}\mu)_{\Omega_h} + \langle 1,[[(\mathcal{U}\mu-\mu)(\hat{\mathcal{Q}}\eta-\mathcal{Q}\eta)]]\rangle_{\mathscr{E}_h}\\
b_h(\mu) &= \langle g_h,[[\hat{\mathcal{Q}}\mu]]\rangle_{\mathscr{E}_h} + (f,\mathcal{U}\mu)_{\Omega_h}\\
&\quad - \langle 1,[[(\mathcal{U}\mu-\mu)(\hat{\mathcal{Q}}f-\mathcal{Q}f)]]\rangle_{\mathscr{E}_h}\\
&\quad + \langle 1,[[\mathcal{U}f(\hat{\mathcal{Q}}\mu-\mathcal{Q}\mu)]]\rangle_{\mathscr{E}_h}\\
&\quad - \langle 1,[[(\mathcal{U}\mu-\mu)(\hat{\mathcal{Q}}g_h-\mathcal{Q}g_h)]]\rangle_{\mathscr{E}_h}\\
&\quad + \langle 1,[[(\mathcal{U}g_h-g)(\hat{\mathcal{Q}}\mu-\mathcal{Q}\mu)]]\rangle_{\mathscr{E}_h}
\end{aligned}
$$

per ogni $\eta,\mu\in M_h$.

La dimostrazione del precedente teorema è tutt'altro che semplice, richiede anzi la prova di alcuni lemmi tecnici, per cui – per non appesantire ulteriormente la trattazione, già sufficientemente ricca di formule – si rimanda a [CGL2009]. Ivi sono anche contenute condizioni sufficienti per l'esistenza e unicità della variabile λ_h, anch'esse abbastanza tecniche. **Ciò che è veramente importante è riconoscere, invece, che i gradi di libertà globalmente accoppiati sono esclusivamente quelli associati alla variabile ibrida** λ_h, che risolve la formulazione debole (2.12).

2.3.3 La struttura di sparsità della matrice di stiffness per λ_h

Coerentemente a quanto anticipato nel § 1.2.1, è utile inserire qualche commento sulla struttura di sparsità della matrice di *stiffness* associata alla formulazione debole (2.12): sappiamo infatti (livello [4] della figura 1.1) che queste informazioni possono risultare importanti per scegliere i migliori metodi di Algebra lineare numerica con i quali risolvere il problema matriciale.
Per qualsivoglia base dello spazio di tracce approssimate M_h, indichiamo con $[\mu]$ il corrispondente vettore dei coefficienti dell'espansione di μ nella base scelta per M_h. Di conseguenza, la formulazione debole (2.12) si può esprimere in forma più compatta come

$$A[\lambda_h] = b$$

dove

$$[\mu]^t A[\lambda_h] = a_h(\lambda_h,\mu) \qquad \text{e} \qquad [\mu]^t b = b_h(\mu)$$

Ora, grazie alle relazioni

$$a_h(\eta,\mu) = -\sum_{K\in\Omega_h}\langle\mu,\hat{\mathcal{Q}}\eta\cdot\boldsymbol{n}\rangle_{\partial K} \qquad \text{e} \qquad b_h(\mu) = \sum_{K\in\Omega_h}\langle\mu,(\hat{\mathcal{Q}}f+\hat{\mathcal{Q}}g_h)\cdot\boldsymbol{n}\rangle_{\partial K}$$

otteniamo:

$$A = \sum_{K \in \Omega_h} A_K \qquad \text{e} \qquad b = \sum_{K \in \Omega_h} b_K$$

dove A_K e b_K sono definiti nel modo seguente:

$$[\mu]^t A_K [\eta] = -\langle \mu, \hat{\mathcal{Q}}\eta \cdot \boldsymbol{n} \rangle_{\partial K} \qquad \text{e} \qquad [\mu]^t b_K = \langle \mu, (\hat{\mathcal{Q}} f + \hat{\mathcal{Q}} g_h) \cdot \boldsymbol{n} \rangle_{\partial K}$$

L'assemblaggio della matrice globale come somma di contributi derivanti da ciascun elemento della *mesh* suggerisce in maniera sempre più chiara che **questi metodi misti ibridi esibiscono tutte le caratteristiche proprie di un metodo di Galerkin**.
Per quanto riguarda la struttura di sparsità, si possono già trarre alcune conclusioni importanti:

- se il supporto di $\mu \in M_h$ non interseca ∂K, allora si ha che $[\mu]^t b_K = 0$.
  Questa proprietà discende immediatamente dalla definizione del vettore di carico b_K: fissato l'elemento K, se il primo dei due ingressi del prodotto scalare è identicamente nullo sulla frontiera ∂K, è naturale che la quantità considerata sia nulla;

- se almeno uno dei supporti di μ o di $\eta \in M_h$ non interseca ∂K, allora si ha che $[\mu]^t A_K [\eta] = 0$.
  Nel caso di μ, valgono le medesime argomentazioni del punto precedente. Qualora invece sia η ad essere nulla su ∂K, ricordando che i risolutori locali consentono di ricavare una ed una sola soluzione (Assunzione 1), si deduce immediatamente che $(\hat{\mathcal{Q}}\eta)_K = 0$ su ∂K, da cui l'asserto.
  Poniamo in particolare l'accento su questo risultato: esso è possibile infatti perché la traccia numerica $\hat{\mathcal{Q}}\cdot$ *è* ***a due valori*** *su tutte le facce interne* $e \in \mathscr{E}_h^i$. Infatti, si prenda η come nella precedente dimostrazione e in aggiunta si assuma che il suo supporto intersechi $\partial K'$, dove l'intersezione fra ∂K e $\partial K'$ è una faccia $e \in \mathscr{E}_h^i$. Allora $(\hat{\mathcal{Q}}\eta)_{K'}$ può essere non banale su e, in generale. Ciononostante, questo non contraddice in alcun modo il fatto che $(\hat{\mathcal{Q}}\eta)_K = 0$ su e, visto che la funzione $\hat{\mathcal{Q}}\eta$ è **a due valori** su e.

2.4 I metodi ibridizzabili studiati

Nel seguito verranno descritti in dettaglio i tre metodi ibridi sviluppati nel codice di calcolo allegato alla tesi.
Per ciascuno di essi si darà adeguato spazio alla **selezione degli spazi approssimanti** e alla **scelta del risolutore locale** in ogni elemento della griglia computazionale.
Non verrà invece trattata la dimostrazione di esistenza e unicità della soluzione numerica, in quanto richiederebbe un'esposizione lunga e troppo articolata, che va oltre le necessità di questo lavoro di tesi. Naturalmente, per un approfondimento completo sotto questo aspetto, si può fare riferimento sempre a [CGL2009].

LDG-H E' necessario definire esplicitamente le tracce numeriche $\hat{\boldsymbol{q}}_h$ e $\hat{u}_h$; seguendo [CGL2009], s'impone:

$$\hat{u}_h = \begin{cases} \mathrm{P}_\partial\, g & \text{su } \partial K \cap \partial\Omega_D \\ \lambda_h & \text{altrimenti} \end{cases} \tag{2.13}$$

$$\hat{\boldsymbol{q}}_h = q_h + \tau\,(u_h - \hat{u}_h)\,\boldsymbol{n} \tag{2.14}$$

dove P_∂ denota una proiezione L^2 definita nel modo seguente: assegnata una generica funzione $\zeta \in L^2(\mathscr{E}_h)$ e un'arbitraria faccia $e \in \mathscr{E}_h$, la restrizione di $\mathrm{P}_\partial\,\zeta$ su e è definita come l'elemento di $M(e)$ tale da soddisfare

$$\langle \mathrm{P}_\partial\,\zeta - \zeta, \omega\rangle_e = 0, \qquad \forall \omega \in M(e). \tag{2.15}$$

La funzione di stabilizzazione τ è definita per ogni elemento $K \in \Omega_h$ in modo tale che τ sia non negativa su ∂K e costante all'interno di ogni generica faccia di ∂K. Si noti, inoltre, che per ogni $e = \partial K_e^+ \cap \partial K_e^-$ si ha, in linea generale, $\tau^{K_e^+}|_e \neq \tau^{K_e^-}|_e$.
Esiste una pluralità di scelte possibili sulla funzione di stabilizzazione; in particolare, possiamo sottolineare che:

- è possibile scegliere, su ciascun elemento della reticolazione, una sola faccia di bordo ove τ sia strettamente positiva, e nulla in tutte le altre. Questa opzione dà luogo ai cosiddetti metodi *single-face*;

- per ciascuna faccia sulla quale τ è non nulla, si può imporre un valore costante, oppure una proporzionalità inversa rispetto al diametro della faccia, ovvero al quadrato di tale diametro.
  Ciascuna scelta conduce ad una rapidità di convergenza numerica diversa.

- a parità di metodo risolutore, il comportamento esibito dalla soluzione è differente nel caso in cui sia la diffusione ovvero il trasporto a dominare; a seconda del regime, la τ potrà essere ottenuta come somma di due contributi, uno di natura "ellittica" (quando è la diffusione a dominare), e un secondo di natura "iperbolica", la cui scelta mostrerà ancora una volta l'intimo legame che i metodi *Discontinuous Galerkin*, applicati anche alle equazioni ellittiche, hanno conservato rispetto all'alveo convettivo in cui sono nati.

Gli spazi di polinomi sono i seguenti: $\boldsymbol{V}(K) = [\mathscr{P}_k(K)]^d$, $W(K) = \mathscr{P}_k(K)$ e $M(e) = \mathscr{P}_k(e)$.

RT-H Questo metodo è ottenibile utilizzando il metodo di Raviart-Thomas per il calcolo dei risolutori locali. I tre ingredienti del metodo RT-H sono i seguenti:

1. per ogni $K \in \Omega_h$, si sceglie

$$\hat{\mathcal{Q}}\mathsf{m} = \mathcal{Q}\mathsf{m} \qquad \hat{\mathcal{Q}}f = \mathcal{Q}f \qquad \text{su } \partial K$$

2. lo spazio ad elementi finiti $\boldsymbol{V}_h \times W_h$ è definito come lo spazio Raviart-Thomas di grado k:

$$\boldsymbol{V}(K) = [\mathscr{P}_k(K)]^d + \boldsymbol{x}\mathscr{P}_k(K) \qquad W(K) = \mathscr{P}_k(K) \qquad k \geq 0$$

3. come spazio di tracce approssimate si sceglie

$$M(e) = \mathscr{P}_k(e)$$

Da quanto indicato sopra possiamo trarre subito alcune conclusioni: innanzitutto, la funzione di stabilizzazione τ è in questo caso identicamente nulla. La *conservativity condition* ci assicura allora che $\hat{\boldsymbol{q}}_h$ concide con $\boldsymbol{q}_h$, e **dunque sarà la componente normale del flusso $\boldsymbol{q}_h$ ad essere continuo lungo le frontiere di separazione tra un elemento e quelli adiacenti**. Da ciò deduciamo che la variabile vettoriale $\boldsymbol{q}_h$ appartiene allo spazio funzionale

$$H(\mathrm{div}, \Omega) = \{\boldsymbol{v} \in \boldsymbol{L}^2(\Omega) : \mathrm{div}(\boldsymbol{v}) \in L^2(\Omega)\}$$

da cui l'eventuale possibilità di studiare l'abbattimento dell'errore rispetto alla norma propria di questo spazio.
Una seconda considerazione, anch'essa molto importante, fa riferimento a due delle caratteristiche per le quali i metodi ibridi riscuotono negli ultimi anni molto interesse nella letteratura scientifica: la sparsità della matrice globale e il rilassamento dei vincoli di continuità tra un elemento e l'altro.
In particolare, da quanto scritto finora si può notare che la soluzione ottenuta nel caso ibridizzato è la **stessa** di quella che si otterrebbe se si risolvesse il medesimo problema differenziale con **Raviart-Thomas con vincolo di continuità sulla componente normale dell'incognita vettoriale**. **Ciò che veramente cambia è la struttura delle funzioni di base impiegate per generare gli spazi approssimanti a dimensione finita**: nel caso ibridizzato, si parte da una formulazione in cui ogni elemento vive a sé, senza raccordi di continuità con quelli a lui adiacenti, e si scopre che **il soddisfacimento della *ulteriore* terza equazione *equivale* all'appartenenza ad $H(\mathbf{div}, \Omega)$**, che *a priori* **non** era stato scelto come requisito di appartenenza per le funzioni di base. Nel caso non ibridizzato, invece, come già suggerivano pionieristicamente gli autori in [AB1985], **i gradi di libertà sono in numero minore** (in quanto il raccordo continuo della componente normale li riduce) **ma producono un accoppiamento globale di difficile manipolazione algebrica**, in quanto si perdono completamente i vantaggi di sparsità.

BDM-H E' ottenibile utilizzando il metodo di Brezzi-Douglas-Marini per definire i tre ingredienti necessari all'ibridizzazione:

1. per ogni $K \in \Omega_h$, si sceglie

$$\hat{\mathcal{Q}}\mathsf{m} = \mathcal{Q}\mathsf{m} \qquad \hat{\mathcal{Q}}f = \mathcal{Q}f \qquad \text{su } \partial K$$

2. lo spazio ad elementi finiti $\boldsymbol{V}_h \times W_h$ è definito come:

$$\boldsymbol{V}_h = \mathscr{P}_k(K)^n \qquad W_h = \mathscr{P}_{k-1}(K) \qquad k \geq 1$$

3. come spazio di tracce approssimate si sceglie

$$M(e) = \mathscr{P}_k(e)$$

Anche per questo metodo valgono le medesime considerazioni su $\boldsymbol{q}_h$ viste in precedenza per l'ibridizzazione del metodo di Raviart-Thomas.

Capitolo 3

Design del codice C++

3.1 Filosofia generale e obiettivi

Il codice si basa su una libreria di classi richiamabili da un pacchetto *Octave*, che diventa a tutti gli effetti l'interfaccia cliente per chi utilizza la libreria stessa. La creazione degli oggetti necessari ad implementare un certo metodo numerico viene effettuata in una opportuna funzione del pacchetto, così come l'estrazione delle quantità necessarie all'assemblaggio delle matrici dei sistemi lineari.
Questo impianto comporta un elevato grado di non trasparenza tra l'implementazione della libreria e il codice scritto da chi ne fa uso, che dovrà, di volta in volta, **scegliere i tipi corretti da istanziare in funzione del metodo numerico e dei parametri risolutivi desiderati**. La suddivisione delle classi è basata in modo piuttosto naturale sui diversi ingredienti che concorrono a definire un certo schema numerico; una prima suddivisione di alto livello prevede:

- la parte del codice dedicata alla entità geometriche ed alla griglia di calcolo;
- la parte che implementa gli spazi di funzioni opportuni;
- la parte che contiene i dati e i metodi per il calcolo degli integrali numerici;
- la parte che permette di ottenere i parametri necessari alla visualizzazione con *Paraview*-VTK.

A ciascuna delle sezioni individuate corrisponde un ambito di visibilità proprio definito da quattro spazi di nomi, elencati nel listato (3.1); questo permette di evitare possibili conflitti tra i nomi definiti nelle quattro parti e garantisce, inoltre, una migliore e più netta individuazione delle macro-unità logiche in cui la libreria è organizzata.

Listato 3.1: Spazi di nomi definiti nel codice

```
1 namespace griglia_calcolo
2 namespace spazio_approx
3 namespace integrazione
```

```
4 namespace visualizzazione
```

Nel prosieguo del capitolo verranno descritte le principali scelte progettuali di ciascuna area semantica e, più specificatamente, di ciascuna classe.
Infine, per facilitare il più possibile la gestione della libreria da *Octave*, sono stati utilizzati, per matrici e vettori, tipi definiti negli *header file* di *Octave*, come **Matrix** per le matrici, **ColumnVector** per i vettori e **NDArray** per tutti i contenitori forniti in *output* al programma *Octave* chiamante.

3.2 Griglia di calcolo

3.2.1 Definizione delle entità geometriche: le classi Point, Forma, BordoElemento, Elemento, Triangolo, Tetraedro

Listato 3.2: Interfaccia della classe Point

```
1  class Point : public ColumnVector {
2  public:
3
4    Point() : ID_punto(0)
5    {
6        (*this).resize(DIM);
7                (*this)(DIM - 1) = 0.;
8    }
9
10   Point(const int & ID) : ID_punto(ID)
11   {
12               (*this).resize(DIM);
13       (*this)(DIM - 1) = 0.;
14   }
15
16   Point(const Point & p) : ID_punto(p.getID_punto())
17   {
18               (*this).resize(p.numel());
19
20       for (int i = 0; i < p.numel(); i++)
21               (*this)(i) = p(i);
22   }
23
24   Point & operator = (const Point & p)
25
26   {
27       if (this != &p)
28       {
29                       ID_punto = p.getID_punto();
```

```
30
31                          for (int i = 0; i < p.numel(); i++)
32                                  (*this)(i) = p(i);
33          }
34
35          return (*this);
36      }
37
38      inline int getID_punto() const
39
40      {
41          return ID_punto;
42      }
43
44          inline void setID_punto(const int & num)
45            {
46                    ID_punto = num;
47            }
48
49    private:
50
51      int ID_punto;
52    };
```

La classe Point consente di gestire un generico punto del dominio di calcolo. Essa deriva pubblicamente da **ColumnVector** di *Octave* (di cui, naturalmente, eredita gli attributi e gli operatori, tra i quali il prodotto matrice-vettore con una qualsivoglia **Matrix**) e ha per costruzione dimensione pari a quella del dominio di calcolo, identificato da un'opportuna macro.

Oltre ai costruttori (di *default*, di copia e con parametro per l'indice) e al sovraccaricamento dell'operatore di assegnazione, essa possiede come attributo privato un intero, che serve ad identificare univocamente il punto: tale valore sarà riempito con quello assegnato dal generatore di griglia e sarà utile per confrontare due entità geometriche di bordo. Si noti, da ultimo, che sono presenti due opportuni metodi che consentono l'assegnamento e l'estrazione dell'attributo privato.

Listato 3.3: Interfaccia della classe astratta Forma

```
1
2 typedef std::vector<Point> point_vector_type;
3
4 enum tipo_bc { Interna = -1, Dirichlet = 0, Neumann = 1, Robin = 2};
5
6 class BordoElemento;
7 class Elemento;
8
```

```
9  typedef boost::shared_ptr<BordoElemento> border_pointer_type;
10
11   class Forma {
12   public:
13
14         Forma();
15         Forma(int const & nP);
16         Forma(const Forma & s);
17     virtual ~Forma() {}
18
19     void set_numPunti(int const & nP);
20         void set_ID(int const & _ID) const;
21
22         int get_numPunti() const;
23         int get_ID() const;
24
25         virtual int get_dimForma () const = 0;
26
27         Point get_point(int const & i) const;
28         void set_point(int const & i, Point const & p);
29
30     virtual void set_detJac() = 0;
31     inline Real get_detJac () const { return detJac; }
32
33   protected:
34
35     Real detJac;
36
37   private:
38
39     mutable int ID;
40     int numPunti;
41     point_vector_type Vett_punti;
42 };
```

La classe astratta Forma rappresenta l'interfaccia di più alto livello per la gestione delle entità geomtriche del problema.

Essa contiene tutte le proprietà comuni: un intero per l'identificazione univoca, un intero per il numero di punti che la compongono e un contenitore atto a memorizzare tali punti. Questi ultimi tre attributi sono privati, in quanto non c'è necessità che vengano modificati da metodi delle classi derivate.

Un'attenzione particolare deve essere prestata all'intero ID: esso è stato dichiarato **mutable** perchè possa essere modificato anche da un metodo dichiarato **const**. Ecco spiegata la controintuitiva dichiarazione del metodo set_ID() che interviene per modificarlo.

L'attributo protetto detJac rappresenta il modulo dello Jacobiano di trasformazione e –

come noto dalla teoria dell'integrazione su domini di riferimento – permette di tenere traccia dei cambiamenti di misura (aree, volumi) che avvengono passando da un'entità trasformata a quella canonica[1]. Per precisione, tale quantità corrisponde al **rapporto di dilatazione superficiale o volumetrica tra la generica forma geometrica nel dominio fisico e la corrispondente forma nella configurazione di riferimento**.
Il metodo set_detJac() è naturalmente **virtuale puro**, in quanto definibile soltanto per ciascuna figura geometrica derivata.

Listato 3.4: Interfaccia della classe astratta BordoElemento

```
1  class BordoElemento : public Forma {
2
3          public:
4
5          BordoElemento(int const & nP);
6          BordoElemento(const BordoElemento & be);
7
8          virtual int get_dimForma () const
9          {
10                 return DIM - 1;
11         }
12
13     virtual void set_Mappa2D() = 0;
14     Matrix get_Mappa2D() const;
15
16     virtual ColumnVector normale() const = 0;
17
18     void set_ElementoSx (int const & sx) const;
19     void set_ElementoDx (int const & dx) const;
20
21     int get_ElementoSx() const;
22     int get_ElementoDx() const;
23
24     virtual Real get_diametro() const = 0;
25
26     void set_boundary_condition (const tipo_bc & _bc);
27     tipo_bc get_boundary_condition () const;
28
29     protected:
30
31     Matrix Mappa2D;
32
```

[1] Si ricordi che non esiste un solo elemento di riferimento, ma sono possibili più scelte. Quella implementata nel codice di calcolo è la più comune per triangoli e tetraedri, vale a dire il triangolo isoscele di lato 1, rettangolo nell'origine degli assi, e il tetraedro avente per vertici l'origine di $\mathbb{R}^3$ e i tre punti con coordinate pari a quelle dei tre versori della base canonica. Per ulteriori dettagli, si veda in appendice, A.2.

```
33      private:
34
35      mutable int ElementoSx;
36      mutable int ElementoDx;
37      tipo_bc boundary_condition;
38  };
```

Listato 3.5: Interfaccia della classe astratta Elemento

```
1  class Elemento : public Forma {
2
3          public:
4
5          Elemento(int const & nP);
6          Elemento(const Elemento & be);
7
8          virtual int get_num_facce_per_elemento () const = 0;
9
10         virtual int get_dimForma () const
11         {
12                 return DIM;
13         }
14
15         virtual void set_Mappa() = 0;
16         Matrix get_Mappa() const;
17         Matrix get_Mappa_Piola () const;
18
19         virtual border_pointer_type faccia (int const & i) const = 0;
20
21         protected:
22
23         Matrix Mappa;
24 };
```

Le due classi BordoElemento e Elemento derivano pubblicamente da Forma; sono ancora classi astratte, ma mettono in luce un livello di astrazione molto importante: **si vuole dividere gli oggetti geometrici che hanno dimensione pari a quella del dominio computazionale**, da quelli che vivono su **varietà con dimensione inferiore di un'unità**, e che sono – proprio per natura dei metodi discontinui misti ibridi – il **dominio di definizione degli spazi funzionali per l'incognita ibrida**.

Entrambe le classi possiedono un attributo privato per la memorizzazione della mappa di trasformazione rispetto al corrispondente dominio di riferimento, nonchè un metodo virtuale (in esse ridefinito) che permette l'estrazione della dimensione della forma geometrica. Si noti che ogni istanza di BordoElemento possiede opportuni campi privati per memorizzare le condizioni al contorno e la numerazione dei due elementi di cui tale figura è interfaccia

di separazione.

Da ultimo, per ogni oggetto di Elemento si ha la possibilità di ottenere sia un **puntatore a ciascuna delle facce di bordo**, sia la **mappa di Piola**, trasformazione fondamentale che permette di conservare le componenti normali delle tracce di bordo delle funzioni vettoriali, e risulta determinante per passare correttamente da integrali sull'elemento di riferimento al generico elemento trasformato. Per una trattazione matematica più dettagliata, per l'ordinamento locale delle facce si consulti l'appendice A.2.1, mentre riguardo alla trasformazione di Piola si faccia riferimento alla sezione A.3.

Listato 3.6: Interfaccia della classe Triangolo

```
class Triangolo : public BordoElemento {
  public:

        Triangolo (bool riferimento = false);
        Triangolo (const Point & P0, const Point & P1, const Point & P2);
        Triangolo (const Triangolo & t);

    virtual void set_detJac();
    virtual void set_Mappa2D();
    virtual ColumnVector normale() const;
};
```

Listato 3.7: Interfaccia della classe Tetraedro

```
 class Tetraedro : public Elemento {

        public:

        Tetraedro (bool riferimento = false);
        Tetraedro (const Point & P0,
         const Point & P1, const Point & P2, const Point & P3);
        Tetraedro (const Tetraedro & t);

        virtual int get_num_facce_per_elemento () const
               {
                 return 4;
               }

        virtual void set_detJac();
        void set_Mappa();
        virtual border_pointer_type faccia(int const & i) const;

        static const int connectivity[4][3];
};
```

Le ultime due classi da considerare sono quelle relative alla creazione di tetraedri e triangoli. Tutti i metodi virtuali puri sono ridefiniti, in modo da **rendere queste due classi istanziabili**; ciascuna classe possiede tre costruttori, in cui il primo è di *default* (con la possibilità di creare l'oggetto di riferimento inserendo **true** al booleano opzionale passato al costruttore). Il secondo costruttore è quello che permette di costruire la forma geometrica vera e propria, a partire dai punti del dominio che la identificano univocamente.
Nella classe Tetraedro è presente un attributo pubblico **static const** che consente di estrarre correttamente tutte le facce di bordo, secondo la convenzione indicata nell'appendice A.2.1.

3.2.2 La classe Mesh

Listato 3.8: Interfaccia della classe astratta Mesh

```
1  class Mesh {
2
3  public:
4          virtual ~Mesh() {}
5          void set_numNodi (const int & num_nodi);
6          void set_numElementi (const int & num_el);
7          void set_numFacce (const int & num_f);
8          int get_numNodi () const;
9          int get_numElementi () const;
10         int get_numFacce () const;
11
12         Point get_point (const int & i) const;
13          element_pointer_type get_elemento (const int & i) const;
14          border_pointer_type get_faccia (const int & i) const;
15
16         static const element_pointer_type elementoRif;
17         static const border_pointer_type bordoRif;
18
19         Matrix get_connettivita () const;
20
21         void printInfo () const;
22
23         virtual void set_Facce () = 0;
24
25 protected:
26
27         point_vector_type M_punti;
28         element_vector_type M_elementi;
29         border_vector_type M_facce;
30         Matrix connettivita;
31
32   private:
```

```
33
34          int numNodi;
35          int numElementi;
36          int numFacce;
37    };
```

3.2.3 La classe MeshGMSH

Listato 3.9: Interfaccia della classe MeshGMSH

```
1  class MeshGMSH : public Mesh
2
3  {
4    public:
5
6          MeshGMSH(const char * nomeFile);
7          virtual ~MeshGMSH() {}
8
9          virtual void set_Facce ();
10
11   private:
12
13         border_vector_type M_facce_bordo;
14 };
```

La trattazione della parte geometrica si conclude con le classi Mesh e MeshGMSH derivata pubblicamente dalla precedente.
Nella classe base sono presenti i tradizionali contenitori per archiviare tutte le informazioni necessarie (nodi geometrici, elementi e facce), con i relativi metodi per estrazione e assegnamento delle dimensioni corrette. Sono anche presenti un contenitore per la connettività (necessario per associare a ciascuna faccia di ciascun tetraedro la corrispondente numerazione assoluta delle facce, pensate come dominio di definizione della variabile ibrida e non invece come porzione della frontiera di un generico elemento della griglia) e due oggetti **static const** per i domini di riferimento, utilissimi nel calcolo degli integrali.
Nella classe derivata, che implementa la lettura della griglia a partire da un opportuno *file* in formato GMSH, è presente un attributo addizionale, relativo alle facce di bordo (vale a dire, non interne). Esso è intimamente associato al formato richiesto al *file* passato in ingresso al costruttore (si veda, a tale proposito, l'appendice A.5).

3.3 Informazioni per la visualizzazione

Listato 3.10: Interfaccia della classe InfoPlot

```
1  class InfoPlot
2
3  {
```

```
4
5  public:
6
7          InfoPlot(const int & dimDominio, const int & gradoGlobale,
8           const int & gradoLocale = 1);
9
10         vector < vector <int> > swap_cube
11         (const vector<int> & valori,
12                 const vector < vector <int> > & rifCutCube) const;
13
14         void print_info() const;
15
16         inline int get_numSottoelementi () const
17         { return adiacenze.size(); }
18
19         inline int get_puntiperSottoelemento () const
20         { return adiacenze[0].size(); }
21
22         inline int get_numPuntiplot () const
23         { return punti_plot.size(); }
24
25         inline Point get_puntoplot (const int & i) const
26         { return punti_plot[i]; }
27
28         inline int get_ind_Paraview () const
29         { return Paraview_index; }
30
31 private:
32
33         point_vector_type punti_plot;
34         vector < vector <int> > adiacenze;
35         int Paraview_index;
36
37 };
```

La classe InfoPlot consente di calcolare tutte le quantità necessarie per scrivere in formato XML gli ingredienti matematici utili per una corretta visualizzazione con *Paraview*.
Per poter rappresentare con *Paraview* elementi finiti con grado elevato (3 o 4, ad esempio), è obbligatorio proiettare la soluzione su uno spazio polinomiale a grado inferiore (1 o 2, le scelte attualmente disponibili), andando contemporaneamente a suddividere gli elementi della griglia di calcolo (nel nostro caso, i tetraedri e i triangoli) in sottoelementi non fisici, funzionali solamente alla visualizzazione.
Il costruttore della classe riceve in ingresso la dimensione del dominio (2 o 3, nella corrente implementazione, qui certamente migliorabile da un possibile estensore), il grado globale della funzione polinomiale da visualizzarsi e il grado locale (i.e., su ciascun sottoelemento

fittizio) di visualizzazione.
I contenitori privati archiviano tutti i punti necessari ad una corretta visualizzazione (definiti per comodità sull'elemento geometrico di riferimento), la connettività per ognuno dei sottoelementi e l'indice corretto secondo la nomenclatura VTK (esso è identico per tutti i sottoelementi).
I metodi della classe consentono di estrarre con sicurezza le quantità necessarie.
Il metodo swap_cube() viene utilizzato nel costruttore per operare un'intelligente trasformazione tra sottoelementi aventi la medesima forma, ma che differiscono tra loro solamente per la numerazione assoluta dei punti.

3.4 Spazi di funzioni approssimanti

Le funzioni di base sono, nell'implementazione corrente, i polinomi di grado minore o uguale ad un intero non negativo k (lo spazio $\mathscr{P}_k$). La base utilizzata per generare tale spazio è quella dei monomi di grado corrispondente: questa scelta, tra le molte possibili, permette di implementare i contenitori delle funzioni e i metodi per calcolarne il valore in un generico punto in modo particolarmente semplice, intuitivo, ed al contempo efficace.
Nell'ambito dei metodi misti ibridi, si ha sempre la necessità di approssimare tre spazi funzionali (si veda il Capitolo 2), corrispondenti alle tre variabili che compaiono nello schema numerico:

- uno spazio per il campo scalare **potenziale**,
- uno spazio per il campo vettoriale **flusso**,
- uno spazio per la variabile **ibrida**, anch'esso a valori scalari.

Si noti che colui che implementa uno schema numerico può scegliere il tipo di approssimazione a lui necessaria, in funzione sia della forma degli elementi della griglia sia del tipo di metodo numerico utilizzato.

3.4.1 La classe SpazioFunzionale

Sulle premesse teoriche appena esposte si basa l'implementazione della gerarchia a cui fa capo la classe SpazioFunzionale; il suo *header file* è contenuto nel listato (3.11).

Listato 3.11: Interfaccia della classe astratta SpazioFunzionale

```
1 class SpazioFunzionale{
2
3 public:
4
5         SpazioFunzionale(const int & _k);
6         virtual ~SpazioFunzionale() {}
7
8         void set_k(const int & _k);
```

```
9           inline int get_k() const { return k; }
10
11          virtual void set_dimDominio() = 0;
12          inline int get_dimDominio() const { return dimDominio; }
13
14          virtual void set_numComp() = 0;
15          inline int get_numComp() const { return numComp; }
16
17          virtual void set_numGdl() = 0;
18          inline int get_numGdl() const { return numGdl; }
19
20          Matrix get_monomi() const;
21
22          virtual void set_fun() = 0;
23          ColumnVector val_fun(const Point & p,
24                  const int & i) const;
25
26  protected:
27
28          int k;
29
30          int dimDominio;
31          int numComp;
32
33          int numGdl;
34
35          Matrix Fun;
36  };
```

Si tratta di una classe **virtuale pura**, grazie alla presenza dei metodi set_dimDominio(), set_numGdl() e set_fun(): tali funzioni servono a definire gli attributi che contengono, con naturale corrispondenza dei nomi, la dimensione del dominio di definizione delle funzioni, il numero di gradi di libertà (cioè il numero di funzioni presenti) e una matrice (Fun) che conterrà gli esponenti dei monomi associati a ciascuna variabile spaziale. Tale classe contiene dunque tutte le quantità fondamentali di uno spazio di funzioni, unitamente ai metodi per la loro lettura e scrittura: derivando opportunamente da essa ed ampliandone le funzionalità, è possibile implementare un qualsivoglia spazio funzionale a dimensione finita.

Il ruolo fondamentale per la scelta della base degli spazi che verranno successivamente utilizzati è svolto dal metodo get_monomi(), che gestisce in modo non trasparente all'utente una chiamata ad *Octave* per calcolare gli esponenti dei monomi che costituiscono le funzioni di base di grado non superiore a k, dove l'intero k è inserito dall'utente e gestito successivamente con un passaggio per referenza costante (scelta tipica in questi casi) dal costruttore della classe. L'implementazione è basata sul fatto che, dato $0 \leq \tilde{k} \leq k$, è ne-

cessario elencare tutte le *combinazioni con ripetizione* di $\tilde{k}$ a gruppi di cardinalità pari alla dimensione spaziale. Uno stralcio della routine di *Octave* che si occupa di ciò è contenuta nel listato (3.12)

Listato 3.12: Descrizione della funzione calc_partizioni

```
function [result] = calc_partizioni (grado, dim)

% function [result] = calc_partizioni (grado, dim)
% Funzione che calcola la base dei monomi
% per uno spazio funzionale di polinomi scalari
% di grado esattamente pari al
% primo ingresso (che deve essere >= 1);
% il secondo ingresso rappresenta invece il numero di coordinate
% indipendenti per costruire la base dei monomi.
% L'output è organizzato in una matrice, in cui ciascuna funzione
% di base corrisponde ad una riga,
% mentre ciascuna colonna (numero totale pari a dim)
% rappresenta l'esponente associato alla rispettiva coordinata
% messa a fattore in quel monomio-f.d.b.
%
% La funzione si basa sul pacchetto aggiuntivo combinatorics,
% che consente di calcolare le partizioni di un qualsivoglia
% intero positivo.

partizioni = partint(grado);
num_righe = size(partizioni)(1);
num_colonne = size(partizioni)(2);
```

Il comando da sottolineare in modo particolare è quello che utilizza la funzione **partint** del pacchetto *Combinatorics* di *Octave*: a partire da tutte le possibili partizioni di un numero intero è sufficiente un semplicissimo *post-processing* per ottenere gli esponenti dei monomi desiderati. Tale facilità giustifica appieno l'utilizzo di una funzione *Octave* in luogo di una riscrittura completa del codice in C++, tantopiù che si tratta di una procedura sì elementare, ma estremamente tecnica e dunque ad alto rischio di errori.
Infine, gli attributi sono classificati **protected** in modo da essere accessibili dalle classi derivate.

3.4.2 Le classi SpazioFunzionalePotenziale, SpazioFunzionaleFlusso e SpazioFunzionaleIbrida

Le classi SpazioFunzionalePotenziale, SpazioFunzionaleFlusso e SpazioFunzionaleIbrida sono derivate tramite ereditarietà pubblica da SpazioFunzionale. Esse costituiscono l'interfaccia da cui sarà poi possibile derivare i tipi che concretamente serviranno ad eseguire i calcoli che coinvolgono le funzioni di base. A questo livello della gerarchia i tipi rimangono astratti,

ma è possibile definire i metodi (virtuali puri in SpazioFunzionale) che definiscono la dimensione del dominio delle funzioni e il loro numero di componenti: il potenziale, la variabile ibrida e il flusso sono infatti campi scalari, i primi, e vettoriale, l'ultimo.
E' di tutta evidenza che **questo tipo di derivazione particolarizza fortemente gli spazi di funzioni effettivamente implementabili, rendendo quest'area del codice inscindibile dall'uso di metodi misti (dunque a due campi) e in più ibridi (con la presenza della variabile ibrida)**. Inoltre vengono aggiunti metodi e attributi che rispecchiano le manipolazioni che avvengono sulle tre variabili nello schema numerico, e che sono ovviamente indipendenti dalla modalità con cui lo spazio viene effettivamente approssimato.

Listato 3.13: Interfaccia della classe astratta SpazioFunzionalePotenziale

```
1  class SpazioFunzionalePotenziale : public SpazioFunzionale {
2
3  public:
4
5          SpazioFunzionalePotenziale(const int & _k);
6          virtual ~SpazioFunzionalePotenziale() {}
7
8          void set_dimDominio() { dimDominio = DIM; }
9
10         void set_numComp() { numComp = 1; }
11
12         ColumnVector
13         val_funBordo(const Point & p, const int i,
14                 const int & ID_faccia) const;
15
16 };
```

Nel caso del flusso, in SpazioFunzionaleFlusso (listato (3.14)), compaiono i metodi per calcolare la divergenza di un campo vettoriale e il suo valore al bordo.

Listato 3.14: Interfaccia della classe astratta SpazioFunzionaleFlusso

```
1  class SpazioFunzionaleFlusso : public SpazioFunzionale {
2
3  public:
4
5          SpazioFunzionaleFlusso(const int & _k);
6          virtual ~SpazioFunzionaleFlusso() {}
7
8          void set_dimDominio() { dimDominio = DIM; }
9
10         void set_numComp() { numComp = DIM; }
11
12         ColumnVector val_funBordo
```

```
13          (const Point & p, const int i, const int & ID_faccia) const;
14
15          Real val_divFun(const Point & p, const int i) const;
16 };
```

Listato 3.15: Interfaccia della classe astratta SpazioFunzionaleIbrida

```
1 class SpazioFunzionaleIbrida : public SpazioFunzionale {
2
3 public:
4
5          SpazioFunzionaleIbrida(const int & _k);
6          virtual ~SpazioFunzionaleIbrida() {}
7
8          void set_dimDominio() { dimDominio = DIM - 1; }
9
10         void set_numComp() { numComp = 1; }
11
12 };
```

3.4.3 Le classi SpFunPotenzialePk, SpFunIbridaPk, SpFunFlussoPk e SpFunFlussoRTk

Dati i due livelli di ereditarietà appena esposti per gli spazi funzionali, non rimane ora che derivare gli opportuni tipi necessari per implementare i metodi che abbiamo esposto; chiaramente la gerarchia è aperta ad ampliamenti, nel senso che che viene lasciata a futuri estensori la possibilità di derivare spazi approssimanti diversi, per implementare nuovi metodi, ma anche semplicemente per approssimare in modo differente gli spazi (su esaedri invece che su tetraedri, sarebbe ad esempio necessario utilizzare $\mathscr{Q}_k$). Nella versione corrente, sono presenti solo i tipi strettamente necessari: $\mathscr{P}_k$ per la variabile ibrida e per il potenziale (con la dimensione spaziale opportuna) e $[\mathscr{P}_k]^n$ e RT_k per la variabile flusso. Ovviamente il metodo fondamentale, virtuale puro in SpazioFunzionale, è set_fun(), che è deputato a riempire il contenitore Fun, contenente gli esponenti che definiscono i monomi base dello spazio vettoriale. Inoltre a questo livello può essere effettuato il calcolo del numero di gradi di libertà degli spazi di volta in volta definiti. In realtà l'implementazione di questo metodo è assai semplice e prevede delle manipolazioni elementari sul contenitore estratto dal metodo get_monomi() (si veda il listato (3.11)), anche nel caso dello spazio RT_k, in cui è necessario aggiungere i gradi di libertà aggiuntivi rispetto a $[\mathscr{P}_k]^n$ secondo la filosofia di Raviart-Thomas.

3.5 Implementazione del contenitore per l'integrazione numerica

3.5.1 La classe RegolaQuad

La classe RegolaQuad manipola i nodi e i pesi necessari all'integrazione numerica.
I parametri da passare al costruttore della classe sono due: la dimensione spaziale della forma geometrica per la quale si vuole utilizzare la regola di quadratura (che potrà corrispondere alla macro **DIM** oppure a **DIM** -1) e il grado di esattezza (selezionabile dall'utente in base al grado polinomiale scelto e ai *files* attualmente a disposizione). La classe è basata sulla classe QuadPoint, che deriva pubblicamente da Point e permette di memorizzare congiuntamente le coordinate spaziali del punto e il peso di integrazione ad esso associato.
L'interfaccia di QuadPoint è contenuta nel listato (3.16).

Listato 3.16: Interfaccia della classe QuadPoint

```
1  class QuadPoint: public Point {
2
3  public:
4
5      QuadPoint() : peso(0) {}
6
7      QuadPoint(const Real & w) : peso(w) {}
8
9      QuadPoint (const QuadPoint & qp) : Point(qp), peso(qp.get_peso())
10     {}
11
12     inline Real get_peso() const { return peso; }
13
14     inline void set_peso (const Real & w) { peso = w; }
15
16 private:
17
18     Real peso;
19 };
```

Il tipo QuadPoint viene poi utilizzato in un vettore (come nel listato (3.17)) per contenere nodi e pesi di integrazione. In questo caso è stato utilizzato il tipo vector della libreria standard.

Listato 3.17: Interfaccia della classe RegolaQuad

```
1  class RegolaQuad {
2
3  public:
4
```

```
5          RegolaQuad (const int & dim, const int & _gradoEsattezza);
6
7          void set_gradoEsattezza(const int & g);
8
9          inline int get_gradoEsattezza() const { return gradoEsattezza; }
10
11         void set_nodiPesi(const char * dati_quadratura, const int & dim);
12
13         inline QuadPoint get_nodiPesi(const int & i) const
14         { return nodiPesi[i]; }
15
16         inline int get_numNodi() const { return numNodi; }
17
18         inline Matrix get_permutazioni() const { return permutazioni; }
19
20 private:
21
22         std::vector<QuadPoint> nodiPesi;
23         int gradoEsattezza;
24         int numNodi;
25         Matrix permutazioni;
26 };
```

E' degno di nota l'attributo privato permutazioni: esso è di tipo **Matrix** e raccoglie le permutazioni degli indici dei nodi di quadratura, rispetto a corrispondenti permutazioni della n-pla di vertici che rappresenta univocamente la forma geometrica sulla quali la formula di quadratura trova applicazione.
L'insieme dei nodi di quadratura utilizzati in questo codice è frutto del lavoro di Ronald Cools (http://nines.cs.kuleuven.be/ecf/), e gode di una particolare proprietà: fissata una forma geometrica di riferimento (il tetraedro o il triangolo, per esempio), i nodi di quadratura garantiscono un certo grado di esattezza; qualora la forma geometrica venga considerata secondo un'altra orientazione, vale a dire **permutando l'ordine dei vertici che la identificano univocamente**, per ottenere il nuovo, corretto *set* di nodi di quadratura **è sufficiente operare una permutazione degli indici dei nodi di partenza**. Se l'entità geometrica sulla quale si sta analizzando la formula di quadratura è identificata univocamente da n punti, allora le possibili, diverse orientazioni sono $n!$; in corrispondenza di ciascuna di queste permutazioni, il *file* contenente nodi e pesi indica qual è la corrispondente permutazione dei nodi da effettuare.
Questa procedura è stata utilizzata **solamente nel caso 2D per i triangoli**, essendo essi le frontiere di separazione fra due tetraedri adiacenti: **una stessa entità geometrica triangolare verrà considerata da ciascuno dei due tetraedri con una diversa orientazione** (frutto dell'ordinamento locale indicato in appendice A.2.1), **a cui corrisponderà una diversa permutazione della terna di vertici del triangolo**. La

Matrix permutazioni viene riempita soltanto per i triangoli ($n = 3$); ha numero di righe pari alle $n! = 3! = 6$ permutazioni possibili, considerate in ordine lessicografico (vale a dire, $ABC, ACB, BAC, BCA, CAB, CBA$).

3.6 Definizione dei metodi numerici

Nell'implementazione corrente la classe MetodoNumerico costituisce un contenitore che compone le diverse parti necessarie a definire uno schema numerico, dalla griglia di calcolo fino agli spazi funzionali e ai tipi per la quadratura numerica.
L'altra componente fondamentale è la definizione di metodi che estraggono dalla libreria le quantità disponibili da *prompt* di *Octave*, in modo del tutto non trasparente all'utente finale. Il tipo di uscita è sempre **NDArray**, come si è già evidenziato, per facilitare la simbiosi con *Octave* stesso. Si noti inoltre che, tra gli attributi privati, **gli spazi funzionali e la griglia sono memorizzati in puntatori alle rispettive classi astratte, in modo da poter contenere, grazie al loro comportamento polimorfico, qualunque tipo derivato selezionabile dall'utente, e garantendo così la possibilità di estensioni del codice fornito**.

3.6.1 La classe MetodoNumerico

Listato 3.18: Interfaccia della classe MetodoNumerico

```
1  class MetodoNumerico  {
2
3  public:
4
5          MetodoNumerico(const Metodo & _IDmetodo,
6          const int & _k_fl, const int & _k_pt, const int & _k_ib,
7          const boost::shared_ptr<SpazioFunzionaleFlusso> _Flusso,
8          const boost::shared_ptr<SpazioFunzionalePotenziale> _Potenziale,
9          const boost::shared_ptr<SpazioFunzionaleIbrida> _Ibrida,
10         const boost::shared_ptr<Mesh> _griglia,
11         const int & _grado_Esattezza_el, const int & _grado_Esattezza_bordo,
12         const int & _grado_plot_flusso, const int & _grado_plot_potenziale,
13         const int & _grado_plot_ibrida);
14
15         inline void set_IDmetodo(const Metodo & _IDmetodo)
16         { IDmetodo = _IDmetodo; }
17         inline Metodo get_IDmetodo() const
18         { return IDmetodo; }
19
20         NDArray get_cont_connettivita() const;
21         NDArray get_cont_conn_permut() const;
22         NDArray get_cont_det3D() const;
```

```
23          NDArray get_cont_det2D() const;
24          NDArray get_cont_mappe3D() const;
25          NDArray get_cont_bc_facce() const;
26          NDArray get_cont_permut_facce() const;
27          NDArray get_nodi_pesi_quad_el() const;
28          NDArray get_nodi_pesi_quad_bordo() const;
29          NDArray get_permut_nodi_bordo() const;
30          NDArray get_diam_facce() const;
31          NDArray get_cont_flusso_elemento() const;
32          NDArray get_cont_flusso_bordo_normale() const;
33          NDArray get_cont_flusso_divergenza() const;
34          NDArray get_cont_potenziale_elemento() const;
35          NDArray get_cont_potenziale_bordo() const;
36          NDArray get_cont_potenziale_grad_el() const;
37          NDArray get_cont_ibrida() const;
38          NDArray get_punti_plot_flusso() const;
39          NDArray get_punti_plot_potenziale() const;
40          NDArray get_punti_plot_ibrida() const;
41          NDArray get_valore_fdb_punti_plot_flusso() const;
42          NDArray get_valore_fdb_punti_plot_potenziale() const;
43          NDArray get_valore_fdb_punti_plot_ibrida() const;
44          int get_ind_Paraview_flusso() const;
45          int get_ind_Paraview_potenziale() const;
46          int get_ind_Paraview_ibrida() const;
47
48  private:
49
50          Metodo IDmetodo;
51
52          boost::shared_ptr<Mesh> griglia;
53
54          boost::shared_ptr<SpazioFunzionalePotenziale> Potenziale;
55          boost::shared_ptr<SpazioFunzionaleFlusso> Flusso;
56          boost::shared_ptr<SpazioFunzionaleIbrida> Ibrida;
57
58          RegolaQuad IntegrazioneElemento;
59          RegolaQuad IntegrazioneBordoElemento;
60
61          InfoPlot Plot_flusso;
62          InfoPlot Plot_potenziale;
63          InfoPlot Plot_ibrida;
64  };
```

3.7 Generazione dell'archivio `libreriaEFMistiIbridi.a`

Tutte le procedure, dal *pre-processing* al *linking* del codice, dall'installazione alla disinstallazione, sono automatizzate grazie all'utilità *make*, quasi sempre presente sui sistemi basati sul paradigma *UNIX*. Essa utilizza una serie di *files* detti – almeno negli usi più comuni – *Makefile*, che codificano un albero delle dipendenze per il codice sorgente e i *files* di intestazione, oltre alle regole da seguire per aggiornare i *targets* definiti.
In ciascuna cartella in cui è contenuto il codice è presente un *Makefile* che si occupa di gestirne il contenuto.
Nella cartella `include` sono presenti gli *header files*, che vengono installati nel sistema in una locazione configurabile dall'utente nel file `make.config`; nella cartella `src` è presente il codice sorgente, con il relativo *Makefile*, che serve a permetterne la compilazione, il *linking* in un **archivio** (libreria statica) e l'installazione in una cartella di sistema ancora selezionabile dell'utente in `make.config`: questi *Makefile* sono richiamati ricorsivamente da un altro *Makefile* di livello superiore che si trova nella cartella in cui viene fornito il codice, insieme al *file* in cui sono definite le variabili utilizzate da *make*, il già citato `make.config`.

3.8 Un esempio di funzione che utilizza la libreria

Nella cartella `FE_MxHyb-1.0.0` è presente tutto il necessario per creare un file `.oct` e installare il pacchetto che utilizza la libreria `libreriaEFMistiIbridi.a` già installata: in questo modo il codice generato diventa a tutti gli effetti parte integrante di *Octave*.
Come appena anticipato, il *file* `.oct` generato nel pacchetto permette di utilizzare la libreria direttamente dalla linea di comando di *Octave*, con tutti gli ovvi vantaggi che ne conseguono: tra questi, avere a disposizione dei comandi su un *prompt*, e la possibilità di usare un linguaggio interpretato, che rendono l'uso della libreria intuitivo e semplicissimo.
Per estendere, nel senso più proprio del termine, le funzionalità di *Octave* con quelle fornite dalla nostra libreria, è necessario scrivere una funzione, che verrà tradotta in un *file* `.oct` e resa visibile da *Octave* grazie alle regole contenute nel *Makefile* del pacchetto (in particolare grazie allo script `mkoctfile` fornito con *Octave*). In questa funzione l'utente potrà, importando gli spazi di nomi e i *files* di intestazione della nostra libreria, utilizzarne i tipi per costruire gli oggetti di cui necessita, ed estrarre i risultati richiesti.
Un esempio di una funzione di questo tipo è contenuta nel listato (3.19): il punto di ingresso al codice richiamato da *Octave* è sancito dalla macro **DEFUN_DLD** che introduce il nome della funzione (obbligatoriamente uguale a quello del *file* in cui è contenuta), i parametri di ingresso e delle istruzioni, visibili da *Octave* lanciando da *prompt*: `help metodo`.

Listato 3.19: Esempio di funzione che permette la chiamata da *Octave* della libreria

```
1 DEFUN_DLD (metodo, args, ,
```

```
2
3         "Funzione che riceve in ingresso, ordinatamente:\n"
4         "- nome del metodo: LDG-H, BDM-H, RT-H \n"
5         "- grado dei polinomi per la variabile FLUSSO,\n"
6         "- grado dei polinomi per la variabile POTENZIALE,\n"
7         "- grado dei polinomi per la variabile IBRIDA,\n"
8         "- nome del file che contiene la griglia di calcolo,\n"
9         "- grado di esattezza per l'integrazione 3-D (facoltativo),\n"
10        "- grado di esattezza per l'integrazione 2-D (facoltativo),\n."
11        "- grado per la visualizzazione del flusso (facoltativo),\n"
12        "- grado per la visualizzazione del potenziale
13              (facoltativo),\n"
14        "- grado per la visualizzazione della variabile ibrida
15              (facoltativo),\n"
16        "- grado per la visualizzazione del flusso ricostruito
17              (facoltativo),\n"
18        "- grado per la visualizzazione del potenziale ricostruito
19              (facoltativo),\n"
20        "I parametri opzionali vanno inseriti tutti in blocco
21              oppure nessuno."
22        "\n"
23        "Restituisce:\n"
24        "- la matrice di connettivita' tra elementi e facce di bordo,\n"
25        "- indici (in ordine locale) dei vertici per faccia, per elemento, \n"
26        "- determinante delle mappe di trasformazione 3-D,\n"
27        "- determinante delle mappe di trasformazione 2-D,\n"
28        "- mappe di trasformazione 3-D, \n"
29        "- condizioni al contorno per ciascuna faccia assoluta, \n"
30        "- indici dei vertici per faccia, in numerazione assoluta, \n"
31        "- nodi e pesi di quadratura 3D,\n"
32        "- nodi e pesi di quadratura 2D,\n"
33        "- permutazioni dei nodi di bordo rispetto alle permutazione
34              dei vertici della faccia, \n"
35        "- diametro di ciascuna faccia, \n"
36        "- valori del flusso nei nodi di quadratura sugli elementi,\n"
37        "- valori della componente normale del flusso nei nodi
38              di quadratura sul bordo degli elementi,\n"
39        "- valori della divergenza del flusso nei nodi di quadratura
40              sugli elementi,\n"
41        "- valori del potenziale nei nodi di quadratura sugli elementi,\n"
42        "- valori del potenziale nei nodi di quadratura sul bordo
43              degli elementi,\n"
44        "- valori della variabile ibrida nei nodi di quadratura sui
45              bordi degli elementi,\n"
46        "- punti di visualizzazione per il flusso,\n"
```

```
47          "- punti di visualizzazione per il potenziale,\n"
48          "- punti di visualizzazione per la variabile ibrida,\n"
49          "- indice di Paraview per la visualizzazione
50                  del flusso,\n"
51          "- indice di Paraview per la visualizzazione
52                  del potenziale,\n"
53          "- indice di Paraview per la visualizzazione
54                  della variabile ibrida,\n"
55          "- valore delle f.d.b. nei punti di visualizzazione
56                  per il flusso,\n"
57          "- valore delle f.d.b. nei punti di visualizzazione
58                  per il potenziale,\n"
59          "- valore delle f.d.b. nei punti di visualizzazione
60                  per la variabile ibrida."
61          "- valori del flusso ricostruito nei nodi di quadratura
62                  sugli elementi,\n"
63          "- valori della componente normale del flusso ricostruito
64                  nei nodi di quadratura sul bordo degli elementi,\n"
65          "- valori del potenziale ricostruito nei nodi di quadratura
66                  sugli elementi,\n"
67          "- valori del potenziale ricostruito nei nodi di quadratura
68                  sul bordo degli elementi,\n"
69          "- valori del gradiente del potenziale ricostruito nei nodi
70                  di quadratura sugli elementi, \n"
71          "- punti di visualizzazione per il flusso ricostruito,\n"
72          "- punti di visualizzazione per il potenziale ricostruito,\n"
73          "- indice di Paraview per la visualizzazione del flusso
74                  ricostruito,\n"
75          "- indice di Paraview per la visualizzazione del potenziale
76                  ricostruito,\n"
77          "- valore delle f.d.b. nei punti di visualizzazione per il flusso
78                  ricostruito,\n"
79          "- valore delle f.d.b. nei punti di visualizzazione per il potenziale
80                  ricostruito,\n"
81  )
82
83  {
84          using namespace ef_misti_ibridi;
85          using namespace spazio_approx;
86          using namespace integrazione;
87          using namespace griglia_calcolo;
88          using namespace visualizzazione;
89
90          octave_value_list retval;
91
```

```
92          struct input {
93                  Metodo nome_metodo;
94                  int k_fl;
95                  int k_pt;
96                  int k_ib;
97                  std::string file_griglia;
98                  int esattezza_integrazione_el;
99                  int esattezza_integrazione_bordo;
100                 int grado_plot_flusso;
101                 int grado_plot_potenziale;
102                 int grado_plot_ibrida;
103                 int grado_plot_flusso_ricostr;
104                 int grado_plot_potenziale_ricostr;
105         };
106
107         if (args.length() < 5 )
108         {
109                 std::cout << "Numero insufficiente di parametri.
110                         Il programma non puo' eseguire. " << std::endl;
111                 abort();
112         }
113
114         input dati_ingresso;
115
116         dati_ingresso.nome_metodo = Metodo(args(0).int_value());
117
118         dati_ingresso.k_fl = args(1).int_value();
119         dati_ingresso.k_pt = args(2).int_value();
120         dati_ingresso.k_ib = args(3).int_value();
121         dati_ingresso.file_griglia = args(4).string_value();
122
123         if (args.length() == 12)
124         {
125                 dati_ingresso.esattezza_integrazione_el =
126                         args(5).int_value();
127                 dati_ingresso.esattezza_integrazione_bordo =
128                         args(6).int_value();
129                 dati_ingresso.grado_plot_flusso =
130                         args(7).int_value();
131                 dati_ingresso.grado_plot_potenziale =
132                         args(8).int_value();
133                 dati_ingresso.grado_plot_ibrida =
134                         args(9).int_value();
135                 dati_ingresso.grado_plot_flusso_ricostr =
136                         args(10).int_value();
```

```
137             dati_ingresso.grado_plot_potenziale_ricostr =
138                     args(11).int_value();
139         }
140
141         else {
142
143             dati_ingresso.esattezza_integrazione_el =
144                     ESATTEZZA_INT_EL;
145             dati_ingresso.esattezza_integrazione_bordo =
146                     ESATTEZZA_INT_BORDO;
147             dati_ingresso.grado_plot_flusso = 1;
148             dati_ingresso.grado_plot_potenziale = 1;
149             dati_ingresso.grado_plot_ibrida = 1;
150             dati_ingresso.grado_plot_flusso_ricostr = 1;
151             dati_ingresso.grado_plot_potenziale_ricostr = 1;
152         }
153
154         boost::shared_ptr<Mesh> ptr_griglia
155             (new MeshGMSH(dati_ingresso.file_griglia.c_str()));
156
157         // boost::shared_ptr<SpazioFunzionaleFlusso> ptr_flusso
158         //       (new SpFunFlussoPk(dati_ingresso.k_fl));
159         boost::shared_ptr<SpazioFunzionaleFlusso> ptr_flusso
160             (new SpFunFlussoRTk(dati_ingresso.k_fl));
161         boost::shared_ptr<SpazioFunzionalePotenziale> ptr_potenziale
162             (new SpFunPotenzialePk(dati_ingresso.k_pt));
163         boost::shared_ptr<SpazioFunzionaleIbrida> ptr_ibrida
164             (new SpFunIbridaPk(dati_ingresso.k_ib));
165
166         MetodoNumerico Risolutore(dati_ingresso.nome_metodo,
167                     dati_ingresso.k_fl, dati_ingresso.k_pt,
168                     dati_ingresso.k_ib, ptr_flusso, ptr_potenziale,
169                     ptr_ibrida, ptr_griglia,
170                     dati_ingresso.esattezza_integrazione_el,
171                     dati_ingresso.esattezza_integrazione_bordo,
172                     dati_ingresso.grado_plot_flusso,
173                     dati_ingresso.grado_plot_potenziale,
174                     dati_ingresso.grado_plot_ibrida);
175
176
177         boost::shared_ptr<SpazioFunzionaleFlusso> ptr_flusso_ricostr
178             (new SpFunFlussoRTk(dati_ingresso.k_fl));
179         boost::shared_ptr<SpazioFunzionalePotenziale> ptr_potenziale_ricostr
180             (new SpFunPotenzialePk(dati_ingresso.k_pt + 1));
181
```

```
182         MetodoNumerico Risol_ricostruz(Ricostr_RT, dati_ingresso.k_fl,
183                         dati_ingresso.k_pt + 1, dati_ingresso.k_ib,
184                         ptr_flusso_ricostr,
185                         ptr_potenziale_ricostr, ptr_ibrida, ptr_griglia,
186                         dati_ingresso.esattezza_integrazione_el,
187                         dati_ingresso.esattezza_integrazione_bordo,
188                         dati_ingresso.grado_plot_flusso_ricostr,
189                         dati_ingresso.grado_plot_potenziale_ricostr,
190                         dati_ingresso.grado_plot_ibrida);
191
192
193         // Quantità che servono alla risoluzione del problema
194
195         retval(0) = octave_value
196                 (Risolutore.get_cont_connettivita());
197         retval(1) = octave_value
198                 (Risolutore.get_cont_conn_permut());
199         retval(2) = octave_value
200                 (Risolutore.get_cont_det3D());
201         retval(3) = octave_value
202                 (Risolutore.get_cont_det2D());
203         retval(4) = octave_value
204                 (Risolutore.get_cont_mappe3D());
205         retval(5) = octave_value
206                 (Risolutore.get_cont_bc_facce());
207         retval(6) = octave_value
208                 (Risolutore.get_cont_permut_facce());
209         retval(7) = octave_value
210                 (Risolutore.get_nodi_pesi_quad_el());
211         retval(8) = octave_value
212                 (Risolutore.get_nodi_pesi_quad_bordo());
213         retval(9) = octave_value
214                 (Risolutore.get_permut_nodi_bordo());
215         retval(10) = octave_value
216                 (Risolutore.get_diam_facce());
217         retval(11) = octave_value
218                 (Risolutore.get_cont_flusso_elemento());
219         retval(12) = octave_value
220                 (Risolutore.get_cont_flusso_bordo_normale());
221         retval(13) = octave_value
222                 (Risolutore.get_cont_flusso_divergenza());
223         retval(14) = octave_value
224                 (Risolutore.get_cont_potenziale_elemento());
225         retval(15) = octave_value
226                 (Risolutore.get_cont_potenziale_bordo());
```

```
227         retval(16) = octave_value
228                 (Risolutore.get_cont_ibrida());
229         retval(17) = octave_value
230                 (Risolutore.get_punti_plot_flusso());
231         retval(18) = octave_value
232                 (Risolutore.get_punti_plot_potenziale());
233         retval(19) = octave_value
234                 (Risolutore.get_punti_plot_ibrida());
235         retval(20) = octave_value
236                 (Risolutore.get_ind_Paraview_flusso());
237         retval(21) = octave_value
238                 (Risolutore.get_ind_Paraview_potenziale());
239         retval(22) = octave_value
240                 (Risolutore.get_ind_Paraview_ibrida());
241         retval(23) = octave_value
242                 (Risolutore.get_valore_fdb_punti_plot_flusso());
243         retval(24) = octave_value
244                 (Risolutore.get_valore_fdb_punti_plot_potenziale());
245         retval(25) = octave_value
246                 (Risolutore.get_valore_fdb_punti_plot_ibrida());
247
248         // Quantità che servono alla ricostruzione delle incognite_star
249
250         retval(26) = octave_value
251                 (Risol_ricostruz.get_cont_flusso_elemento());
252         retval(27) = octave_value
253                 (Risol_ricostruz.get_cont_flusso_bordo_normale());
254         retval(28) = octave_value
255                 (Risol_ricostruz.get_cont_potenziale_elemento());
256         retval(29) = octave_value
257                 (Risol_ricostruz.get_cont_potenziale_grad_el());
258         retval(30) = octave_value
259                 (Risol_ricostruz.get_cont_potenziale_bordo());
260         retval(31) = octave_value
261                 (Risol_ricostruz.get_punti_plot_flusso());
262         retval(32) = octave_value
263                 (Risol_ricostruz.get_punti_plot_potenziale());
264         retval(33) = octave_value
265                 (Risol_ricostruz.get_ind_Paraview_flusso());
266         retval(34) = octave_value
267                 (Risol_ricostruz.get_ind_Paraview_potenziale());
268         retval(35) = octave_value
269                 (Risol_ricostruz.get_valore_fdb_punti_plot_flusso());
270         retval(36) = octave_value
271                 (Risol_ricostruz.get_valore_fdb_punti_plot_potenziale());
```

```
272
273         return retval;
274 }
```

Il cuore della funzione è costituito dalle istanze di tipo MetodoNumerico, qui chiamate Risolutore e Risol_ricostruz , che costruiscono, direttamente o indirettamente, tutti gli oggetti necessari al calcolo (il primo per il metodo di calcolo vero e proprio, il secondo per le procedure di ricostruzione delle incognite interne).
La scelta effettiva dei tipi degli spazi funzionali è effettuata definendo i tre puntatori di tipo boost :: shared_ptr <SpazioFunzionaleFlusso>, boost :: shared_ptr <SpazioFunzionalePotenziale> e boost :: shared_ptr <SpazioFunzionaleIbrida>, che rientrano poi nei parametri in ingresso del costruttore di MetodoNumerico. A questo punto, la funzione è richiamabile dal *prompt* di *Octave* con i parametri in ingresso scelti dall'utente.

Capitolo 4

Design del codice *Octave*

La struttura del codice *Octave* che verrà esposta nel seguito metterà in luce in modo particolare la logica di risoluzione richiesta dai metodi misti ibridi.

Dal punto di vista modellistico, il problema che si vuole andare a risolvere è un'equazione ellittica del second'ordine, contenente i termini di diffusione, trasporto, reazione e un termine forzante definito sull'intero volume.
Le equazioni del modello sono le seguenti:

$$\begin{aligned} \nabla \cdot (-\boldsymbol{a}\,\nabla u + \boldsymbol{v}\,u) + d\,u &= f \qquad \text{in } \Omega & \text{(4.1a)} \\ u &= g_D \qquad \text{su } \Gamma_D & \text{(4.1b)} \\ -(-\boldsymbol{a}\,\nabla u + \boldsymbol{v}\,u)\cdot \boldsymbol{n} + \gamma\,u &= h_R \qquad \text{su } \Gamma_R & \text{(4.1c)} \end{aligned}$$

dove l'argomento della divergenza è il *flusso totale*, da approssimarsi con l'incognita vettoriale $\boldsymbol{q}$.
La (4.1c) rappresenta la *legge di Newton per il trasferimento di calore*, con $\gamma \geq 0$ pari al coefficiente di scambio termico e

$$\boldsymbol{q}\cdot\boldsymbol{n} = \gamma\,(u - \frac{h_R}{\gamma})$$

dove $\frac{h_R}{\gamma}$ rappresenta la temperatura dell'ambiente circostante.
La nomenclatura di questa sezione è leggermente diversa rispetto a quella introdotta in 2.3.1; l'obiettivo è quello di essere fedeli con simboli e variabili utilizzati all'interno del codice *Octave*.
Scrivendo prima il sistema in forma mista e procedendo poi con le usuali integrazioni per parti, si perviene alla seguente formulazione:

$$
\begin{aligned}
(\boldsymbol{\alpha}\,\boldsymbol{q}_h, \boldsymbol{v}_h)_{\Omega_h} - (u_h, \nabla\cdot\boldsymbol{v}_h + \boldsymbol{\beta}\cdot\boldsymbol{v}_h)_{\Omega_h} + \langle\lambda_h, \boldsymbol{q}_h\cdot\boldsymbol{n}\rangle_{\partial K\setminus\Gamma_D} &= -\langle \mathrm{P}_\partial\, g_D, \boldsymbol{q}_h\cdot\boldsymbol{n}\rangle_{\Gamma_D} \\
(v_h, \nabla\cdot\boldsymbol{q}_h)_{\Omega_h} - \langle v_h, \tau\,\lambda_h\rangle_{\partial\Omega_h\setminus\Gamma_D} + \langle v_h, \tau\,u_h\rangle_{\partial\Omega_h} + (d\,u_h, v_h)_{\Omega_h} &= (f, v_h)_{\Omega_h} + \langle \tau\,\mathrm{P}_\partial\, g_D, v_h\rangle_{\Gamma_D} \\
\langle\mu_h, \boldsymbol{q}_h\cdot\boldsymbol{n}\rangle_{\partial\Omega_h\setminus\Gamma_D} - \langle\mu_h, \tau\,(\lambda_h - u_h)\rangle_{\partial\Omega_h\setminus\Gamma_D} - \langle\mu_h, \gamma\,\lambda_h\rangle_{\Gamma_R} &= -\langle\mu_h, \mathrm{P}_\partial\, h_R\rangle_{\Gamma_R}
\end{aligned}
$$

dove $\boldsymbol{q}_h, u_h, \lambda_h$ sono le tre incognite, $\boldsymbol{v}_h, v_h, \mu_h$ le rispettive funzioni test, mentre $\boldsymbol{\alpha} := \boldsymbol{a}^{-1}$, $\boldsymbol{\beta} := \boldsymbol{\alpha}\,\boldsymbol{v}$ e τ è la stabilizzazione locale, dipendente dal metodo numerico selezionato per la risoluzione.

Nel seguito verranno presentate le principali componenti del codice *Octave*.

4.1 Calcolo delle matrici locali

Il *file* da considerare è `calcola_matrici_locali.m`; nell'ordine vengono calcolate le matrici seguenti:

$$
\begin{aligned}
A_{ij}^K &= \frac{1}{|B|}\int_{\widehat{K}} \widehat{\boldsymbol{\omega}}_i^T\, B^T\,\boldsymbol{\alpha}\,B\,\widehat{\boldsymbol{\omega}}_j \\
B_{ij}^K &= -\int_{\widehat{K}} \widehat{\phi}_j\,(\widehat{\nabla}\cdot\widehat{\boldsymbol{\omega}}_i + \widehat{\boldsymbol{\omega}}_i^T\, B^T\boldsymbol{\beta}) \\
C_{ij}^K &= \int_{\widehat{K}} \widehat{\eta}_j\,\widehat{\boldsymbol{\omega}}_i\cdot\widehat{\boldsymbol{n}}_{\partial\widehat{K}} \\
D_{ij}^K &= \int_{\widehat{K}} \widehat{\phi}_i\,\widehat{\nabla}\cdot\widehat{\boldsymbol{\omega}}_j \\
E_{ij}^K &= |B|\int_{\widehat{K}} d\,\widehat{\phi}_j\,\widehat{\phi}_i + \sum_{\widehat{f}\,\in\,\partial\widehat{K}} \tau_f\,|f|\int_{\widehat{f}} \widehat{\phi}_j\,\widehat{\phi}_i \\
F_{ij}^K &= -\sum_{\widehat{f}\,\in\,\partial\widehat{K}} \tau_f\,|f|\int_{\widehat{f}} \widehat{\eta}_j\,\widehat{\phi}_i \\
G_{ij}^K &= C_{ji}^K \\
H_{ij}^K &= -F_{ji}^K \\
I_{ij}^K &= -\sum_{\widehat{f}\,\in\,\partial\widehat{K}} \tau_f\,|f|\int_{\widehat{f}} \widehat{\eta}_j\,\widehat{\eta}_i - \sum_{\widehat{f}\,\in\,(\partial\widehat{K}\,\cap\,\Gamma_R)} |f|\int_{\widehat{f}} \gamma\,\widehat{\eta}_j\,\widehat{\eta}_i \\
b_i^K &= \mathbf{0} \\
f_i^K &= |B|\int_{\widehat{K}} f\,\widehat{\phi}_i \\
h_i^{\partial K} &= -\sum_{\widehat{f}\,\in\,(\partial\widehat{K}\,\cap\,\Gamma_R)} |f|\int_{\widehat{f}} \widehat{\mathrm{P}_\partial\, h_R}\,\widehat{\eta}_i
\end{aligned}
$$

dove B è la mappa di trasformazione di ciascun tetraedro, $|B|$ il valore assoluto del suo determinante, $|f|$ il rapporto di dilatazione superficiale per ciascuna faccia f rispetto al

triangolo 2D di riferimento e $(\{\boldsymbol{\omega}\}, \{\phi\}, \{\eta\})$ sono le funzioni di base generanti gli spazi a cui appartengono le tre incognite del problema.

E' opportuno fare alcune considerazioni riguardo alla matrici precedentemente introdotte:

- l'assemblaggio assume che il bordo dell'elemento sia caratterizzato da **sole condizioni naturali** (con $\gamma = 0$ per ottenere specificatamente Neumann al posto del più generale Robin), in quanto l'imposizione delle condizioni essenziali di Dirichlet verrà operata a valle dell'assemblaggio della matrice globale per le sole incognite ibride e immediatamente prima della risoluzione del sistema lineare ad essa associato.
  Questa scelta è dettata da maggiore praticità: così facendo, infatti, tutti i cicli *for* necessari all'interno non devono distinguere se una faccia richieda o meno le condizioni essenziali, rendendo così **le dimensioni locali del problema rispetto alla variabile ibrida identiche per tutti i tetraedri**. Ciò è molto utile, in quanto i gradi di libertà assoluti saranno ordinati a blocchi regolari, elemento per elemento, e all'interno di ogni blocco, faccia per faccia. L'imposizione delle condizioni di Dirichlet, come preannunciato, verrà effettuata successivamente, selezionando opportunamente le sole facce coinvolte e i gradi di libertà ibridi ad esse associati.

- l'assemblaggio di tutte le matrici in cui interviene un prodotto fra le funzioni di basi per l'incognita ibrida (definite sul dominio 2D di riferimento) e le restrizioni su ciascuna faccia di bordo delle funzioni di base per il flusso e il potenziale (definite invece sul dominio 3D di riferimento) va trattato con delicatezza. La parte C++ del codice di calcolo restituisce infatti le funzioni di base per λ_h calcolate nei nodi di quadratura dell'elemento di riferimento 2D, e **per ciascuna faccia in numerazione assoluta** tali valori coincidono – nodo per nodo – con quelli delle funzioni di base "fisiche" definite su ciascuna faccia assoluta, avente una ben precisa e ordinata terna di identificativi per i vertici.
  Ciascuna faccia interna viene condivisa da due tetraedri adiacenti, che la considerano una faccia di bordo avente una precisa numerazione locale (da 1 a 4: per la scelta, si vada l'appendice A.2.1), e un preciso ordinamento dei vertici. Per garantire che il prodotto nodo per nodo fra le funzioni di base sia corretto, nel codice è stato operato un **riordinamento degli indici dei nodi di bordo per la faccia *locale* del tetraedro, su cui sono calcolate le restrizioni delle funzioni di base per flusso e potenziale**.
  Ecco un estratto del codice: fissata la faccia locale `f` (che sarà un intero compreso fra 1 e 4), si estrae da un opportuno contenitore la faccia in numerazione assoluta ad essa associata e si va a stabilire qual è la permutazione della terna dei vertici che permette di passare da quella assoluta a quella locale:

```
f_assoluta = cont_connettivita(f);
```

```
idx_perm = trova_perm (cont_permut_facce(f_assoluta, :),
cont_conn_permut(f, :));
```

Fatto ciò, per ciascun nodo `t` è possibile estrarre l'indice del nodo corrispondente, permutato secondo la permutazione individuata:

```
idx_nodo_permutato = permut_nodi_bordo(idx_perm, t);
```

In questo modo si garantisce che il prodotto, nodo per nodo, fra l'incognita ibrida e quelle di volume ristrette al bordo sia eseguito in maniera corretta.

- il codice di calcolo manipola in maniera leggermente diversa le integrazioni in cui compaiono termini del tipo $B\,\widehat{\boldsymbol{\omega}}$. Infatti, la parte C++ del codice restituisce il valore delle funzioni di base per l'incognita vettoriale già moltiplicate per la mappa di Piola (si veda l'appendice A.3 per i dettagli analitici), vale a dire le quantità $\frac{1}{|B|}B\,\widehat{\boldsymbol{\omega}}$; nel codice comparirà, ove necessario, un'oppurtuna correzione per bilanciare numeratori e denominatori in modo coerente.

Una volta assemblate tutte le matrici locali, si procede su ogni elemento con la condensazione statica, in modo da poter trovare $\boldsymbol{q}_h$ e u_h espresse in funzioni della sola incognita λ_h. Precisamente si ottiene:

$$\begin{aligned}
V^K &= (E^K - D^K\,(A^K)^{-1}\,B^K)^{-1} \\
U^K &= V^K\,(D^K\,(A^K)^{-1}\,C^K - F^K) \\
Q^K &= -\,(A^K)^{-1}(B^K\,U^K + C^K) \\
W^K &= -\,(A^K)^{-1}B^K\,V^K \\
u_h^K &= U^K\,\lambda_h^{\partial K} + V^K\,(f^K - D^K\,(A^K)^{-1}\,b^K) \\
\boldsymbol{q}_h^K &= Q^K\,\lambda_h^{\partial K} + W^K\,(f^K - D^K\,(A^K)^{-1}\,b^K) + (A^K)^{-1}\,b^K
\end{aligned}$$

Sfruttando ora la terza equazione del modello, si perviene ad un sistema locale in K per la sola incognita ibrida λ_h, esprimibile nella forma $\mathbb{E}^K\,\lambda_h^{\partial K} = \mathbb{H}^K$, dove

$$\begin{aligned}
\mathbb{E}^K &= G^K\,Q^K + H^K\,U^K + I^K \\
\mathbb{H}^K &= h^{\partial K} - G^K\,W^K\,(f^K - D^K\,(A^K)^{-1}\,b^K) - H^K\,V^K\,(f^K - D^K\,(A^K)^{-1}\,b^K) \\
&\quad - G^K\,(A^K)^{-1}\,b^K
\end{aligned}$$

Iterando su tutti i tetraedri, si otterrà il sistema lineare globale $\mathbb{E}\,\lambda_h = \mathbb{H}$ che, invertito numericamente da *Octave*, consentirà di determinare i gradi di libertà globalmente accoppiati, quelli cioè corrispondenti alla sole variabile λ_h.

Il flusso logico fin qui descritto testimonia ancora una volta che **i metodi misti ibridi in esame sono a pieno titolo metodi di Galerkin**, basati su una formulazione agli spostamenti generalizzata per l'incognita di bordo λ_h; il processo di assemblaggio di $\mathbb{E}$ a partire dai contributi su ogni tetraedro K ricalca con perfetta analogia quanto viene operato nel caso dei più tradizionali e diffusi metodi *Continuous Galerkin*.

4.2 Risoluzione del sistema per l'incognita λ_h

Una volta assemblata la matrice globale $\mathbb{E}$, prima di passare alla risoluzione numerica del sistema lineare è necessario imporre le condizioni di Dirichlet, perchè $\mathbb{E}$ è singolare.
Precisamente, il codice va a selezionare quali sono le facce di bordo (in numerazione assoluta) alle quali è associata la condizione essenziale, e impone tale condizione attraverso la proiezione (*file* `proiez_L2_bordo.m`) già descritta nella sezione 2.4.
Così facendo, una parte del vettore di coefficienti associato all'incognita λ_h è stato determinato; i rimanenti coefficienti incogniti possono essere facilmente ottenuti risolvendo un sistema quadrato di dimensione minore, avente come matrice la sottomatrice estratta da $\mathbb{E}$ rispetto agli indici (in numerazione assoluta) dei gradi di libertà diversi da quelli associati a condizioni di Dirichlet.
In formule, a meno di un riordinamento di indici, si ha:

$$\begin{pmatrix} \mathbb{E}_{11} & \mathbb{E}_{12} \\ \mathbb{E}_{21} & \mathbb{E}_{22} \end{pmatrix} \begin{pmatrix} L_1 \\ L_2 \end{pmatrix} = \begin{pmatrix} \mathbb{H}_1 \\ \mathbb{H}_2 \end{pmatrix}$$

che corrisponde al seguente estratto di codice:

```
L1 = EE11 \ (HH1 - EE12 * L2);
HH2 = EE21 * L1 + EE22 * L2;
L = zeros ( length (gdl_ibridi_totali) , 1);
L ( gdl_lambda_h ) = L1;
L ( gdl_dirichlet ) = L2;
```

dove `L` memorizza l'intero vettore di coefficienti per l'incognita ibrida.

4.3 Calcolo di $\boldsymbol{q}_h$, $\boldsymbol{q}_h^*$, u_h e u_h^*

E' ora possibile calcolare, tetraedro per tetraedro, il valore delle incognite $\boldsymbol{q}_h$ e u_h, grazie alle relazioni matriciali descritte nella sezione 4.1. Inoltre, note $\boldsymbol{q}_h$ e u_h, è possibile eseguire anche un appropriato *post-processing* (descritto dettagliatamente nell'appendice A.4) allo scopo di calcolare $\boldsymbol{q}_h^*$ e u_h^*.
Tale ricostruzione locale richiede la conoscenza di $\hat{\boldsymbol{q}}_h \cdot \boldsymbol{n}$, che è ricavabile in quanto $\boldsymbol{q}_h$ e u_h sono state opportunamente determinate; il codice di calcolo provvede a suddividere i

diversi contributi $\hat{\boldsymbol{q}}_h \cdot \boldsymbol{n}$, operando una permutazione nei nodi di quadratura identica a quella già descritta nella fase di assemblaggio delle matrici locali.

```
f_assoluta = cont_connettivita(f);
idx_perm = trova_perm (cont_permut_facce(f_assoluta, :),
cont_conn_permut(f, :));
vec_perm = permut_nodi_bordo(idx_perm, :);
q_h_n_flusso (f, :)  = transpose(q_h) *
squeeze(cont_flusso_bordo_normale (:, vec_perm, f));
q_h_n_pot (f, :)  = transpose(u_h) *
squeeze(cont_potenziale_bordo (:, vec_perm, f));
q_h_n_ib (f, :)  =
transpose(lambda_h((f - 1) * gdl_ibrida_per_faccia + 1 :  f * gdl_ibrida_per_faccia))
* cont_ibrida;
```

Il vettore `vec_perm` ha lunghezza pari al numero di nodi di quadratura di bordo e archivia gli indici di tali nodi, permutati secondo la permutazione della terna di vertici che consente di passare dalla faccia di riferimento alla faccia locale.

4.4 Calcolo dell'errore

Nel caso in cui la soluzione sia nota, è possibile calcolare l'errore rispetto alla soluzione esatta (da impostare correttamente nel *file* `set_soluz_esatta.m`). Gli errori calcolabili con il *file* `calc_errori.m` sono:

- per la variabile ibrida, la norma L^2 della differenza fra la soluzione esatta e la soluzione numerica calcolata. Matematicamente:

$$||u - \lambda_h||_{L^2(\mathscr{E}_h)} = \sqrt{\sum_{f \in \mathscr{E}_h} h_f \int (u - \lambda_h)^2}$$

  Si noti la presenza di h_f (nel codice posto pari al diametro della faccia[1]), necessario a rendere commensurabile l'errore commesso sulle facce con quello commesso sull'intero volume.

- per la variabile scalare potenziale (sia esso proprio del metodo o derivante dal *post-processing*), la norma L^2 della differenza fra la soluzione esatta e la soluzione numerica calcolata. Matematicamente:

[1] Andrebbe bene anche una grandezza variabile linearmente al raffinarsi della griglia: ad esempio, la radice quadrata dell'area della faccia.

$$||u - u_h||_{L^2(\Omega_h)} = \sqrt{\sum_{K \in \Omega_h} \int (u - u_h)^2}$$

- per la variabile scalare flusso (sia esso proprio del metodo o derivante dal *post-processing*), la norma $\boldsymbol{L}^2$ della differenza fra la soluzione esatta e la soluzione numerica calcolata. Matematicamente:

$$||\boldsymbol{q} - \boldsymbol{q}_h||_{\boldsymbol{L}^2(\Omega_h)} = \sqrt{\sum_{K \in \Omega_h} \int |\boldsymbol{q} - \boldsymbol{q}_h|^2}$$

4.5 Visualizzazione della soluzione

I due *files* `visualizzazione_ibrida.m` e `visualizzazione_flusso_potenziale.m` provvedono a salvare in formato XML di VTK tutte le informazioni necessarie alla visualizzazione della soluzione numerica, grazie alla conoscenza di λ_h, $\boldsymbol{q}_h$, $\boldsymbol{q}_h^*$, u_h e u_h^*.

Capitolo 5

Simulazioni numeriche

5.1 Caso test 1: soluzione analitica e calcolo dell'errore

Per studiare gli ordini di convergenza sperimentale dei metodi descritti nella sezione 2.4, si presenta di seguito un interessante caso test, dotato di soluzione analitica nota.

Sia assegnata come geometria di calcolo il dominio unitario $\Omega = [0,1]^d$; consideriamo il problema di diffusione-trasporto-reazione scritto in forma conservativa:

$$\begin{aligned} \nabla \cdot (-\epsilon \nabla u + \boldsymbol{a}\, u) + \sigma\, u &= f \qquad \text{in } \Omega \\ u &= 0 \qquad \text{su } \partial\Omega \end{aligned}$$

dove i coefficienti del problema sono indicati come segue: assegnato lo scalare $\epsilon \in \mathbb{R}^+$, il vettore di interi $\boldsymbol{m} \in (\mathbb{N}^+)^d$ e la funzione univariata

$$\eta_s(t) = 1 - e^{-\frac{1-t^s}{s\epsilon}} \qquad s = 1, 2, \ldots$$

si definiscono in questo modo le componenti del campo avvettivo $\boldsymbol{a}$:

$$a_i = x_i^{m_i - 1}$$

il contributo di reazione è $\sigma = \sum_{i=1}^{d} \sigma_i(\boldsymbol{x})$, con

$$\sigma_i(\boldsymbol{x}) = \begin{cases} 0 & \text{se } m_i = 1 \\ x_i^{m_i - 2} & \text{se } m_i \geq 2 \end{cases}$$

mentre il termine forzante $f = \sum_{i=1}^{d} f_i(\boldsymbol{x})$ con

$$f_i(\boldsymbol{x}) = \begin{cases} (1+m_i-\eta_{m_i}(x_i))x_i^{m_i-1}\prod_{j\neq i} x_j\eta_{m_j}(x_j) & \text{se } m_i = 1 \\ (1+m_i)x_i^{m_i-1}\prod_{j\neq i} x_j\eta_{m_j}(x_j) & \text{se } m_i \geq 2 \end{cases}$$

La soluzione esatta del problema differenziale è definita nel modo seguente:

$$u(\boldsymbol{x}) = \prod_{i=1}^{d} x_i\eta_{m_i}(x_i)$$

con gradiente

$$\partial_{x_i} u(\boldsymbol{x}) = (\eta_{m_i}(x_i) - \frac{x_i^{m_i}}{\epsilon}(1-\eta_{m_i}(x_i)))\prod_{j\neq i} x_j\eta_{m_j}(x_j)$$

flusso totale

$$q_i = (-\epsilon\,\eta_{m_i}(x_i) + x_i^{m_i})\prod_{j\neq i} x_j\eta_{m_j}(x_j)$$

e divergenza del flusso totale

$$\nabla\cdot\boldsymbol{q} = \sum_{i=1}^{d} x_i^{m_i-1}(1+m_i-\eta_{m_i}(x_i))\prod_{j\neq i} x_j\eta_{m_j}(x_j)$$

Grazie alla conoscenza della soluzione esatta, sarà possibile analizzare il profilo asintotico di abbattimento dell'errore, al raffinarsi della griglia di calcolo e all'aumentare del grado polinomiale scelto per l'approssimazione con elementi finiti.

5.1.1 Risultati di convergenza

Presentiamo ora i risultati di convergenza rispetto al caso test considerato.
In particolare, sono stati scelti i seguenti parametri: $\epsilon = 5$ e $\boldsymbol{a} = [1, 2, 3]^T$, da cui si ottengono immediatamente i contributi di trasporto, reazione e la forzante. In virtù di questa scelta, il problema è a **diffusione dominante**.

Tra i metodi del codice di calcolo disponibili all'uso, viene qui dapprima presentato il profilo di decadimento dell'errore per il metodo LDG-H, con scelta del coefficiente di stabilizzazione $\tau = 1$ su tutte la quattro facce di frontiera di ciascun tetraedro e grado polinomiale $k = 0, 1, 2$. La scelta di τ, semplice nell'implementazione in quanto non richiede alcun ulteriore calcolo rispetto ai parametri che definiscono ciascuna faccia (il diametro, ad esempio), consente di ottenere il massimo ordine di convergenza, a parità di geometria computazionale e di grado polinomiale (si veda, per un'adeguata trattazione nel caso bidimensionale con trasporto e reazione non identicamente nulli, [CDGRS2009]. Ivi vengono discusse anche possibili scelte di τ che degradano l'ordine di convergenza).

Le variabili che sono state considerate nello studio di convergenza sono cinque: oltre alle usuali incognite $\boldsymbol{q}_h, u_h, \lambda_h$ rappresentanti il flusso, il potenziale e l'incognita ibrida sul bordo, è stato studiato il comportamento di nuove variabili calcolate *ad hoc*, con un opportuno *post-processing*: il potenziale ricostruito u_h^* e il flusso ricostruito $\boldsymbol{q}_h^*$. Per ulteriori dettagli, relativi all'implementazione di tali ricostruzioni e alle proprietà delle nuove grandezze così costruite, si rimanda alle appendici A.4.1 (per il flusso) e A.4.2 (per il potenziale).

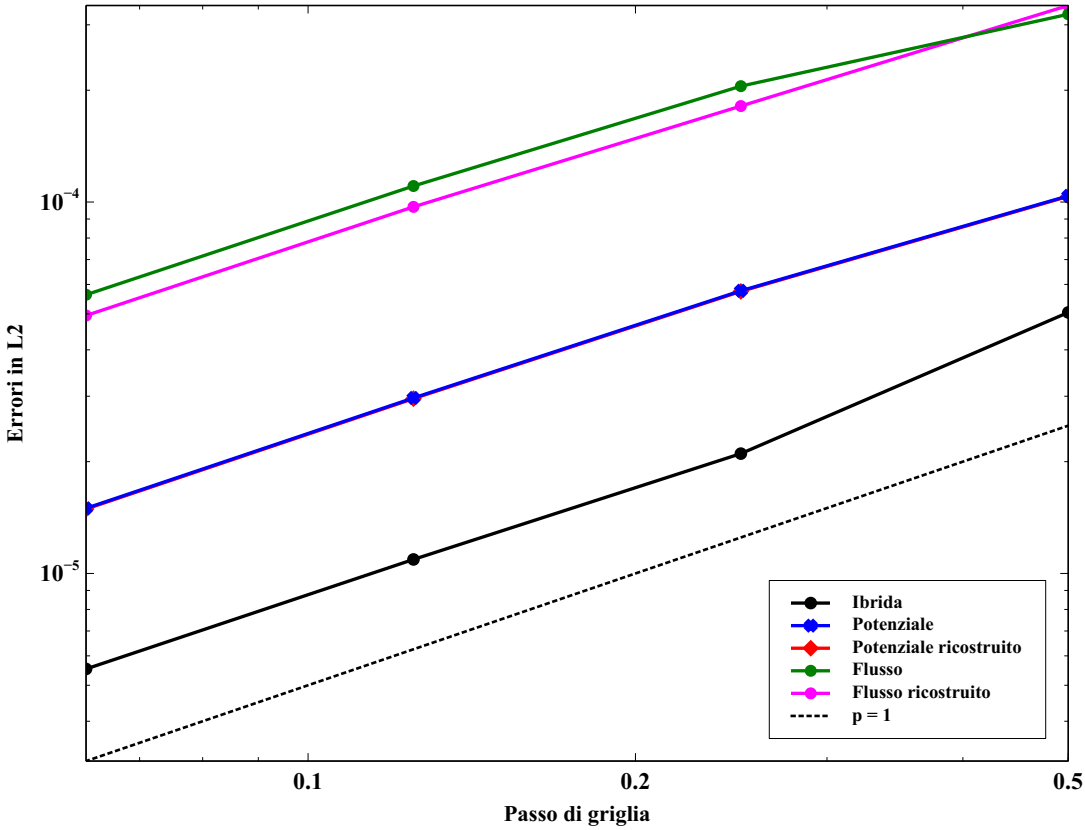

Figura 5.1: Profilo di decadimento dell'errore per il metodo LDG-H, con stabilizzazione $\tau = 1$ su ciascuna faccia di bordo per ogni tetraedro della griglia di calcolo e grado polinomiale $k = 0$. Il dominio è stato ottenuto suddividendo gli spigoli del cubo unitario rispettivamente in 2, 4, 8 e 16 parti uguali, dando luogo ad una griglia strutturata composta da 48, 384, 3072 e 24576 tetraedri.

La figura 5.4 illustra invece i risultati di convergenza per il metodo RT-H con grado $k = 0$; in questo caso non è stato necessario operare alcuna ricostruzione sulla variabile vettoriale flusso, in quanto $\boldsymbol{q}_h$ è già, per costruzione, appartenente allo spazio $H(\mathrm{div}, \Omega_h)$.

Dalla figure 5.1, 5.2, 5.3 e 5.4, tutte tracciate in coordinate doppio-logaritmiche, possiamo trarre alcune conclusioni molto importanti:

- il metodo LDG-H esibisce, **rispetto al campo vettoriale flusso e al campo scalare potenziale in norma L^2, una convergenza ottimale pari a grado**

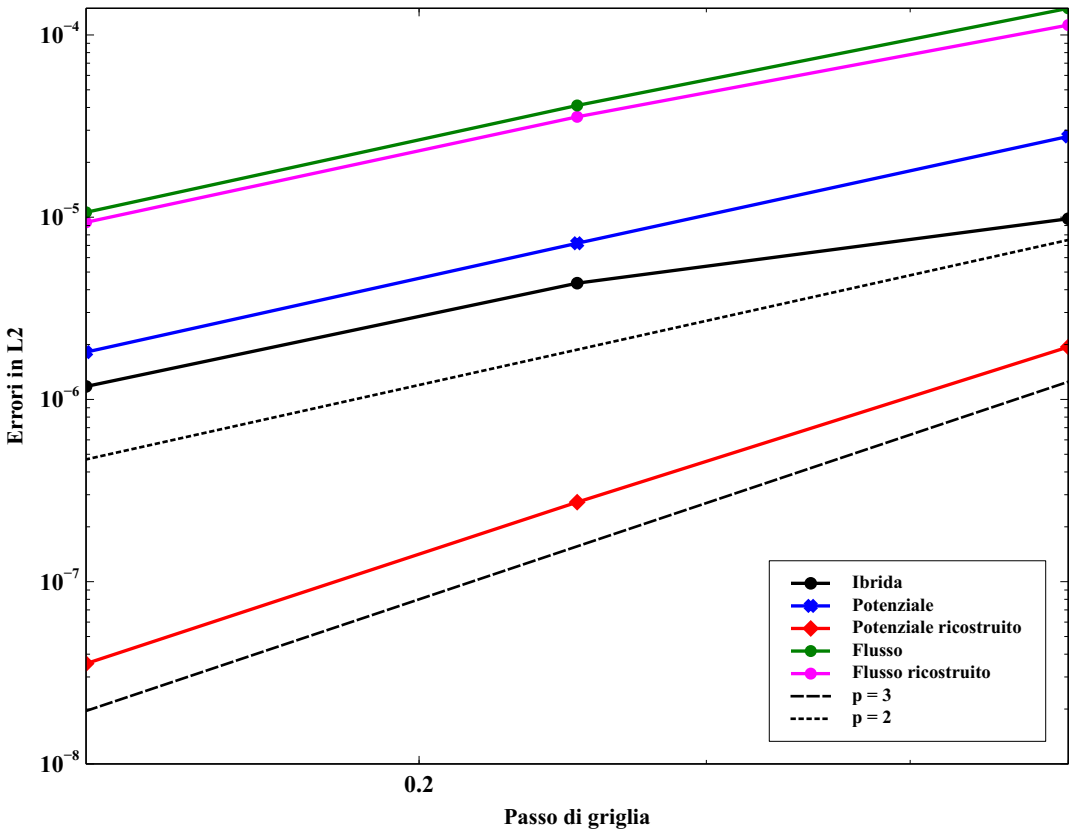

Figura 5.2: Profilo di decadimento dell'errore per il metodo LDG-H, con stabilizzazione $\tau = 1$ su ciascuna faccia di bordo per ogni tetraedro della griglia di calcolo e grado polinomiale $k = 1$. Il dominio è stato ottenuto suddividendo gli spigoli del cubo unitario rispettivamente in 2, 4 e 8 parti uguali, dando luogo ad una griglia strutturata composta da 48, 384 e 3072 tetraedri.

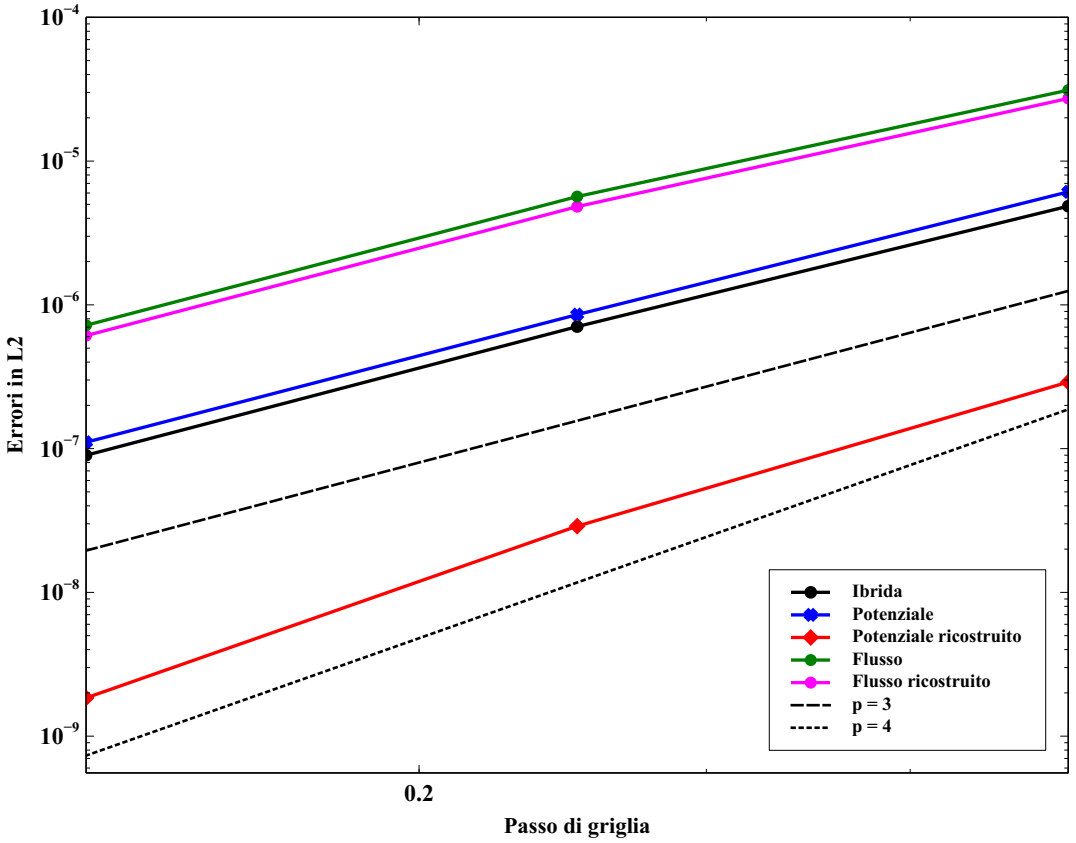

Figura 5.3: Profilo di decadimento dell'errore per il metodo LDG-H, con stabilizzazione $\tau = 1$ su ciascuna faccia di bordo per ogni tetraedro della griglia di calcolo e grado polinomiale $k = 2$. Il dominio è stato ottenuto suddividendo gli spigoli del cubo unitario rispettivamente in 2, 4 e 8 parti uguali, dando luogo ad una griglia strutturata composta da 48, 384 e 3072 tetraedri.

$k+1$. In particolar modo, riguardo al flusso, non si dimentichi che nel caso di elementi finiti *Continuous Galerkin* si può dimostrare una convergenza di ordine k per la norma H^1 dell'unica incognita presente u. Tale norma – almeno nel caso comune in cui una porzione non nulla del bordo del dominio sia di Dirichlet – può essere sostituita (grazie alla *diseguaglianza di Poincaré*) dalla norma equivalente del gradiente di u in $\boldsymbol{L}^2$.

- l'oculato ***post-processing*** **locale ha consentito di ottenere una superconvergenza** $k+2$ **sul potenziale *ricostruito***, a patto – naturalmente – di **scegliere uno spazio polinomiale più ricco, cioè comprendente polinomi di un grado superiore**. Non sarebbe lecito attendersi, infatti, *prima della ricostruzione*, un ordine migliore per il potenziale, in quanto $k+1$ è il *best approximation error* ottenibile attraverso $\mathscr{P}_k(K)$.
  L'unico caso in cui non si ottiene tale superconvergenza è il grado $k=0$.

- a parità di griglia di calcolo e di grado polinomiale k, il metodo RT-H permette di ottenere **errori assoluti più piccoli** rispetto al metodo LDG-H. Ciò è spiegabile grazie ad uno **spazio approssimante per il campo vettoriale** $\boldsymbol{q}$ **più ricco**; nondimeno, nel caso di più **basso ordine** $k=0$, il metodo Raviart-Thomas esibisce la **superconvergenza di ordine** $k+2$ **sul potenziale ricostruito**.

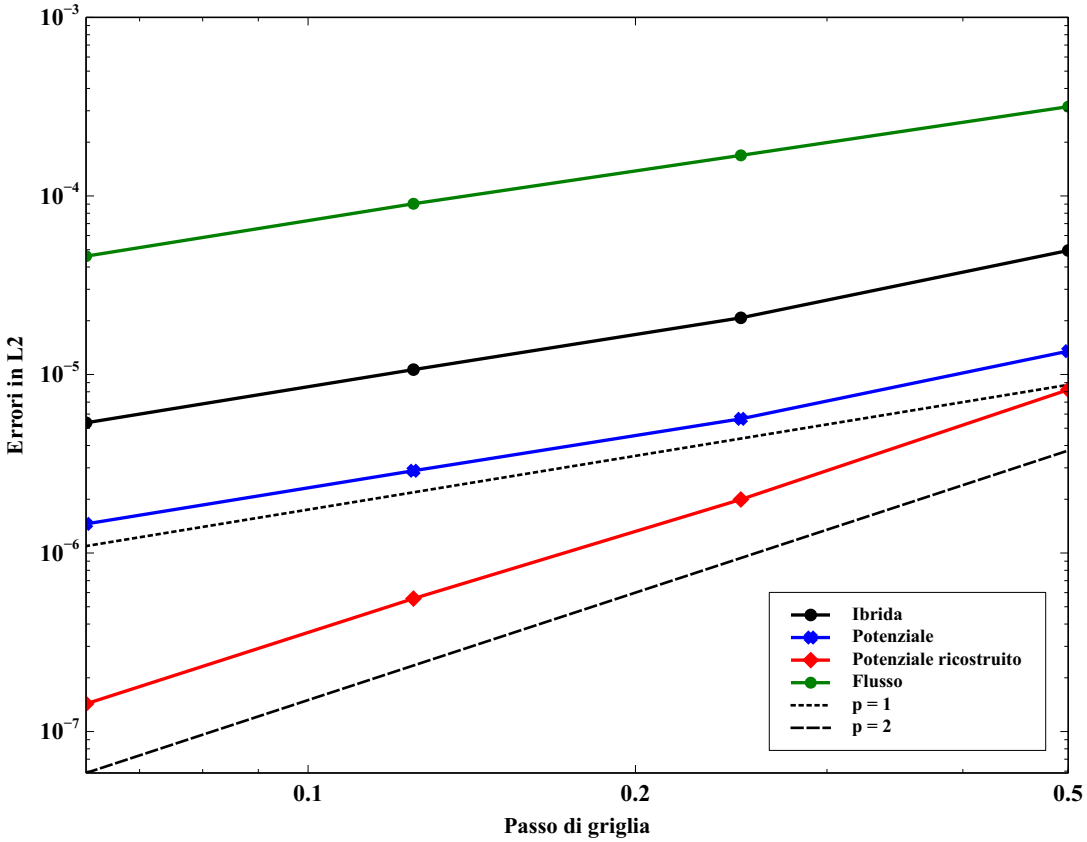

Figura 5.4: Profilo di decadimento dell'errore per il metodo RT-H con grado polinomiale $k = 0$. Il dominio è stato ottenuto suddividendo gli spigoli del cubo unitario rispettivamente in 2, 4, 8 e 16 parti uguali, dando luogo ad una griglia strutturata composta da 48, 384, 3072 e 24576 tetraedri.

5.2 Caso test 2: flusso di Darcy in un mezzo poroso

Come seconda applicazione pratica, si presenta nel seguito un interessante problema, già affrontato in [CDASC2010]. Si tratta di un problema di Darcy in un mezzo poroso, nel quale è presente una frattura: all'interno e all'esterno di tale frattura, i parametri di permeabilità sono diversi. La fisica sottostante il problema fa riferimento allo studio dei flussi di acque nel sottosuolo, oppure all'analisi dei bacini geologici per l'estrazione del petrolio. Rispetto a [CDASC2010], in cui viene proposto il metodo XFEM (si vedano a tal proposito le referenze citate dagli autori), la frattura non sarà modellata come una varietà bidimensionale inserita in un dominio tridimensionale, bensì come **una sottile porzione tridimensionale del dominio fisico**.
Desumendo dalla referenza tutti i parametri di interesse idraulico (in ultima analisi, la permeabilità trasversale e tangenziale nella frattura e il contributo forzante esteso a tutto il volume), verrà utilizzato il codice 3D per riprodurre, in primo luogo, la fenomenologia già riscontrata dagli autori nel caso 2D; in seguito, verrà simulata una filtrazione in cui il termine forzante dipende dalla terza dimensione spaziale.

5.2.1 Descrizione del problema

Il problema può essere matematicamente formulato nel modo seguente: dette $\boldsymbol{u}$ e p rispettivamente la velocità di filtrazione del fluido e la sua pressione in ogni punto dello spazio fisico, le equazioni di Darcy si esprimono nella forma:

$$\eta\,\boldsymbol{u} + \nabla p = 0 \qquad \text{in } \Omega \tag{5.1a}$$

$$\nabla \cdot \boldsymbol{u} = f_q \qquad \text{in } \Omega \tag{5.1b}$$

dove il termine forzante f_q, presente a secondo membro nella (5.1b), rappresenta un eventuale termine sorgente presente all'interno del dominio fisico.
Le equazioni sono già scritte in forma mista, in cui è naturale riconoscere come $\boldsymbol{u}$ l'incognita vettoriale e p l'incognita scalare sul volume.
La grandezza η rappresenta l'inverso del tensore di permeabilità proprio del materiale considerato: a seconda che si faccia riferimento all'interno oppure all'esterno della frattura, tale grandezza esibirà un valore differente.
Per quanto riguarda le condizioni al contorno, esse saranno:

$$p = p_0 \qquad \text{su } \Gamma_D$$

$$\boldsymbol{u} \cdot \boldsymbol{n} = u_0 \qquad \text{su } \Gamma_N$$

con frontiera di Dirichlet e di Neumann che verranno indicate in seguito[1].
Si può subito notare che nel caso in cui il **tensore di permeabilità corrisponda all'identità** (o comunque ad un suo multiplo), vale a dire in caso di isotropia, **la velocità di filtrazione è esattamente il gradiente della velocità, a meno del segno**. Questa prima osservazione si rivelerà utile nell'analisi dei risultati numerici che verranno presentati a breve.

5.2.2 Descrizione dei parametri fisici

La frattura all'interno del mezzo poroso è stata matematicamente simulata attraverso una sottile porzione del dominio di calcolo, assunto per semplicità pari al cubo unitario, come già in [CDASC2010].
Tale porzione è stata generata nel piano xy e poi estrusa verticalmente rispetto alla direzione z. Più precisamente, l'equazione delle rette delimitanti la frattura sono, rispettivamente, $y+2x = 1.4$ e $y+2x = 1.5$, per uno spessore di 0.05. La figura 5.5 è stata ottenuta con il generatore di griglia GMSH (per una trattazione specifica del programma e delle sue caratteristiche algoritmiche, si può fare diretto riferimento a [GR2009]).

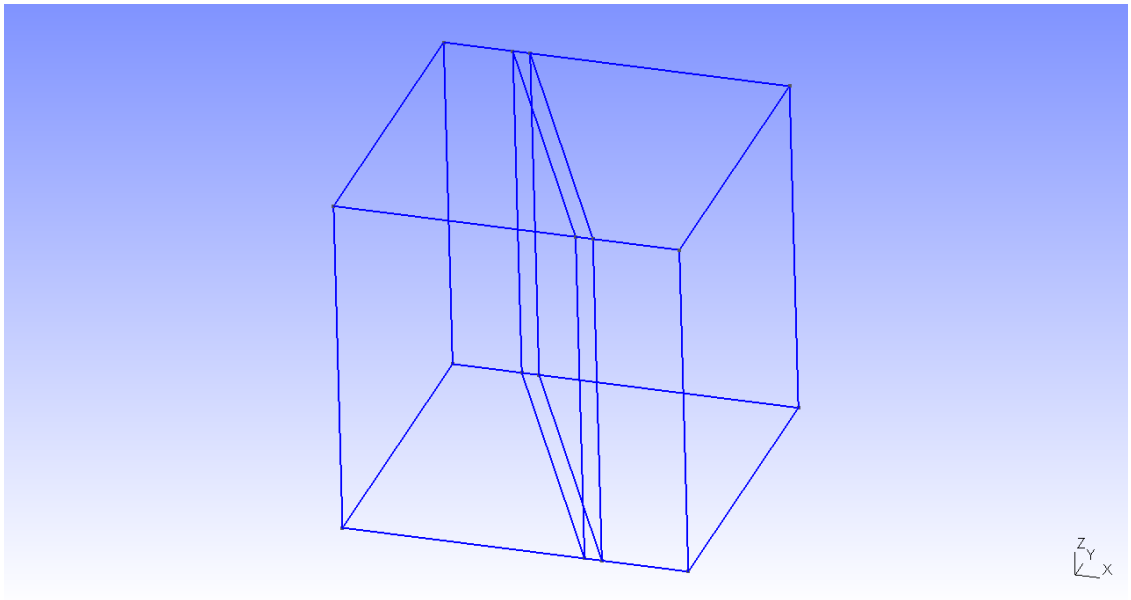

Figura 5.5: Geometria di calcolo per un flusso di Darcy in un mezzo poroso: la frattura è stata ottenuta da una sottile porzione bidimensionale di spessore 0.05, compresa fra due rette nel piano xy e poi estrusa rispetto alla terza coordinata z.

[1] Si presti attenzione alla nomenclatura utilizzata in questa sezione: rispetto a [CDASC2010], è stata mantenuta la medesima simbologia sulle incognite $\boldsymbol{u}$ e p, mentre i due tratti di frontiera sono stati scambiati: per mantenere uniformità con quanto scritto nel codice di calcolo, il bordo di Dirichlet corrisponde a quello in cui si vuole imporre il valore dell'incognita scalare, mentre il bordo di Neumann quello in cui si va a cercare il soddisfacimento della condizione sulla derivata normale del flusso totale, che in questo caso è puramente diffusivo, non essendoci contributi avvettivi.

Per descrivere compiutamente il mezzo poroso, bisogna indicare il suo tensore di permeabilità.
Gli autori in [CDASC2010] fanno riferimento a tre parametri idraulici molto importanti:

- η: coefficiente che rappresenta l'inverso della permeabilità nel mezzo al di fuori della frattura. Rispetto all'equazione (5.1a), in cui η veniva introdotto come inverso del generico tensore di permeabilità, d'ora in avanti η servirà a rappresentare compiutamente il comportamento locale *al di fuori della frattura*, sotto ipotesi di isotropia.

- η_Γ: coefficiente che rappresenta la resistenza alla filtrazione in direzione normale alla frattura. E' ottenuta dal rapporto $\frac{l_\Gamma}{K_{\Gamma,n}}$, dove il numeratore rappresenta lo spessore della frattura, mentre il denominatore la permeabilità in direzione normale alla frattura.

- $\hat{\eta}$: coefficiente che rappresenta la resistenza alla filtrazione in direzione tangenziale alla frattura. E' ottenuta dal rapporto $\frac{1}{l_\Gamma K_{\Gamma,\tau}}$, dove il secondo fattore del denominatore rappresenta la permeabilità in direzione tangenziale alla frattura.

Ora si hanno a disposizione tutte le informazioni per costruire il tensore di permeabilità:

- all'*esterno* della frattura: in tutte le simulazioni eseguite si è considerato $\eta = 1$, da cui il tensore identità;

- all'*interno* della frattura: scelti i valori η_Γ, $\hat{\eta}$ (identici a quelli in [CDASC2010] per poter confrontare i risultati numerici) e lo spessore l_Γ (fissato autonomamente all'atto della generazione della geometria di calcolo), invertendo facilmente le precedenti relazioni costitutive sarà immediato ottenere il tensore di permeabilità. Esso sarà rappresentato (in coordinate *locali*) dalla matrice diagonale seguente:

$$\tilde{K} = \begin{pmatrix} K_{\Gamma,\tau} & 0 & 0 \\ 0 & K_{\Gamma,\tau} & 0 \\ 0 & 0 & K_{\Gamma,n} \end{pmatrix}$$

  Resta ora da operare un opportuno cambio di coordinate per ottenere il tensore in coordinate fisiche, da inserire direttamente nel codice di calcolo. Ricordando le equazioni generanti la frattura, una base ortonormale di generatori del piano della frattura è data dai seguenti due vettori: $\frac{1}{\sqrt{5}}[-1,\, 2,\, 0]^T$ e $[0,\, 0,\, 1]^T$. La direzione normale è ottenibile direttamente dall'equazione del piano: $\frac{1}{\sqrt{5}}[2,\, 1,\, 0]^T$.
  La matrice Q che segue è dunque ortogonale:

$$Q = \begin{pmatrix} -\frac{1}{\sqrt{5}} & 0 & \frac{2}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} & 0 & \frac{1}{\sqrt{5}} \\ 0 & 1 & 0 \end{pmatrix}$$

da cui, ricordando le ben note formule di Algebra lineare, si ottiene il tensore K:

$$K = Q\tilde{K}Q^T$$

Da ultimo, le condizioni al contorno imposte sono state le seguenti: $u_0 = 0$ e $p_0 = y$, con Γ_D corrispondente alle due facce $y = 0$ e $y = 1$, mentre Γ_N comprende le rimanenti quattro.
Tutti i grafici che verranno mostrati nel seguito sono stati ottenuti con il metodo di Raviart-Thomas ibridizzabile di grado $k = 0$, applicato su una griglia di calcolo di 11573 tetraedri.

5.2.3 Caso A: forte permeabilità in direzione normale

La tripletta di coefficienti da considerare è seguente: $\eta = 1$, $\eta_\Gamma = 0.01$, $\hat{\eta} = 1$.
In questa configurazione la resistenza alla filtrazione in direzione normale è molto bassa, da cui un tensore (locale) di permeabilità così costruito:

$$\tilde{K} = \begin{pmatrix} 20 & 0 & 0 \\ 0 & 20 & 0 \\ 0 & 0 & 5 \end{pmatrix}$$

Il termine forzante è stato dapprima posto pari a 4 (come in [CDASC2010]), in seguito modificato in $4(1 - z^2)$ per introdurre una dipendenza esplicita dalla terza coordinata e sfruttare appieno le potenzialità del codice di calcolo 3D.

Sorgente costante Essendo il termine sorgente una funzione costante in tutto il volume, sia all'interno sia all'esterno della frattura, la fenomenologia è squisitamente bidimensionale; per ogni sezione a z costante si ottiene lo stesso profilo di velocità di filtrazione e di pressione. Dalla figura 5.6 si nota che la pressione, a cavallo della frattura, è pressoché continua; l'eterogeneità dei parametri non influisce molto a causa dell'elevata permeabilità in direzione trasversale.
Tale risultato è in ottimo accordo con quanto ottenuto dagli autori in [CDASC2010].

Sorgente dipendente da z Il fenomeno fisico assume una piena connotazione tridimensionale. Rispetto al precedente caso con forzante costante e pari a 4, in questo caso si può notare che i massimi assunti dalla pressione hanno valore più elevato in corrispondenza di quote z piccole (figura 5.7); con il crescere di z, la sorgente tende a 0, da cui un campo di pressione meno intenso.

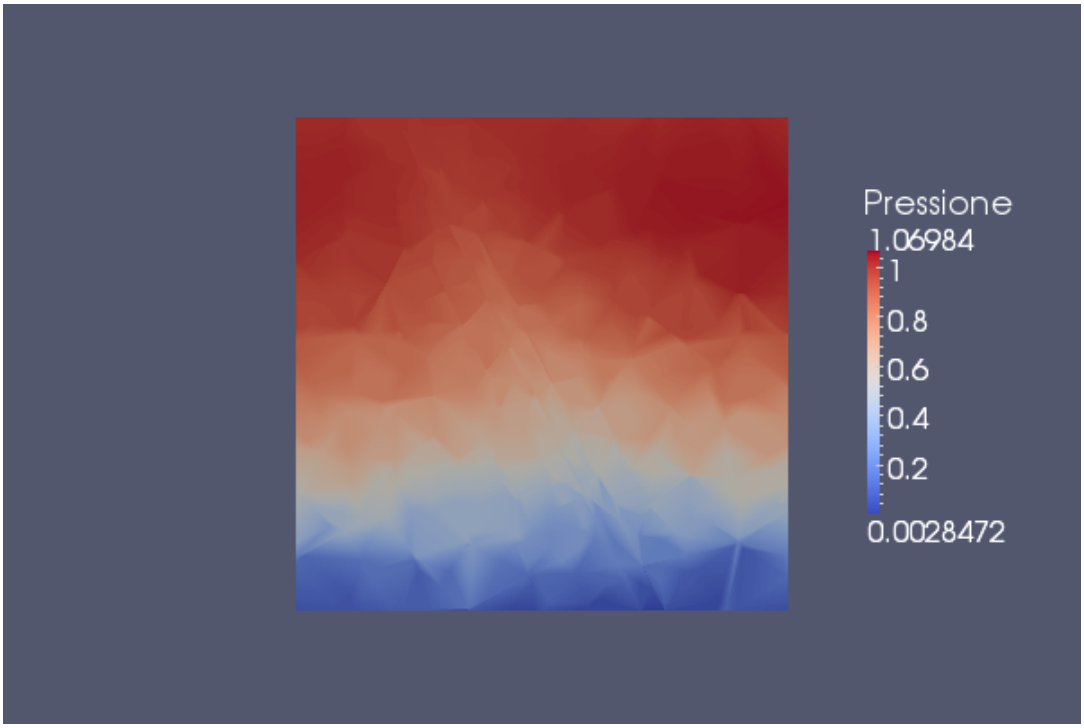

Figura 5.6: Grafico della pressione in quota $z = 0.5$ nel caso di forte permeabilità in direzione normale alla frattura: si può notare come la pressione sia pressoché continua, nonostante la frattura.

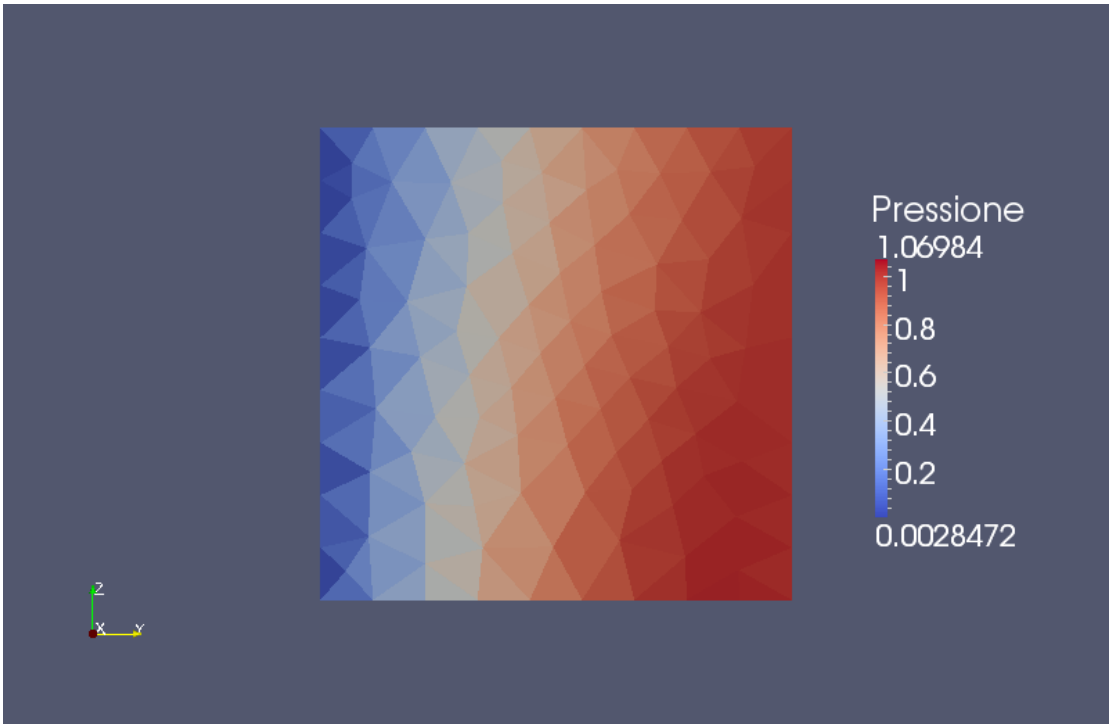

Figura 5.7: Grafico della pressione nel piano yz (asse x uscente), nel caso di forte permeabilità in direzione normale e sorgente variabile con z. Il campo di pressione mantiene un profilo di crescita simile al caso con sorgente costante, ma l'intensità dei massimi è minore al crescere di z, a causa del progressivo calo della sorgente $f_q = 4(1 - z^2)$.

5.2.4 Caso B: forte permeabilità in direzione tangenziale

La tripletta di coefficienti da considerare è seguente: $\eta = 1$, $\eta_\Gamma = 1$, $\hat{\eta} = 0.01$.
In questa configurazione la resistenza alla filtrazione in direzione tangenziale è molto bassa, da cui un tensore (locale) di permeabilità così costruito:

$$\tilde{K} = \begin{pmatrix} 2000 & 0 & 0 \\ 0 & 2000 & 0 \\ 0 & 0 & 0.05 \end{pmatrix}$$

Analogamente al caso 5.2.3, il termine forzante è stato dapprima posto pari a 4, in seguito modificato in $4(1 - z^2)$.

Sorgente costante La figura 5.8 presenta il grafico della pressione: si nota innanzitutto una discontinuità nel passaggio tra la frattura e il dominio circostante; il *range* di valori assunti dalla soluzione numerica mostra che effettivamente la condizione di Dirichlet viene soddisfatta sulle due facce in cui è stata imposta.

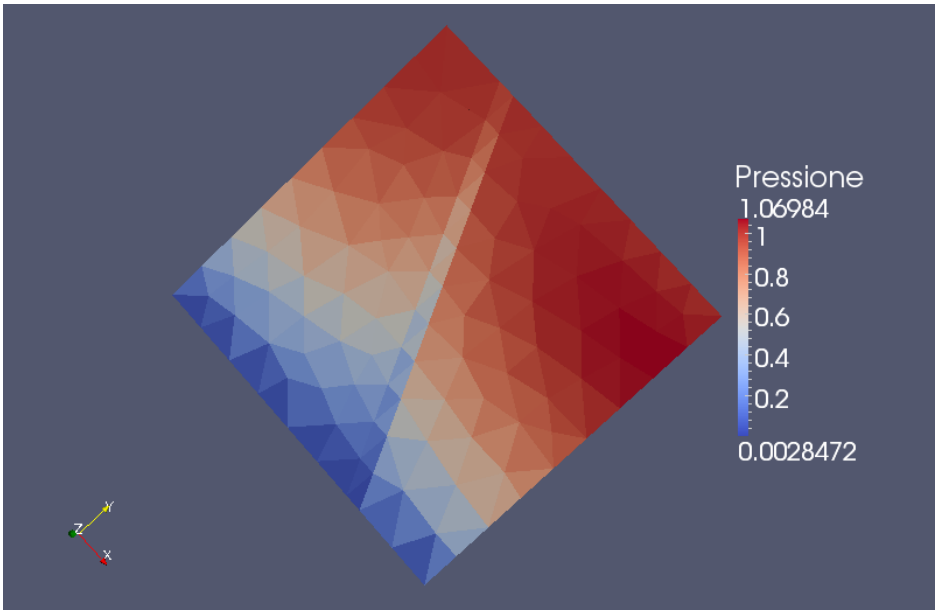

Figura 5.8: Grafico della pressione nel caso di forte permeabilità in direzione tangenziale alla frattura.

La pressione descresce pressoché linearmente (figura 5.9) all'interno della frattura.
Le linee di corrente della velocità di filtrazione entrano nella frattura (figura 5.10), in quanto essa tende a risultare luogo privilegiato di filtrazione in virtù della forte permeabilità tangenziale.
Come nel caso precedente, i risultati ottenuti sono in ottimo accordo con [CDASC2010].

Sorgente dipendente da z La dipendenza da z induce, riguardo alla distribuzione e all'intensità dei massimi del campo di pressione, le medesime conclusioni sulla già rilevate

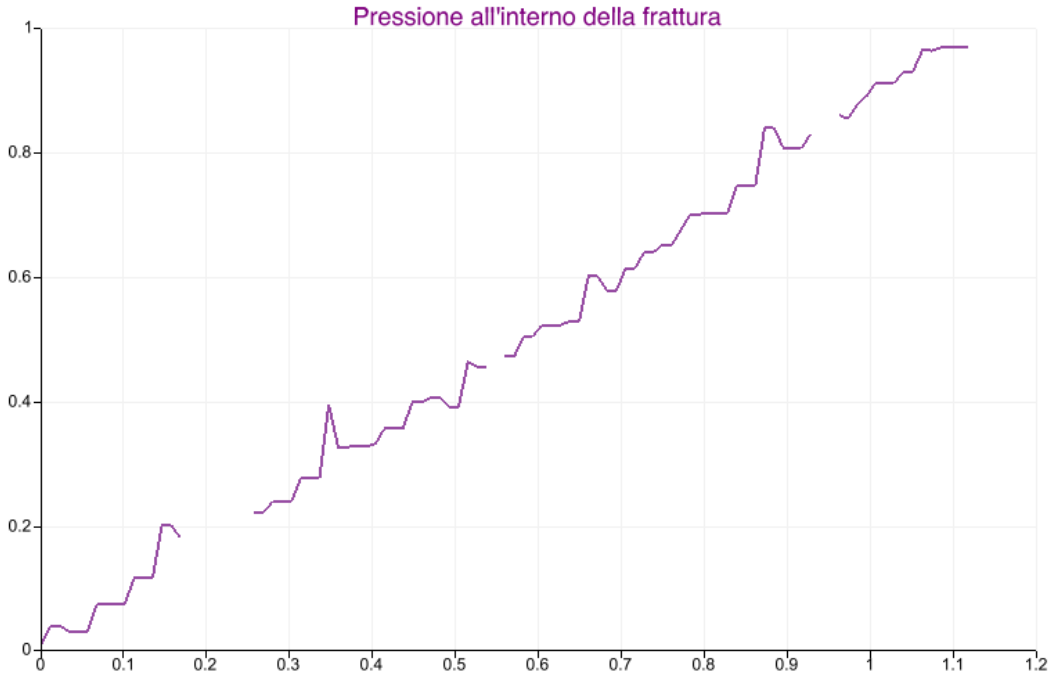

Figura 5.9: Grafico della pressione nel caso di forte permeabilità in direzione tangenziale alla frattura, ottenuto lungo la retta $y + 2x = 1.45$ in quota $z = 0.5$: si può notare l'andamento pressoché lineare del campo di pressione.

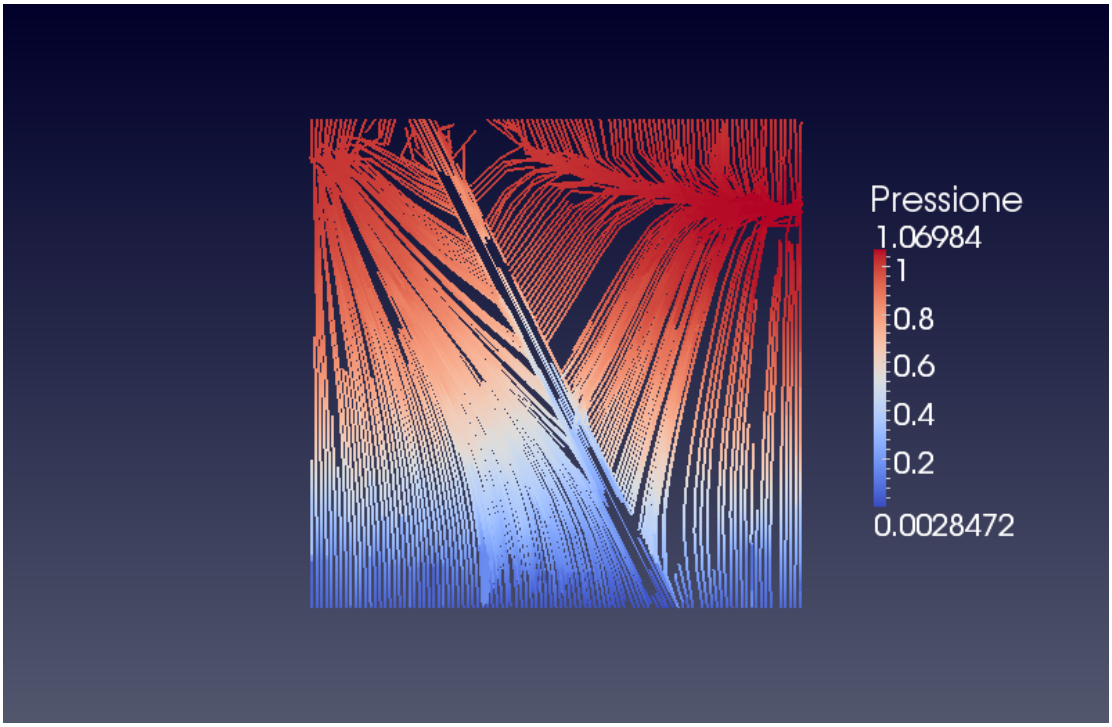

Figura 5.10: Linee di corrente per la velocità di filtrazione sul piano $z = 0.5$: la frattura è luogo privilegiato di filtrazione, a causa dell'elevata permeabilità in direzione tangenziale.

nel caso di forte permeabilità normale.

Capitolo 6

Conclusioni e sviluppi futuri

La trattazione teorica operata nel Capitolo 2 ha messo in luce le principali proprietà dei **metodi *Discontinuous Galerkin***, grazie alla sinossi con la tradizionale versione *Continuous Galerkin*; in modo particolare, si è focalizzata l'attenzione sul **significato dei gradi di libertà** e sul **comportamento esibito dalla soluzione alle interfacce fra elementi adiacenti**: da questo punto di vista, infatti, l'approccio discontinuo è notevolmente diverso rispetto a quello continuo.

La procedura di **ibridizzazione** e la conseguente comparsa della **nuova incognita** λ_h hanno permesso di arricchire la comprensione di **ciò che succede sui bordi degli elementi**, nonché di mostrare che **i metodi misti ibridi**, ancora poco noti, **sono a tutti gli effetti metodi di Galerkin**: pur presentando un'articolazione diversa, a causa sia della contemporanea presenza di incognite di volume e di bordo, sia dell'apparente indipendenza nella definizione di ciascun elemento rispetto ai suoi adiacenti, tali metodi sono caratterizzati da una **formulazione agli spostamenti generalizzata** per una **nuova incognita**, λ_h, **definita sulle frontiere degli elementi**, da cui è possibile calcolare il valore delle incognite interne a ciascun tetraedro. Nondimeno, anche per questi metodi ibridi, sono stati esaminati in dettaglio alcuni aspetti già affrontati con i più tradizionali metodi *Continuous Galerkin*: il processo di assemblaggio della matrice globale a partire dai contributi di ogni singolo elemento, nonché la struttura di sparsità di tale matrice al variare delle funzioni di base che generano lo spazio in cui è ambientata l'incognita λ_h.

Lo studio dettagliato dei metodi *ibridizzabili* e la loro applicazione alle equazioni ellittiche del second'ordine hanno permesso di ottenere, attraverso adeguate simulazioni numeriche descritte nel Capitolo 5, **una *convergenza sperimentale con grado* $k+1$, nell'abbattimento dell'errore rispetto alla norma L^2, per la variabile q**, campo vettoriale che possiede un interessante significato fisico di *flusso* e che riveste in molte applicazioni ingegneristiche un ruolo anche superiore rispetto alla tradizionale variabile scalare u.
Nondimeno, grazie ad un'apposita procedura di **ricostruzione locale**, si è dimostrata anche la **superconvergenza di grado** $k+2$ sulla variabile scalare u, che assume il signi-

ficato fisico di *potenziale*.

Lo studio sin qui condotto non può, tuttavia, dirsi completo; rimangono naturalmente direttrici di sviluppo possibili, alcune delle quali sono:

- **estensione delle simulazioni al caso di trasporto dominante**. L'attuale codice di calcolo prevede, relativamente al metodo LDG ibridizzabile, la possibilità di diverse configurazioni per il parametro di stabilizzazione locale τ, dipendenti sì dal diametro di ciascuna faccia di bordo, ma non dall'intensità e dalla direzione del campo avvettivo presente.
  Con un'adeguata e non difficile estensione, sarebbe possibile applicare il metodo LDG anche al caso di viscosità tendente a 0: con questi parametri, la fisica del problema esibirebbe una natura squisitamente iperbolica. Sarebbe allora possibile studiare il comportamento del metodo nell'approssimazione degli strati limite della soluzione.

- in un futuro decisamente più remoto, **parallelizzazione del codice di calcolo e adattività di griglia**. I metodi misti ibridizzabili esibiscono infatti un'intrinseca località, che li rende più facilmente parallelizzabili; l'adattività di griglia, in termini sia h sia p, potrebbe condurre all'accoppiamento di metodi diversi in regioni diverse della *mesh*, caratterizzate da proprietà della soluzione, note *a priori*, richiedenti un maggiore o minore dispendio computazionale o la scelta di uno specifico metodo numerico.

Appendice A

Richiami analitici

A.1 Considerazioni aggiuntive sull'ibridizzazione

A.1.1 La *conservativity condition*

Si inseriscono qui alcune utili considerazioni sulla terza delle equazioni (precisamente, la (2.6c)) che un generico metodo ibrido deve soddisfare. Tale equazione viene definita in [CGL2009] come ***conservativity condition***.
Supponendo che su una porzione del bordo venga imposta una condizione di Neumann pari a q_N, dopo qualche semplice manipolazione, si può riscrivere la medesima equazione nella forma

$$\sum_{e\in\mathscr{E}_h^i} \langle [[\hat{\boldsymbol{q}}_h \cdot \boldsymbol{n}]], \mu\rangle_e + \langle \hat{\boldsymbol{q}}_h \cdot \boldsymbol{n}, \mu\rangle_{\partial\Omega_N} = \langle \mathsf{q}_\mathsf{N}, \mu\rangle_{\partial\Omega_N} \qquad \forall \mu \in M_h$$

in cui sono stati separati gli integrali di bordo riferiti alla frontiera di Neumann da quelli relativi ai lati interni. Non deve stupire la presenza di un secondo membro non identicamente nullo (com'era invece in (2.6c)), in quanto la trattazione operata nella sezione 2.3.1 ricalca [CGL2009], in cui la frontiera è tutta di Dirichlet.
Si nota immediatamente che – oltre ad imporre le condizioni di Neumann – s'impone che il salto della componente normale della traccia numerica $\hat{\boldsymbol{q}}_h$, definita come

$$[[\hat{\boldsymbol{q}}_h \cdot \boldsymbol{n}]] := \hat{\boldsymbol{q}}_h^+ \cdot \boldsymbol{n}^+ + \hat{\boldsymbol{q}}_h^- \cdot \boldsymbol{n}^-$$

sia nulla attraverso le frontiere di interfaccia tra due elementi adiacenti. Grazie a ciò, **tale traccia numerica è una funzione univoca e non multivoca**, tale da garantire la proprietà di ***conservativity***, come definita in [ABCM2002].

A.1.2 Solutori locali

Nel caso generale in cui sono presenti diffusione, reazione e trasporto con condizioni eventualmente sia naturali sia essenziali, il **primo risolutore locale** si compone delle seguenti

equazioni: data $\mathsf{m} \in \mathsf{M}_h$, si cerca una tripletta $(\mathcal{Q}\mathsf{m}, \hat{\mathcal{Q}}\mathsf{m}, \mathcal{U}\mathsf{m})$

$$\begin{aligned}(\boldsymbol{c}\,\mathcal{Q}\mathsf{m}, \boldsymbol{v})_K + (\boldsymbol{b}\,\mathcal{U}\mathsf{m}, \boldsymbol{v})_K - (\mathcal{U}\mathsf{m}, \nabla\cdot\boldsymbol{v})_K &= -\langle \mathsf{m}, \boldsymbol{v}\cdot\boldsymbol{n}\rangle_{\partial K} \\ -(\mathcal{Q}\mathsf{m}, \nabla\omega)_K + \langle \hat{\mathcal{Q}}\mathsf{m}\cdot\boldsymbol{n}, \omega\rangle_{\partial K} + (d\,\mathcal{U}\mathsf{m}, \omega)_K &= 0\end{aligned}$$

per ogni coppia $(\boldsymbol{v}, \omega) \in \boldsymbol{V}(K) \times W(K)$. Come ormai noto, $\hat{\mathcal{Q}}\mathsf{m}$ può essere definita esplicitamente oppure ricavabile da una terza equazione scritta *ad hoc*. Nel caso dei metodi presentati in questa tesi alla sezione 2.4, la definizione è esplicita, anche se in linea generale ciò non è strettamente necessario.

Il **secondo risolutore locale** è invece così esprimibile:

$$\begin{aligned}(\boldsymbol{c}\,\mathcal{Q}f, \boldsymbol{v})_K + (\boldsymbol{b}\,\mathcal{U}f, \boldsymbol{v})_K - (\mathcal{U}f, \nabla\cdot\boldsymbol{v})_K &= 0 \\ -(\mathcal{Q}f, \nabla\omega)_K + \langle \hat{\mathcal{Q}}f\cdot\boldsymbol{n}, \omega\rangle_{\partial K} + (d\,\mathcal{U}f, \omega)_K &= (f, \omega)_K\end{aligned}$$

per ogni coppia $(\boldsymbol{v}, \omega) \in \boldsymbol{V}(K) \times W(K)$, con analoghe considerazioni su $\hat{\mathcal{Q}}f$.

A.2 Elemento di riferimento e cambio di coordinate

Allo scopo di gestire correttamente le informazioni geometriche e gli integrali numerici, è utile richiamare alcuni concetti importanti riguardo all'*elemento di riferimento* e al *cambio di coordinate* verso un elemento generico della griglia di calcolo.
Per quanto riguarda il caso bidimensionale, l'elemento finito di riferimento è il triangolo rappresentato nella figura A.1.

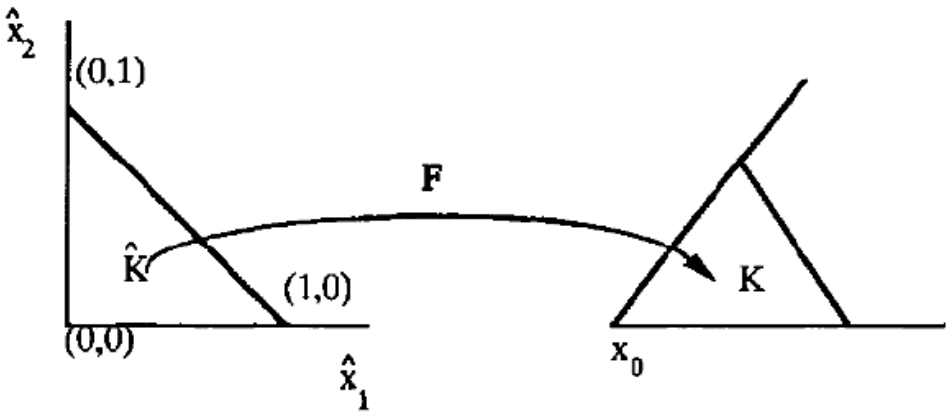

Figura A.1: Trasformazione di coordinate nel caso 2D: a sinistra l'elemento di riferimento, a destra un generico elemento trasformato in $\mathbb{R}^2$.

La seguente **trasformazione affine** F consente di passare dal dominio di riferimento a quello trasformato:

$$F(\hat{x}) = x_0 + B\hat{x}$$

dove B è la **mappa lineare di trasformazione** propria di ciascun elemento della griglia di calcolo ed è così definita:

$$B = \begin{pmatrix} x_1 - x_0 & x_2 - x_0 \\ y_1 - y_0 & y_2 - y_0 \end{pmatrix}$$

con (x_0, x_1, x_2) punti del piano che costituiscono i tre vertici ordinati del triangolo trasformato.
Il valore assoluto del determinante di B coincide con il rapporto di dilatazione superficiale fra il triangolo trasformato e quello di riferimento: per esempio, qualora $|B|$ fosse pari a 4, vorrebbe dire che il triangolo trasformato ha un'area pari a quattro volte quella del triangolo di riferimento.
E' immediato dimostrare (basta ricordare le proprietà dell'Algebra lineare sui determinanti) che modificando l'ordinamento dei vertici che individuano univocamente il triangolo può variare solo il *segno* del determinante, ma non il suo *valore assoluto*.
Analoghi discorsi possono essere fatti per il caso tridimensionale, in cui l'elemento di riferimento è un tetraedro e non più un triangolo. La mappa di trasformazione B diventa ora una matrice 3×3 definita nel modo seguente:

$$B = \begin{pmatrix} x_1 - x_0 & x_2 - x_0 & x_3 - x_0 \\ y_1 - y_0 & y_2 - y_0 & y_3 - y_0 \\ z_1 - z_0 & z_2 - z_0 & z_3 - z_0 \end{pmatrix}$$

con $|B|$ pari al rapporto fra il volume del tetraedro trasformato e quello del tetraedro canonico. In figura A.2 si può vedere molto bene la trasformazione.

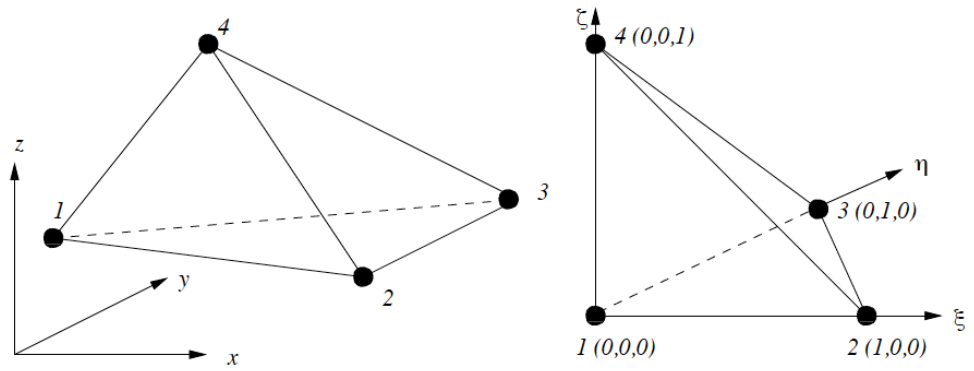

Figura A.2: Trasformazione di coordinate nel caso 3D. Il tetraedro di riferimento è quello identificato dall'origine di $\mathbb{R}^3$ e dai tre punti aventi come coordinate quelle dei tre versori della base canonica.

E' bene sempre ricordare che la **trasformazione di coordinate non è unica**. La scelta operata nel codice di calcolo è la più comune; nondimeno, permette un'interpretazione

geometrica immediata e una comprensione migliore delle trasformazioni che avvengono negli integrali numerici.

A.2.1 Varietà bidimensionali in $\mathbb{R}^3$

A complemento di quanto esposto in precedenza, è utile fornire qualche dettaglio in più per la trattazione delle facce di bordo di un generico elemento della griglia computazionale.
Siano dati i quattro vertici (x_0, x_1, x_2, x_3) che identificano univocamente un tetraedro. Ciascuna delle quattro facce triangolari che costituiscono la frontiera di un qualsivoglia tetraedro può essere pensata in due modi differenti.
In primo luogo, come **porzione del bordo di tale tetraedro**: la faccia dovrà essere attentamente valutata relativamente alle condizioni al contorno (Dirichlet, Neumann, Robin) da imporre. La numerazione locale delle facce è la seguente:

faccia 1	x_0, x_2, x_1
faccia 2	x_1, x_2, x_3
faccia 3	x_2, x_0, x_3
faccia 4	x_0, x_1, x_3

L'ordinamento dei vertici per ciascuna faccia segue la convenzione antioraria, con normale uscente dal piano della faccia.

In secondo luogo, come **dominio di definizione della variabile ibrida di competenza di quella faccia**. In questo caso, però, essa va pensata come una varietà bidimensionale immersa in $\mathbb{R}^3$, trasformata dall'elemento triangolare di riferimento in $\mathbb{R}^2$ attraverso un'opportuna mappa di trasformazione, di dimensioni 3×2. Tale mappa è rappresentata dalla matrice B_f seguente:

$$B_f = \begin{pmatrix} x_B - x_A & x_C - x_A \\ y_B - y_A & y_C - y_A \\ z_B - z_A & z_C - z_A \end{pmatrix}$$

dove la tripletta (x_A, x_B, x_C) rappresenta la terna ordinata di vertici con cui è memorizzata la faccia trangolare. Alle tre lettere A, B, C vanno fatti corrispondere di volta in volta tre punti, secondo la convenzione sull'ordinamento introdotta sopra.
Essendo B_f rettangolare e non quadrata, in questo caso non è possibile calcolare alcun determinante, quindi l'informazione squisitamente geometrica relativa al rapporto di dilatazione areale rispetto al dominio 2D di riferimento va desunta per altra via. Precisamente, detta $\boldsymbol{c_1}$ la prima colonna di B_f e $\boldsymbol{c_2}$ la seconda, tale rapporto di dilatazione è pari alla norma del vettore risultante dal prodotto vettoriale dei due precedenti: $||\boldsymbol{c_1} \times \boldsymbol{c_2}||$.

A.3 La trasformazione di Piola

Per la corretta gestione matematica di funzioni a valori vettoriali definite su elementi diversi da quello di riferimento, è necessario utilizzare la cosiddetta **trasformazione di Piola**.
Supponiamo che sia assegnata la trasformazione affine F descritta in A.2, che mappa il tetraedro di riferimento in un generico tetraedro della *mesh*.
La seguente trasformazione, detta **trasformazione di Piola** $\mathcal{P}$, manda $[L^2(\hat{K})]^d$ in $[L^2(K)]^d$:

$$\hat{\boldsymbol{q}} \mapsto \boldsymbol{q}(\boldsymbol{x}) := \frac{1}{|det(B)|} B\hat{\boldsymbol{q}}(\hat{\boldsymbol{x}})$$

Questo operatore è molto utile nella manipolazione delle funzioni a valori vettoriali utilizzate per approssimare campi vettoriali, quali, ad esempio, la velocità di filtrazione attraverso un mezzo poroso nel problema di Darcy.
La trasformazione esibisce alcune fondamentali proprietà nel cambio di coordinate per l'integrazione. In particolare, si scelgano $\hat{v} \in H^1(\hat{K})$, $\hat{\boldsymbol{q}} \in H(\widehat{\mathrm{div}}, \hat{K})$; sia $v := \hat{v} \circ F^{-1}$, $\boldsymbol{q} := \mathcal{P}(\hat{\boldsymbol{q}})$; allora valgono le relazioni seguenti:

$$\begin{aligned}
\int_K \boldsymbol{q} \cdot \nabla v \,\mathrm{d}x &= \int_{\hat{K}} \hat{\boldsymbol{q}} \cdot \hat{\nabla} \hat{v} \,\mathrm{d}\hat{x} \\
\int_K \nabla \cdot \boldsymbol{q}\, v \,\mathrm{d}x &= \int_{\hat{K}} \hat{\nabla} \cdot \hat{\boldsymbol{q}}\, \hat{v} \,\mathrm{d}\hat{x} \\
\int_{\partial K} \boldsymbol{q} \cdot \boldsymbol{n}\, v \,\mathrm{d}\sigma &= \int_{\partial \hat{K}} \hat{\boldsymbol{q}} \cdot \hat{\boldsymbol{n}}\, \hat{v} \,\mathrm{d}\hat{\sigma}
\end{aligned}$$

Nondimeno, la trasformata $\mathcal{P}$ è un isomorfismo da $H(\widehat{\mathrm{div}}, \hat{K})$ in $H(\mathrm{div}, K)$.
La dimostrazione dei precedenti asserti non è complicata: basta semplicemente esplicitare correttamente le relazioni esistenti tra le coordinate di riferimento e quelle trasformate e ricordare il modulo del determinante della matrice jacobiana di trasformazione (jacobiana che in questo caso coincide, vista la natura *affine* della mappa, con B stessa) che compare nel cambio di coordinate.
Le precedenti relazioni conducono a due importanti conclusioni:

1. una volta scelte sull'*elemento di riferimento* le funzioni di base dello spazio finito-dimensionale che serve ad approssimare l'incognita vettoriale, **è sufficiente memorizzare il valore della divergenza di tali vettori solamente nei nodi di quadratura dell'elemento canonico**, senza dover effettuare alcuna riscalatura. Analogo discorso nel caso della **componente normale su ciascuna faccia di bordo**.
   Ciò dà luogo ad un importante risparmio computazionale, in particolar modo quando la griglia è molto fine;

2. le funzioni di base definite sul dominio "fisico", ottenute applicando a quelle definite sul dominio di riferimento la trasformata di Piola, **conservano *per costruzione* la continuità della loro componente normale attraverso il bordo del tetraedro**, quindi danno luogo ai corretti spazi funzionali ad elementi finiti per l'approssimazione numerica.

A.4 Ricostruzione locale di flusso e potenziale

A partire da quantità definite localmente su ciascun tetraedro della griglia computazionale, è possibile ottenere nuove grandezze (scalari e vettoriali) attraverso un adeguato *post-processing*, avente lo scopo di garantire un profilo di convergenza migliore, a parità di finezza di griglia.

A.4.1 Il flusso ricostruito $\boldsymbol{q}_h^*$

Per quanto riguarda l'incognita vettoriale flusso $\boldsymbol{q}_h$, la ricostruzione implementata nel codice di calcolo fa riferimento al cosiddetto proiettore di Raviart-Thomas, spiegato in dettaglio in [BF1991], § III.3, pag. 127. L'idea sottostante è quella di ricostruire un campo vettoriale appartenente allo spazio di Raviart-Thomas di grado k pari a quello utilizzato nel calcolo di $\boldsymbol{q}_h$, che esibisca però la continuità della componente normale all'interfaccia di bordo.
Precisamente, l'obiettivo è quello di trovare un vettore $\boldsymbol{q}_h^*$ appartenente allo spazio lineare $[\mathscr{P}_k(K)]^d + \boldsymbol{x}\mathscr{P}_k(K)$, tale da soddisfare le seguenti equazioni:

$$\begin{aligned} \langle(\boldsymbol{q}_h^* - \hat{\boldsymbol{q}}_h)\cdot\boldsymbol{n}, \mu\rangle_e &= 0 \quad \forall \mu \in \mathscr{P}_k(e)\ \forall \text{ faccia } e \text{ di } K \\ (\boldsymbol{q}_h^* - \boldsymbol{q}_h, \boldsymbol{v})_K &= 0 \quad \forall \boldsymbol{v} \in [\mathscr{P}_{k-1}(K)]^d \end{aligned}$$

Il precedente sistema consente di determinare univocamente una funzione dello spazio di Raviart-Thomas, vista l'unisolvenza dei gradi di libertà così imposti. Per una dettagliata trattazione sulla costruzione dello spazio e per la dimostrazione di unisolvenza, si può fare riferimento all'articolo originale [RT1975].
La suddetta procedura di ricostruzione locale induce due importanti sottolineature:

1. si ottiene un campo vettoriale in $H(\mathrm{div}, \Omega_h)$, da cui la possibilità di effettuare, nei casi in cui la soluzione esatta è nota, un possibile profilo di decadimento dell'errore nella norma specifica di questo spazio.
   Se il metodo scelto per risoluzione del problema è RT-H oppure BDM-H, non c'è necessità di alcuna ulteriore ricostruzione, in quanto il campo $\boldsymbol{q}_h$ soddisfa *per costruzione* la condizione di continuità della componente normale attraverso la frontiera.

2. le funzioni di base dello spazio di Raviart-Thomas di grado k comprendono polinomi di grado $k+1$: è pertanto obbligatorio utilizzare, per l'integrazione numerica, formule con esattezza superiore rispetto a quelle usate nei calcoli (con LDG-H).

A.4.2 Il potenziale ricostruito u_h^*

L'idea sottostante alla ricostruzione del potenziale è quella di risolvere un nuovo problema *locale*, su ciascun tetraedro della *mesh*, sulla falsariga della forma forte del problema differenziale di diffusione-trasporto-reazione.
Lo spazio ad elementi finiti impiegato per la ricostruzione sarà sempre polinomiale, ma arricchito dalle funzioni con grado $k+1$. Matematicamente la procedura di ricostruzione è la seguente: trovare $u_h^* \in \mathscr{P}_{k+1}(K)$ t.c., $\forall\, w \in \mathscr{P}_{k+1}(K)$

$$(\boldsymbol{a}\,\nabla u_h^* - \boldsymbol{\beta}\,u_h^*, \nabla w)_K + (d\,u_h^*, w)_K = (f, w)_K - \langle \hat{\boldsymbol{q}}_h \cdot \boldsymbol{n}, w\rangle_{\partial K}$$

dove $\boldsymbol{a}$, $\boldsymbol{\beta}$, d sono i contributi di diffusione, trasporto e reazione aventi le usuali proprietà necessarie per la buona positura del problema differenziale.
Si noti che le condizioni al bordo sono in questo caso naturali, in quanto il secondo membro incorpora naturalmente il flusso attraverso il bordo, ottenuto a partire dalla grandezza vettoriale $\hat{\boldsymbol{q}}_h$ già calcolata ai passi precedenti della risoluzione numerica.
Qualora la reazione fosse identicamente nulla, la buona positura non sarebbe più garantita, in quanto verrebbe meno la coercività e la soluzione sarebbe di conseguenza definita a meno di una costante additiva; per poter determinare una ben precisa soluzione, si può imporre che la media dell'incognita u_h^* eguagli la media della grandezza u_h. Dal punto di vista numerico, nel caso $k = 0$, alla media di u_h viene sostituita la sommatoria $\frac{1}{d}\sum_{e\in\partial K} \hat{u}_h|_e$, così come suggerito in [CDG2008].

A.5 La generazione della griglia con GMSH

Il generatore di griglia utilizzato durante questa tesi è GMSH, *software* gratuitamente scaricabile dal sito web `http://geuz.org/gmsh/`.
La filosofia seguita per costruire la *mesh* è semplice: bisogna partire dalle entità geometricamente più semplici (punti, linee) per poi arrivare a quelle di dimensione maggiore (superfici, volumi).
Si presti attenzione al fatto che il codice è pensato per leggere un *file* avente estensione `.msh`, scritto in formato ASCII secondo le specifiche del capitolo 9 del manuale del programma, scaricabile sempre da `http://geuz.org/gmsh/`. Inoltre, è richiesto all'utente di fornire un file in cui siano definite alcune "**Physical entities**", come nell'esempio seguente:

```
$PhysicalNames
2
```

```
2 1 ''Dirichlet''
3 2 ''Regione_intera''
$EndPhysicalNames
```

dove il primo intero indica quante sono le "Physical entities", mentre le righe che seguono indicano i corrispondenti nomi, accompagnate dalla dimensione spaziale (primo indice) e dall'identificativo dell'entità (secondo indice).
In GMSH è possibile utilizzare il concetto di "entità fisica" per raggruppare oggetti geometrici che condividono una specifica proprietà (astratta o fisica), quale ad esempio una determinata condizione al bordo. Il *file* con estensione `.msh` è stato generato da un omonimo *file* con estensione `.geo`, in cui le facce di bordo (che devono essere necessariamente definite, pena errore nella parte C++) sono raggruppate a seconda delle condizioni al bordo (Dirichlet con identificativo pari a 0, Neumann pari a 1, Robin pari a 2) su di esse imposte.
Per forzare il generatore di griglia a restituire nel file `.msh` tutte le informazioni sui tetraedri generati e non soltanto le entità geometriche raggruppate come "entità fisiche" (questa sarebbe la scelta di *default* di GMSH), si è inserita un'ulteriore entità fisica, battezzata "Regione_intera", che raccoglie tutti i tetraedri della griglia di calcolo.

Appendice B

Codice di calcolo

Per utilizzare il codice di calcolo sono necessari alcuni passi.

Innanzitutto, bisogna compilare la parte scritta in linguaggio C++, in modo che venga resa disponibile al *prompt* di *Octave* una funzione caricata dinamicamente (DLD, *dynamically linked function*). Essa ha il compito di precalcolare tutte le quantità necessarie alla risoluzione dell'equazione di diffusione-trasporto-reazione con uno dei metodi descritti nella sezione 2.4.
Per procedere correttamente all'installazione, leggere attentamente i *files* `README` e `INSTALL` contenuti nel pacchetto; nel secondo, in particolare, sono indicate le dipendenze presenti, in termini di librerie C++ e altri pacchetti *Octave*.
Alcuni dettagli più specifici sono già stati forniti nella sezione 3.7.

Una volta completata la fase di installazione, l'utente è chiamato all'inserimento dei parametri necessari alla corretta risoluzione del problema.
Il *file* principale è `risolutore_fem_hyb.m`, nel quale bisogna modificare, nell'ordine, i seguenti campi:

- `set_bc.m`: questo *file* consente di impostare le condizioni al contorno per il problema differenziale. Attualmente all'utente è consentito di impostare condizioni di Dirichlet e Robin; la condizione di Neumann è implementabile imponendo $\gamma = 0$.
  Qualora si desiderasse risolvere un problema con puro Dirichlet, è necessario modificare la variabile globale `tipo_bc_conserv_condition` come indicato, per segnalare al programma questa scelta.

- `set_parametri_fisici.m`: questo *file* consente di impostare i parametri fisici (diffusione, trasporto, reazione, forzante) per il problema differenziale. All'utente è concessa la possibilità di inserire funzioni non necessariamente costanti, bensì variabili rispetto alle tre coordinate indipendenti x, y, z.
  Si presti attenzione al caso in cui la reazione fosse identicamente nulla: in questa configurazione, allora, è necessario modificare il valore della variabile globale

`is_null_reazione`, in modo che il programma, all'atto della ricostruzione del potenziale u_h^* (come spiegato in A.4.2), scelga il valor medio della nuova incognita da determinarsi.

- `set_soluz_esatta.m`: questo *file* consente di impostare l'espressione analitica della soluzione esatta, qualora nota. In caso contrario, impostare a 0 la variabile `exist_soluz_esatta` e commentare la riga sottostante.

- `set_parametri_cplusplus.m`: questo *file* consente di impostare tutti i parametri di simulazione relativi alla scelta del metodo numerico, del grado polinomiale, del file contenente la griglia di calcolo, dell'esattezza della quadratura numerica (di volume e di bordo) e ai gradi polinomiali per la visualizzazione delle soluzioni.
  Si presti attenzione alla scelta del metodo numerico desiderato: 0 per LDG-H, 1 per BDM-H e 2 per RT-H. Qualunque sia la scelta operata, bisogna assicurarsi che la funzione DLD descritta poco sopra contenga al suo interno la generazione degli spazi funzionali in maniera coerente con la scelta operata. Se si desidera passare da un metodo all'altro (più precisamente, vista la struttura dei metodi, da LDG-H/BDM-H a RT-H o viceversa), non è sufficiente modificare il campo `nome_metodo` con il corrispondente intero, ma è obbligatorio modificare il *file* `FE_MxHyb-1.0.0/src/metodo.cpp` con la medesima scelta e **rieseguire la procedura di installazione del pacchetto**.
  Per quanto riguarda la scelta del grado di esattezza richiesto alla formula di quadratura numerica, la selezione è lasciata all'utente; è bene ricordare, comunque, che il risolutore andrà a calcolare anche le ricostruzioni del flusso e del potenziale, che prevedono spazi polinomiali con grado superiore. L'esattezza va impostata a partire da qui, nonché dall'eventuale presenza di coefficienti non lineari, che richiedono tipicamente una formula con grado di esattezza più elevato.
  Da ultimo, per quanto riguarda la scelta del grado polinomiale per la visualizzazione delle soluzioni, *Paraview* consente allo stato attuale l'utilizzo di funzioni lineari e quadratiche. Tuttavia, nel caso di grado polinomiale elevato, non tutte le combinazioni (grado globale, grado locale) sono possibili; il grado locale 2 può essere scelto se la funzione da visualizzare (sia essa propria del metodo o derivante dal *post-processing*, scalare o vettoriale) è un polinomio di tale grado sull'elemento o sulla faccia, oppure – per la sola variabile ibrida – se l'incognita è di grado 4. In tutti gli altri casi è necessario impostare grado locale 1 per la visualizzazione.

- `tau_stab`: questa grandezza va modificata durante l'assemblaggio, al variare del tetraedro considerato. La funzione di stabilizzazione locale τ è identicamente nulla per i metodi RT-H e BDM-H, mentre va selezionata opportunamente nel caso LDG-H. All'utente è lasciata ampia libertà di scelta, nella consapevolezza che τ diverse producono profili asintotici di convergenza diversi (si veda, a tal proposito, nel caso 2D, [CDGRS2009], § 3.1).

Bibliografia

[AB1985] D. N. Arnold, F. Brezzi (1985), *Mixed and nonconforming finite element methods: implementation, postprocessing and error estimates*, Mathematical Modelling and Numerical Analysis, Vol. 19, No. 1, pp. 7–32.

[ABCM2002] D. N. Arnold, F. Brezzi, B. Cockburn, L. D. Marini (2002), *Unified analysis of discontinuous Galerkin methods for elliptic problems*, SIAM Journal of Numerical Analysis, Vol. 39, No. 5, pp. 1749–1779.

[BF1991] F. Brezzi, M. Fortin (1991), *Mixed and hybrid finite element methods*, Springer-Verlag, New York.

[CIA1978] P. G. Ciarlet (1978), *The finite element method for elliptic problems*, North-Holland Publishing Company, Amsterdam.

[COCK2003] B. Cockburn (2003), *Discontinuous Galerkin methods*, School of Mathematics, University of Minnesota, pp. 1–25.

[CDG2008] B. Cockburn, B. Dong, J. Guzman (2008), *A superconvergent LDG-hybridizable Galerkin method for second-order elliptic problems*, Mathematics of Computation, Vol. 77, No. 264, pp. 1887–1916.

[CDGRS2009] B. Cockburn, B. Dong, J. Guzman, M. Restelli, R. Sacco (2009), *A hybridizable Discontinuous Galerkin method for steady-state convection-diffusion-reaction problems*, SIAM Journal on Scientific Computing, Vol. 31, No. 5, pp. 3827–3846.

[CG2004] B. Cockburn, J. Gopalakrishnan (2004), *A characterization of hybridized mixed methods for second order elliptic problems*, SIAM Journal of Numerical Analysis, Vol. 42, No. 1, pp. 283–301.

[CGL2009] B. Cockburn, J. Gopalakrishnan, R. Lazarov (2009), *Unified hybridization of Discontinuous Galerkin, mixed, and Continuous Galerkin methods for second order elliptic problems*, SIAM Journal of Numerical Analysis, Vol. 47, No. 2, pp. 1319–1365.

[CKS2000] B. Cockburn, G. E. Karniadakis, C.-W. Shu (2000), *The development of Discontinuous Galerkin methods*, in *Discontinuous Galerkin methods. Theory, computation and applications*, Springer-Verlag, Berlin-Heidelberg.

[CDASC2010] C. D'Angelo, A. Scotti (2010), *A mixed finite element method for Darcy flow in fractured porous media with non-matching grids*, MOX Report No. 40-2010, Dipartimento di Matematica, Politecnico di Milano.

[GR2009] C. Geuzaine, J.-F. Remacle (2009), *Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities*, International Journal for Numerical Methods in Engineering, Vol. 79, Issue 11, pp. 1309–1331.

[LEV2004] R. J. LeVeque (2004), *Finite volume methods for hyperbolic problems*, Cambridge University Press, Cambridge.

[LI2006] B. Q. Li (2006), *Discontinuous finite elements in fluid dynamics and heat transfer*, Springer-Verlag, Londra.

[QUA2008] A. Quarteroni (2008^4), *Modellistica numerica per problemi differenziali*, Springer-Verlag Italia, Milano.

[QV2008] A. Quarteroni, A. Valli (2008^2), *Numerical approximation of partial differential equations*, Springer-Verlag, Berlin-Heidelberg.

[RT1975] P. A. Raviart, J. M. Thomas (1975), *A mixed finite element method for second-order elliptic problems*, in I. Galligani, E. Magenes (edit.) (1977), *Mathematical aspects of finite element methods. Proceedings of the Conference held in Rome, December 10-12, 1975*, Springer-Verlag, Berlin-Heidelberg.

[RIV2008] B. Rivière (2008), *Discontinuous Galerkin methods for solving elliptic and parabolic equations. Theory and implementation*, SIAM – Frontiers in Applied Mathematics.

[SAL2007] S. Salsa (2007), *Equazioni a derivate parziali. Metodi, modelli, applicazioni*, Springer-Verlag Italia, Milano.