

LAUREA MAGISTRALE
IN INGEGNERIA MATEMATICA

Progetto per il corso di
Programmazione Avanzata per il Calcolo Scientifico.



**Implementazione in LifeV dell'algoritmo di
Riduzione Gerarchica di Modello**

Progetto svolto da:
Matteo Carlo Maria Aletti
Matr. 783045
Andrea Bortolossi
Matr. 783023

Anno Accademico 2012–2013

Indice

1	Design relazione	2
1.1	Temi di programmazione importanti	2
2	Introduzione	3
2.1	Nozioni base	3
2.2	Forma matriciale	5
2.3	Implementazione integrali	6
2.4	Ipotesi	7
3	Descrizione classi	9
3.1	Modalspace	9
3.1.1	I membri	9
3.2	HiModAssembler	12
3.3	Basis1dAbstract	13
4	Conti 3D	16

Capitolo 1

Design relazione

1.1 Temi di programmazione importanti

- Spostamento dell'ereditarietà;
- Eigensprovider
- Utilizzo di AddrhsHiPrec
- Generalizzazione e non dei coefficienti
- Difficoltà nell'includere il cerchio
- Factory per le basi educate
- La ricerca degli zeri per basi educate

Capitolo 2

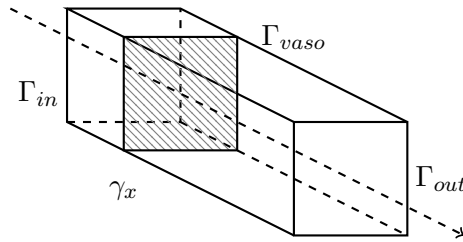
Introduzione

2.1 Nozioni base

L'obiettivo primale del progetto è stato di implementare in LifeV un risolutore ADR 3D, basato sulla tecnica di Riduzione Gerarchica di Modello. Il problema trattato è il seguente:

$$\begin{cases} -\mu\Delta u + \mathbf{b} \cdot \nabla u + \sigma u = f & \text{in } \Omega \\ u = u_{in} & \text{su } \Gamma_{in} \\ \frac{\partial u}{\partial \mathbf{n}} = 0 & \text{su } \Gamma_{out} \\ u = 0 & \text{su } \Gamma_{vaso} \end{cases} \quad (2.1)$$

$$\Omega = \bigcup_{x \in \Omega_{1D}} \gamma_x \quad (2.2)$$



Si consideri il dominio Ω , come l'unione di slice poste trasversalmente alla direzione longitudinale del tubo a sezione rettangolare, la quale verrà indicata d'ora in poi con Ω_{1D} :

Lungo le slice γ_x vengono utilizzate funzioni spaziali differenti rispetto a quelle utilizzate lungo Ω_{1D} . Si consideri infatti per Ω_{1D} , lo spazio funzionale $V_{1D} = H_{\Gamma_{in}}^1(\Omega_{1D})$, mentre sulla generica γ_x si introducano le basi modali $\{\varphi_k(y, z)\}$ ortonormali in $L^2(\gamma_x)$, con $k \in \mathbb{N}$. Quest'ultime definiscono su γ_x

lo spazio funzionale $V_{\gamma_x} := \text{span}\{\varphi_k\}$. Definiamo ora il sottospazio generato solo dai primi m modi ovvero $V_{\gamma_x}^m := \text{span}\{\varphi_1, \dots, \varphi_m\}$ e combiniamolo con V_{1D} , ottenendo il seguente spazio ridotto:

$$V_m := \left\{ v_m(x, y, z) = \sum_{k=1}^m \varphi_k(y, z) \tilde{v}_k(x), \text{ con } \tilde{v}_k \in V_{1D} \right\} \quad (2.3)$$

L'ortogonalità in $L^2(\gamma_x)$ implica che i coefficienti \tilde{v}_k in (2.3) sono il risultato del seguente prodotto scalare per $k = 1, \dots, m$:

$$\tilde{v}_k(x) = \int_{\gamma_x} \varphi_k(y, z) v_m(x, y, z) dy dz$$

La convergenza di una soluzione u_m tale che soddisfi il problema (2.1) è garantita osservando che:

- $V_m \subset V \forall m \in \mathbb{N}$, ossia che lo spazio ridotto V_m è conforme in V ;
- $\lim_{x \rightarrow +\infty} \left(\inf_{v_m \in V_m} \|v - v_m\| \right) = 0$ per ogni $v \in V$, ossia che vale la proprietà di approssimazione di V_m rispetto a V ;

È possibile dimostrare che le ipotesi di conformità e approssimazione sono ancora valide in una trattazione con dato di Dirichlet non omogeneo sulle pareti del tubo ([?]).

2.2 Forma matriciale

La risoluzione del problema ADR può avvenire quindi sullo spazio ridotto V_m . Dunque, per ogni $m \in \mathbb{N}$ si riconosca il seguente problema ridotto del problema originale (2.1), trovare $u_m \in V_m$ tale che $\forall v_m \in V_m$:

$$\int_{\Omega} (\mu \nabla u_m \nabla v_m + \mathbf{b} \nabla u_m v_m + \sigma u_m v_m) d\Omega = \int_{\Omega} f v dxdy \quad (2.4)$$

Si adoperi l'espansione tramite i coefficienti di Fourier della $u_m(x, y, z) = \sum_{j=k}^m \tilde{u}_j(x) \varphi_j(y, z)$ dove:

$$\tilde{u}_j(x) = \int_{\gamma(x)} u_m(x, y, z) \varphi_j(y, z) dydz$$

e si considerino le funzioni test $v_m = \vartheta(x) \varphi_k(y, z)$ con $\vartheta(x) \in V_{1D}$ e $k = 1, \dots, m$. Il problema assume la seguente forma:

$$\begin{aligned} & \sum_{j=1}^m \left[\int_{\Omega} \mu \nabla (\tilde{u}_j(x) \varphi_j(y, z)) \nabla (\vartheta(x) \varphi_k(y, z)) dxdydz \right. \\ & + \int_{\Omega} \mathbf{b} \nabla (\tilde{u}_j(x) \varphi_j(y, z) \vartheta(x) \varphi_k(y, z)) dxdydz \\ & \left. + \int_{\Omega} \sigma \tilde{u}_j(x) \varphi_j(y, z) \vartheta(x) \varphi_k(y, z) dxdydz \right] \\ & = \int_{\Omega} f \vartheta(x) \varphi_k(y, z) dxdydz \end{aligned} \quad (2.5)$$

Svolgendo l'operatore gradiente si ottiene:

$$\begin{aligned} & \sum_{j=1}^m \left[\int_{\Omega} \mu (\partial_x \tilde{u}_j \partial_x \vartheta \varphi_j \varphi_k + \tilde{u}_j \vartheta \partial_y \varphi_j \partial_y \varphi_k + \tilde{u}_j \vartheta \partial_z \varphi_j \partial_z \varphi_k) dxdydz \right. \\ & + \int_{\Omega} (b_1 \partial_x \tilde{u}_j \varphi_j + b_2 \tilde{u}_j \partial_y \varphi_j + b_3 \tilde{u}_j \partial_z \varphi_j) \vartheta \varphi_k dxdydz \\ & \left. + \int_{\Omega} \sigma \tilde{u}_j \vartheta \varphi_j \varphi_k dxdydz \right] \\ & = \int_{\Omega} f \vartheta \varphi_k dxdydz \end{aligned} \quad (2.6)$$

Definito N il numero di nodi scelti uniformemente distribuiti lungo Ω_{1D} , si determina una partizione T_h , dove $h = |\Omega_{1D}|/(N - 1)$ è il passo spaziale. Introduciamo lo spazio agli elementi finiti lungo Ω_{1D} definito come segue

$$X_h^r = \{\psi_h \in C^0(\Omega_{1D}) : \psi_h|_K \in \mathbb{P}_r, \forall K \in T_h\}$$

Nella successiva implementazione del metodo si è considerato per semplicità, una base F.E.M. di primo grado. Possiamo quindi esprimere i coefficienti di Fourier nel seguente modo: $\tilde{u}_j(x) = \sum_{s=1}^N u_{js} \psi_s(x)$.

Si ottiene dunque la formulazione matriciale del nostro problema, trovare $\mathbf{u} \in \mathbb{R}^{N*m}$ tale che $\forall \psi_l$ e $\forall \varphi_k$, con $l = 1, \dots, N$ e $k = 1, \dots, m$ si ha che:

$$\begin{aligned} \sum_{j=1}^m \sum_{s=1}^N u_{js} \left[\int_{\Omega} \mu (\partial_x \psi_s \partial_x \psi_l \varphi_j \varphi_k + \psi_s \psi_l \partial_y \varphi_j \partial_y \varphi_k + \psi_s \psi_l \partial_z \varphi_j \partial_z \varphi_k) dx dy dz \right. \\ \left. + \int_{\Omega} (b_1 \partial_x \psi_s \varphi_j + b_2 \psi_s \partial_y \varphi_j + b_3 \psi_s \partial_z \varphi_j) \psi_l \varphi_k dx dy dz \right. \\ \left. + \int_{\Omega} \sigma \psi_s \psi_l \varphi_j \varphi_k dx dy dz \right] \\ = \int_{\Omega} f \psi_l \varphi_k dx dy dz \quad (2.7) \end{aligned}$$

Si osservi che il doppio indice "js", in realtà scorre un vettore, la rimappatura in un solo indice può facilmente essere dedotta ottenendo che $[\mathbf{u}]_{js} = \mathbf{u}[(j-1)N + s]$. La matrice generata ha quindi dimensioni $(mN)^2$, tuttavia fissata la frequenza delle soluzioni e della funzione test è possibile identificare un blocco che corrisponde ad un problema monodimensionale. Se utilizziamo, in direzione x, gli elementi finiti di grado 1, il blocco risulta tridiagonale e, in questo caso, la matrice ha un numero di elementi non zero pari a $m^2(3N-2)$. Il pattern di sparsità per un caso con $m=3$ e $N=14$ è riportato in figura ???. La matrice dei coefficienti è dunque sparsa ed inoltre il pattern è noto a priori, queste informazioni hanno permesso un assemblaggio più veloce in sede implementativa.

In generale il problema che si porrebbe ora sarebbe la scelta della base modale. Esistono svariati metodi al fine di determinare la natura della base modale, tuttavia questa problematica va al di fuori degli scopi di questo elaborato. Seguendo le linee guida in (e qua ci autocitiamo!!!!) scegliamo la base modale in grado di garantire le condizioni di parete:

$$\varphi_j(y, z) = \sin\left(\frac{\alpha}{\pi L_y} y\right) \sin\left(\frac{\beta}{\pi L_z} z\right) \quad \lambda_j = \alpha^2 + \beta^2 \quad (2.8)$$

2.3 Implementazione integrali

Nel caso i coefficienti del problema ADR siano dipendenti dalla sola coordinata x o risultino fattorizzabili lungo la direzione x e il piano ortogonale, il

risultato finale di HiMod è la trasformazione di un problema ADR full 3D a m^2 problemi ADR 1D accoppiati con coefficienti modificati opportunamente dalle funzioni modali a seconda della coppia di frequenze considerata. Nel caso non ricadiamo in tale ipotesi vale comunque la scomposizione in problemi 1D ma risulta più delicata l'integrazione. **Nel caso si fattorizza anche μ proiettandola sulle basi modali, non penso che dia dei buoni risultati, tuttavia è fattibile**

$$\begin{aligned} \frac{\partial_x \psi_s \partial_x \psi_l}{\psi_s \psi_l} &= \frac{\int_{\gamma_x} \mu \varphi_j \varphi_k dydz}{\int_{\gamma_x} \varphi_j \varphi_k dydz} \\ &\quad \int_{\gamma_x} (\mu \partial_y \varphi_j \partial_y \varphi_k + \mu \partial_z \varphi_j \partial_z \varphi_k + b_2 \partial_y \varphi_j \varphi_k + b_3 \partial_z \varphi_j \varphi_k + \sigma \varphi_j \varphi_k) dydz \end{aligned} \quad (2.9)$$

2.4 Ipotesi

Le ipotesi alla base di questo progetto sono le seguenti:

- Il dominio di calcolo è un parallelepipedo che si estende nell'ottante positivo.
- I coefficienti della forma sono assunti costanti.
- Viene risolto un problema ADR stazionario con condizioni di inflow di tipo Dirichlet e di outflow di tipo Neumann omogeneo.
- Condizioni sulle pareti omogenee (estensione e non omogenee non difficile ma cmq costanti lungo x)

La forma del dominio considerato ci consente agilmente di applicare le tecniche di separazione di variabili e utilizzare la teoria delle basi educate. Più delicata risulterebbe la gestione di condizioni di bordo sulla parete del vaso, nel caso di sezione a forma generica. Nei capitoli successivi verranno accennate le difficoltà che presenta questa tematica. Anche nel caso di generalizzazione dei coefficienti della forma, viene presentata una soluzione possibile, tuttavia il codice è strutturato per l'utilizzo di coefficienti costanti. In ogni caso consideriamo termini forzanti non costanti lungo il dominio. Per quanto il problema a sezione cilindrica potesse sembrare uno stretto parente del parallelepipedo così non è; renderemo chiare le principali differenze, insite nell'equazione risultante dalla separazione di variabili. Per quanto riguarda le condizioni di inflow, il codice permette di applicare condizioni di Dirichlet non omogenee, tale generalità non vale per la condizione di Neumann all'outflow, ma l'eventuale estensione è triviale, dato che è sufficiente modificare opportunamente la forma bilineare.

Dalla teoria di HiMod sarà ormai chiaro che nella discretizzazione del dominio si fondono due concezioni molto diverse, da una parte gli Elementi Finiti lungo la fibra di supporto, dall'altra la base modale 2D, che ricorda molto i metodi spettrali. Dunque l'organizzazione delle classi è seguita naturalmente dalle necessità del metodo. Avevamo bisogno inizialmente di una classe che ci permettesse di maneggiare gli elementi dello spazio modale, ovvero la classe **ModalSpace**, ottenuto questo primo risultato è stata creata la classe che mette in comunicazione la fibra di supporto con le slices e risolve il problema, ovvero **HiModAssembler**. Il terzo soggetto principale di questo lavoro è **Basis1DAbstract**, ovvero la classe su cui si appoggia **ModalSpace** per costruire le basi modali basate sulla teoria delle basi educate. Nel prossimo capitolo analizzeremo nel dettaglio queste tre colonne portanti del codice.

Capitolo 3

Descrizione classi

3.1 Modalspace

Inizialmente ModalSpace è stata concepita per essere una classe base, dalla quale ereditasse ogni possibile scelta delle condizioni di bordo sulla parete del vaso. Facendo un rapido conto ci si accorge che comprendendo le condizioni di Dirichlet, Neumann e Robin su una sezione rettangolare arriviamo a 81 possibili combinazioni. Una grande quantità di codice da scrivere, che comprende casistiche molto simili fra loro se non identiche. Questo è stato il primo motivo che ci ha portato a scorporare il trattamento delle condizioni di bordo dalla classe ModalSpace per poi includerlo in modo ottimale in Basis1DAbstract. Un secondo punto a favore di questa scelta riguarda la valutazione e la lettura delle basi modali. È chiaro che le basi modali sono contenute in ModalSpace, tuttavia ogni figlia avrebbe avuto un tipo di base differente e accedervi tramite la classe base ogni qual volta fosse necessario, non risultava essere efficiente. Infine abbiamo ottenuto maggiore generalità, ModalSpace è pronta ad utilizzare nuovi metodi in grado di generare una corretta base modale, infatti la classe possiede esclusivamente le valutazioni delle basi modali su un'opportuna griglia e i generatori di basi.

3.1.1 I membri

ModalSpace conosce la geometria della sezione (L_y , L_z) e sicuramente deve conoscere il grado di precisione desiderato dall'utente, ovvero il numero di modi da utilizzare ($mtot$). Nel caso di basi educate per procedere al passo successivo occorrono le condizioni sulla parete del vaso, tuttavia questa operazione è stata scorportata dal costruttore generico al fine di garantire una certa generalità e seguendo l'ottica della separazione di variabili sono state create due funzioni pubbliche dedicate ad aggiungere le condizioni di bordo:

- AddSliceBCY(const string left, const string right, const Real mu = 1, const Real Chi = 1);

- AddSliceBCZ(const string left, const string right, const Real mu = 1, const Real Chi =1)

Un altro punto fondamentale del costruttore generico è senz'altro la regola di quadratura da utilizzare sulla slice. Si noti che le basi utilizzate necessitano regole di quadratura di alto ordine e il grado di esattezza è strettamente legato al numero di modi. Questo legame è evidente se si pensa che maggiore è il modo, maggiore sarà la frequenza della base modale e di conseguenza si avrà bisogno di una fitta successione di nodi di quadratura. Su una sezione quadrata una buona approssimazione dei nodi necessari su ciascun lato è \sqrt{mtot} . Il risultato non è valido nel caso di sezioni molto asimmetriche, infatti rettangoli molto allungati in una direzione avranno bisogno di più nodi lungo la direzione maggiore e meno sull'altra (si porti qui ad esempio i conti che abbiamo fatto sulle basi). In definitiva non parliamo della regola di quadratura, ma piuttosto permettiamo all'utente di scegliere le regole di quadratura da adottare lungo i lati, sarà compito suo scegliere un grado di esattezza adeguato per il caso di studio.

Modalspace é una classe base da cui derivano le figlie di modalspace che nascono dal momento che tipo di condizioni al bordo di parete vengono scelte.

Perché si é scelto questo tipo di ereditá? Gli autovalori generati e le rispettive basi sono proprietà esclusiva del tipo di problema ai limiti che si intende istanziare. Dunque le classi figlie si preoccupano di calcolare l'appropriata successione di autovalori e inoltre registrano i valori nodali delle funzioni di base sulla griglia di quadratura scelta. I dati vengono raccolti nei membri:

- Eigenvalues
- Mphi
- Mdphi

Fissandoci sulla slice di riferimento $[0, 1] \times [0, 1]$, si osserva che l'unico parametro che occorre ad una figlia per generare il materiale elencato precedentemente la specifica delle condizioni di bordo **anche mtot!!**.

Mi sta venendo un dubbio, il discorso della factory mi aveva intrigato, tuttavia mi stavo chiedendo, ma noi cosa registriamo di preciso? Un oggetto modalspace DDDD a cui però vanno specificati mtot, gli autovalori e i valori delle basi? Dovrei registrare modalspaceDDDD per ogni possibile valore di mtot!!! Forse mi sfugge qualcosa....

Modalspace raccoglie gli elementi e le operazioni comuni a tutti i suoi figli tra questi vi sono sicuramente:

- Dimensioni della slice

- Numero modi utilizzato
- Regola di quadratura
- Calcolo coefficienti Fourier $g(y,z)$ - vettore $[g_1 \ g_2 \dots]$
- Calcolo del coefficiente Fourier di $g(x,y,z)$ - reale $(g(x))$

In realtà Eigenvalues appartiene ancora a Modalspace sarà il caso di sposarlo? Fare attenzione a `findmyzmax()` che definito in `Modalspace.cpp`, lo teniamo perché le funzioni comuni fanno uso di tale membro? Dovremmo costruire il getter nelle classi figlie secondo me sarebbe più coerente

Stabiliti infatti i valori nodali delle funzioni di base e delle loro derivate tutte le basi modali istruite possono essere trattate alla stessa maniera. Sono comuni infatti i seguenti metodi (con j si intende l'indice legato alla soluzione e con k quello legato alla funzione test):

- `ComputePhiPhi` $\int_{\gamma_x} \varphi_j(y, z) \varphi_k(y, x) dydz$
- `ComputeDyPhiPhi` $\int_{\gamma_x} \partial_y \varphi_j(y, z) \varphi_k(y, x) dydz$
- `ComputeDzPhiPhi` $\int_{\gamma_x} \partial_z \varphi_j(y, z) \varphi_k(y, x) dydz$
- `ComputeDyPhiDyPhi` $\int_{\gamma_x} \partial_y \varphi_j(y, z) \partial_y \varphi_k(y, x) dydz$
- `ComputeDzPhiDzPhi` $\int_{\gamma_x} \partial_z \varphi_j(y, z) \partial_z \varphi_k(y, x) dydz$
- `ComputePhi` $\int_{\gamma_x} \varphi_k(y, x) dydz$

L'elemento principale di questa classe è `ConstructModalBasis`. In realtà tale metodo non è altro che un wrappers che facilita l'istanziamento della classe da parte dell'utente. Infatti dietro tale metodo si nasconde una chiamata alla factory dove sono schedati i possibili figli di `ModalSpace`. **Qua possiamo aggiungere il codice di `ConstructModalBasis` così si capisce cosa nascondiamo.**

```
boost::shared_ptr <ModalSpace> ModalSpace::
ConstructModalBasis (    const Real & Ly,
                        const Real & Lz,
                        const UInt & M,
                        const std::string & down,
                        const std::string & left,
                        const std::string & up,
                        const std::string & right )
{
    std::string bc = down + left + up + right;
    std::string id;

    if (bc == "dirdirdirdir")
    {
```

```

        id = "DDDD";
    }
    else if (bc == "robrobrobrob")
    {
        id = "RRRR"
    }
    else
    {
        ERROR_MSG ("This kind of bc has not been
                    implemented yet, please add it!");
    }
}

```

Mi sono convinto che l'approccio ereditarietà é quello migliore, tuttavia dovremmo vedere il reale utilizzo che facciamo di questa factory

3.2 HiModAssembler

Sono momentaneamente separate ma é chiaro che devono appartenere alla stessa classe, ovvero a quella che sintetizzerá insieme Modalspace e FESpace 1D (ovvero quello costruito sulla fibra di supporto).

HiModView e HiModAssembler devono continuamente lavorare con gli elementi di Modalspace e FESpace 1D non sarebbe il caso di instaurare un legame piú intimo? Magari specificando l'amicizia di HiMod con Modalspace e FESpace? Troppo incasinato? Si velocizza il tutto (non occorre infatti passare dai noiosi getters)?

Gli unici membri di HiModAssembler e HiModView sono fespace e modalbasis (in HiModAssembler c'è anche etfespace ma direi che dobbiamo toglierlo e non farlo creare nel main, quella é sicuramente una questione interna di come abbiamo voluto implementare il calcolo dei coefficienti della matrice, inoltre aumentiamo in leggibilità).

Ecco le utilità di HiModAssembler:

- AddADRProblem
- interpolate
- Addrhs (costante e functionType)
- Addrhsfunctor
- AddDirichletBCIn (Momentaneamente via penalizzazione)

Ecco invece le utilità di HiModView:

- funCoeff3D (genera i valori nodali su una griglia partendo dal vettore soluzione o da una funzione)
- normL2 (dato il vettore che sputa fuori funCoeff3D ne fa la normaL2)

- ConvergeFile (crea un file di output gestibile tramite getpot, utilizzo limitato al testconvergence)

3.3 Basis1dAbstract

Le Educated Bases (E.B.) sono una scelta possibile da adottare nella risoluzione di un problema ADR con tecniche di riduzione gerarchica. Il concetto base si fonda sul teorema (citazione di Salsa) e sull'ipotesi di separabilità delle variabili. Le E.B. vengono costruite ad hoc sulle condizioni di vaso del problema in oggetto. L'algoritmo che permette la generazione di queste basi è riassunto qui di seguito:

1. **Costruzione di un problema ausiliario** che rispecchi la natura delle condizioni alle pareti del problema originale (devono essere omogenee) e passaggio ai relativi problemi agli autovalori.

Esempio - RRRR

Nel caso si abbiano condizioni di robin uguali sull'intera parete del vaso, dovremo considerare il seguente problema ausiliario:

$$\begin{cases} -\Delta u(y, z) = 0 & \text{in } \gamma_x \\ \mu \nabla u(y, z) \cdot \mathbf{n}_{\gamma_x} + \chi u(y, z) = 0 & \text{su } \Gamma_{vaso} \end{cases} \quad (3.1)$$

Si passi ora al problema agli autovalori associato al precedente sistema e ipotizzando la separazione di variabili per $u(y, z) = \varphi(y)\vartheta(z)$, si arrivano facilmente ad ottenere i seguenti sottoproblemi agli autovalori:

$$\begin{cases} -\varphi(y)'' = K_y \varphi(y) \\ \mu \varphi(y)' + \chi \varphi(y) = 0 & \text{per } y = L_y \\ -\mu \varphi(y)' + \chi \varphi(y) = 0 & \text{per } y = 0 \end{cases} \quad (3.2)$$

$$\begin{cases} -\vartheta(z)'' = K_z \vartheta(z) \\ \mu \vartheta(z)' + \chi \vartheta(z) = 0 & \text{per } z = L_z \\ -\mu \vartheta(z)' + \chi \vartheta(z) = 0 & \text{per } z = 0 \end{cases} \quad (3.3)$$

2. **Identificazione del tipo di soluzione** dei problemi agli autovalori associati.

Esempio - RRRR

Per i sottoproblemi ottenuti i generi di soluzione sono i seguenti:

$$\begin{aligned} \varphi(y) &= A \sin(\sqrt{K_y} y) + B \cos(\sqrt{K_y} y) \\ \vartheta(z) &= A \sin(\sqrt{K_z} z) + B \cos(\sqrt{K_z} z) \end{aligned} \quad (3.4)$$

3. Ricerca degli autovalori di un sottoproblema tramite risoluzione dell'equazione non lineare associata ad esso, ottenuta risolvendo le condizioni di bordo.

Esempio - RRRR

Nel caso trattato in esempio le equazioni che si ottengono sono le seguenti ($x = \sqrt{K_y}$ e $w = \sqrt{K_z}$):

$$\begin{aligned} f(x) &= 2\mu x + \tan(L_y x) \left(\chi - \frac{\mu^2 x^2}{\chi} \right) \\ f(w) &= 2\mu w + \tan(L_z w) \left(\chi - \frac{\mu^2 w^2}{\chi} \right) \end{aligned} \quad (3.5)$$

Come presentato in breve nel capitolo (Nozioni base), HiMod si basa sulla riduzione dei gradi di libertà del sistema discreto, questo risultato si ottiene ordinando le basi scelte in maniera gerarchica ed utilizzando quelle legate alle frequenze più basse. Dunque il primo passo è capire, dato un numero di modi massimo che si vuole dedicare per la componente trasversale, qual'è il set di basi modali da utilizzare.

Osservazione

Nel caso di condizioni al bordo di Dirichlet il problema si semplifica. Infatti non occorre adottare l'algoritmo mostrato precedentemente, gli autovalori che si ottengono sono noti a priori e sono della forma:

$$\begin{aligned} K_y &= \left(\frac{\pi p}{L_y} \right)^2 & p &= 1, \dots, m_y \\ K_z &= \left(\frac{\pi q}{L_z} \right)^2 & q &= 1, \dots, m_z \end{aligned} \quad (3.6)$$

Dunque è nota la relazione $\lambda(K_y, K_z)$ a priori e risulta molto semplice ordinare in modo crescente gli autovalori definendo quindi m_y e m_z .

Possiamo discutere velocemente l'albero delle scelte (che poi sarebbe la funzione EigensProvider), che in realtà sarà utilizzato tale e quale per il caso generale delle educated basis, infatti se adesso il passaggio è semplicemente $p = p + 1$, dopo il passaggio sarà $p = \text{Next}()$. La ricerca degli autovalori rallenterà parecchio dato che si dovrà risolvere un problema non lineare ogni volta che si vuole aggiornare un nodo.

Esponiamo quindi lo pseudocodice che, dato il numero di modi da utilizzare, determina l'elenco delle frequenze e sotto frequenze:

```
//Dichiarazione set di ritorno
M_Eigenvalues;
//Suggerisco di farlo un pair di pair visto che non possiamo
    fare la tupla, la prima pair sono i sotto autovalori,
    la seconda pair sono le coordinate (ovver p e q)
```

```

//Dichiarazione set degli scarti
scarti;

//Inserimento delle prime sottofrequenze e prima coordinata
(1,1)
KzOld = M_EducatedBasisy->Next();
kyOld = M_educated->BasiszNext();
M_eigenvalues.insert(make_pair(KyOld,KzOld) );

//Inizializzazione set degli scarti
scarti.insert(make_pair(M_EducatedBasisy->Next(),KzOld));
scarti.insert(make_pair(KyOld,M_EducatedBasisz->Next()));

//Ciclo sul numero di modi scelto
for(UInt i=1;i\M_mtot;++i)
{
    //Prendo il primo elemento del vettore degli scarti,
    che sono ordinati per lambda=Ky+Kz dal pi
    piccolo al pi grande e lo inserisco nel set di
    uscita
    M_eigenvalues.insert(scarti.begin());

//Inoltre registro le coordinate del nodo preso e i valori
dei sottoautovalori
i = prima coordinata;
j = seconda coordinata;
KzOld =
KyOld =

//Mi occupo ora di branchare in maniera corretta e senza
ricalcolare autovalori che gi conosco
if(!Check_presence(i+1,j))
    scarti.insert(Check_eigen(1,i+1),KzOld);

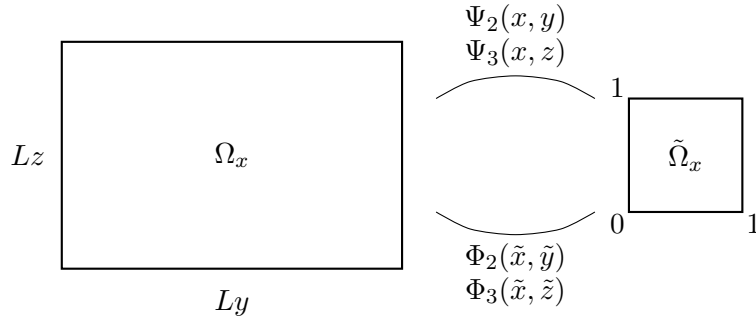
if(!Check_presence(i,j+1))
    scarti.insert(KyOld,Check_eigen(2,j+1));
}

```


Capitolo 4

Conti 3D

Si consideri la generica mappa che passa dalla slice reale a quella di riferimento.



Consideriamo una mappa della seguente forma:

$$\Psi(x, y, z) := \begin{cases} \Psi_1 = x; \\ \Psi_2 = \Psi_2(x, y); \\ \Psi_3 = \Psi_3(x, z); \end{cases} \quad (4.1)$$

Lo Jacobiano prodotto sarà della forma seguente (si noti che $x = \tilde{x}$):

$$J_{\Psi}(x, y, z) := \begin{bmatrix} 1 & 0 & 0 \\ \partial_x \Psi_2(x, y) & \partial_y \Psi_2(x, y) & 0 \\ \partial_x \Psi_3(x, z) & 0 & \partial_z \Psi_3(x, z) \end{bmatrix} \quad (4.2)$$

Lo spazio ridotto viene costruito sul dominio di riferimento, ovvero il parallelepipedo di sezione $(0, 1) \times (0, 1)$ e lunghezza pari alla lunghezza reale L_x .

$$V_m := \left\{ v_m(x, y, z) = \sum_{k=1}^m v_k(\Psi_1(x)) \varphi_k(\Psi_2(x, y), \Psi_3(x, z)) : v_k \in V_{\tilde{\Omega}_{1D}} \right\} \quad (4.3)$$

Le $\varphi_k(\Psi_2(x, y), \Psi_3(x, z))$ sono invece le basi modali costruite sulla slice di riferimento. Ricordiamo la formulazione debole del problema in analisi:

$$\int_{\Omega} (\mu \nabla u_m \nabla v_m + \mathbf{b} \nabla u_m v_m + \sigma u_m v_m) d\Omega = \int_{\Omega} f v dx dy \quad (4.4)$$

Consideriamo la seguente espressione per u_m , dove si è definito $\Psi_2(\mathbf{x}) := (\Psi_2(x, y), \Psi_3(x, z))$,

$$u_m(x, y, z) := \sum_{j=1}^m u_j(x) \varphi_j(\Psi_2(\mathbf{x})) \quad (4.5)$$

Procediamo con i calcoli sapendo che:

$$\nabla(u_j(x) \varphi_j(\Psi_2(\mathbf{x}))) = \begin{bmatrix} u'_j(x) \varphi_j(\Psi_2(\mathbf{x})) + u_j(x) \tilde{\nabla} \varphi_j \cdot \partial_x \Psi_2 \\ u_j(x) \tilde{\nabla} \varphi_j \cdot \partial_y \Psi_2 \\ u_j(x) \tilde{\nabla} \varphi_j \cdot \partial_z \Psi_2 \end{bmatrix} \quad (4.6)$$

$$\tilde{\nabla} \varphi_j := \begin{bmatrix} \partial_y \varphi_j \\ \partial_z \varphi_j \end{bmatrix} \quad \partial_i \Psi_2 := \begin{bmatrix} \partial_i \Psi_2 \\ \partial_i \Psi_3 \end{bmatrix} \quad (4.7)$$

Riscriviamo quindi la formulazione come segue:

$$\begin{aligned} & \int_{\Omega} \mu(\mathbf{x}) (\\ & u'_j(x) \vartheta'(x) \varphi_j(\Psi_2(\mathbf{x})) \varphi_k(\Psi_2) \\ & + u'_j(x) \vartheta(x) \varphi_j(\Psi_2(\mathbf{x})) [\tilde{\nabla} \varphi_k(\Psi_2) \cdot \partial_x \Psi_2] \\ & + u_j(x) \vartheta'(x) [\tilde{\nabla} \varphi_j(\Psi_2(\mathbf{x})) \cdot \partial_x \Psi_2] \varphi_k(\Psi_2) \\ & + u_j(x) \vartheta(x) [\tilde{\nabla} \varphi_j(\Psi_2(\mathbf{x})) \cdot \partial_x \Psi_2] [\tilde{\nabla} \varphi_k(\Psi_2) \cdot \partial_x \Psi_2] \\ & + u_j(x) \vartheta(x) [\tilde{\nabla} \varphi_j(\Psi_2(\mathbf{x})) \cdot \partial_y \Psi_2] [\tilde{\nabla} \varphi_k(\Psi_2) \cdot \partial_y \Psi_2] \\ & + u_j(x) \vartheta(x) [\tilde{\nabla} \varphi_j(\Psi_2(\mathbf{x})) \cdot \partial_z \Psi_2] [\tilde{\nabla} \varphi_k(\Psi_2) \cdot \partial_z \Psi_2] \\ &) d\Omega \\ & + \int_{\Omega} \\ & b_1(\mathbf{x}) \vartheta(x) \varphi_k(\Psi_2) (u'_j(x) \varphi_j(\Psi_2) + u_j(x) [\tilde{\nabla} \varphi_j(\Psi_2(\mathbf{x})) \cdot \partial_x \Psi_2]) \\ & + b_2(\mathbf{x}) \vartheta(x) \varphi_k(\Psi_2) u_j(x) [\tilde{\nabla} \varphi_j(\Psi_2(\mathbf{x})) \cdot \partial_y \Psi_2] \\ & + b_3(\mathbf{x}) \vartheta(x) \varphi_k(\Psi_2) u_j(x) [\tilde{\nabla} \varphi_j(\Psi_2(\mathbf{x})) \cdot \partial_z \Psi_2] \\ & d\Omega \\ & + \int_{\Omega} \sigma(\mathbf{x}) u_j(x) \vartheta(x) \varphi_j(\Psi_2) \varphi_k(\Psi_2) d\Omega \\ & = \int_{\Omega} f(\mathbf{x}) \vartheta(x) \varphi_k(\Psi_2) d\Omega \end{aligned} \quad (4.8)$$