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

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

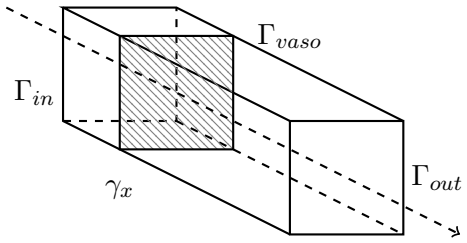
Introduzione

2.1 Nozioni base

L'obbiettivo 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:

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

$$\Omega = \bigcup_{x\in\Omega_{1D}} \gamma_x \tag{2.2}$$



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 $V_{1D} = H^1_{\Gamma_{in}}(\Omega_{1D})$, 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 $V_{\gamma_x} := span\{\varphi_k\}$. Definiamo ora il sottospazio generato solo dai primi m modi ovvero $V_{\gamma_x}^m := span\{\varphi_1, ..., \varphi_m\}$ e combiniamolo con V_{1D}, ottenendo il seguente spazio ridotto:

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

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

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

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;
- $\lim\limits_{x \to +\infty} \left(\inf\limits_{v_m \in V_m} || \, v - v_m \, || \right) = 0$ 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 $k = 1, ...m$. Il problema assume la seguente forma:

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

Svolgendo l'operatore gradiente si ottiene:

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

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

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

Nella successiva implementazione del metodo si è considerato per semplicità, una base F.E.M. di primo grado. Possiamo quindi esprimere i coefficienti di Fuorier 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 $l = 1, ...N$ e $k = 1, ...m$ si ha che:

$$\begin{aligned}\sum_{j=1}^{m}\sum_{s=1}^{N} u_{js}\Bigg[&\int_\Omega \mu(\partial_x\psi_s\partial_x\psi_l\varphi_j\varphi_k + \psi_s\psi_l\partial_y\varphi_j\partial_y\varphi_k + \psi_s\psi_l\partial_z\varphi_j\partial_z\varphi_k)\,dxdydz \\ &+ \int_\Omega (b_1\partial_x\psi_s\varphi_j + b_2\psi_s\partial_y\varphi_j + b_3\psi_s\partial_z\varphi_j)\psi_l\varphi_k\,dxdydz \\ &+ \int_\Omega \sigma\psi_s\psi_l\varphi_j\varphi_k\,dxdydz\Bigg] \\ &= \int_\Omega f\psi_l\varphi_k\,dxdydz \quad (2.7)\end{aligned}$$

Si osservi che il doppio indice $''js''$, in realtà scorre un vettore, la rimappatura in un solo indice può facilmente essere dedotta ottenenedo che $[\mathbf{u}]_{js} = \mathbf{u}[(j-1)N + s]$. La matrice generata ha quindi dimensioni $(mN)^2$, tuttavia fissata la frequenza delle soluzione 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:

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

2.3 Implementazione integrali

Nel caso i coefficienti del problema ADR siano dipendenti dalla sola coordinata x o risultino fattorizabili 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 opportunatamente 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

$$\begin{array}{ll} \partial_x\psi_s\partial_x\psi_l & \int_{\gamma_x} \mu\varphi_j\varphi_k \, dydz \\ \partial_x\psi_s\psi_l & \int_{\gamma_x} \varphi_j\varphi_k \, dydz \\ \psi_s\psi_l & \int_{\gamma_x} (\mu\partial_y\varphi_j\partial_y\varphi_k + \mu\partial_z\varphi_j\partial_z\varphi_k + b_2\partial_y\varphi_j\varphi_k + b_3\partial_z\varphi_j\varphi_k + \sigma\varphi_j\varphi_k) \, dydz \end{array} \tag{2.9}$$

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.
- È possibile separare il problema lungo le direzioni trasversali, in due sotto problemi agli autovalori.

La forma del dominio considerato ci consente agilmente di applicare le tecninche 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 Neuamann 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 necessitàdel 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 quantià 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 (Ly, Lz) e sicuramente deve conoscere il grado di precisione desiderato dall'utente, ovvero il numero di modi da utilizzare (mtot). 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. Come esempio si osservi la seguente tabella dove sono riportati i check dei valori di normalità di una base, fissata la regola di quadratura al variare della dimensione L_y.

TABELLA (3.1)

Una volta creato l'oggetto `ModalSpace` bisogna eseguire alcuni set importanti. Per prima cosa dobbiamo impostare i generatori di base lungo le direzioni trasversali. Nel caso di basi educate questa operazione viene eseguita assieme all'imposizione delle condizioni di parete tramite i metodi pubblici:

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

```
void ModalSpace::
AddSliceBCY (const string& left, const string& right, const
    Real& mu, const Real& chi)
{
  M_genbasisY = Basis1DFactory::istance().createObject(left+
    right;)
  M_genbasisY->setL(M_Ly);
  M_genbasisY->setMu(M_mu);
  M_genbasisY->setChi(chi)

  return;
}
```

Nel caso di future generalizzazioni o aggiunte di nuove basi sarà in questo punto che occorrerà procedere. Qua aprire il discorso della factory?. In pratica si occupano di assegnare il giusto generatore ai membri `M_genbasisY` e `M_genbasisZ`. Nel nostro codice questi membri sono dei puntatori ad oggetti di tipo `Basis1DAbstract`, vedremo nel dettaglio la loro implementazione nella prossima sezione. Infine si conlude il seting della classe ModalSpace tramite la funzione membro pubblica EvaluateBasis(), che chiama le funzioni che si occupano di riempire le strutture dati.

```
boost::shared_ptr<ModalSpace> MB (new ModalSpace(Ly,Lz,mtot,
    quadY,quadZ));
MB->AddSliceBCY("dir","dir");
MB->AddSliceBCZ("rob","rob",1.,3.);
MB->EvaluateBasis();
```

Strutture dati

Diamo un breve descrizione delle strutture dati possedute dalla classe ModalSpace. Per prima cosa però, occupiamoci aspetto fondamentale: le basi modali sono determinate sull'intervallo di riferimento. Per non incorrere in errori fra dominio reale e riferimento, utilizzeremo la seguente notazione:

$$\begin{aligned} &\hat{\varphi}_j(\hat{y},\hat{z}) = \hat{\eta}_j(\hat{y})\hat{\xi}_j(\hat{z}) \quad \hat{y}\in[0,1] \qquad \hat{z}\in[0,1] \\ &\int_0^1 \hat{\eta}_j^2\, d\hat{y} = 1 \qquad\qquad \int_0^1 \hat{\xi}_j^2\, d\hat{z} = 1 \end{aligned} \tag{3.2}$$

Dove $\hat{\varphi}_j$ è la base modale ortonormale sul dominio di riferimento, risultato del prodotto delle basi ottenute tramite i generatori di basi. Vediamo ora come gestire il passaggio dalle basi definite sul riferimento a quelle invece sul dominio reale. L'ortogonalità si conserva facilmente, ma lo stesso discorso non vale per la normalizzazione. Verifichiamo che un semplice cambio di coordinate non conserva la normalizzazione:

$$\begin{aligned} &\int_0^{L_y}\int_0^{L_z} \varphi_j(y,z)^2\, dydz \\ &= \int_0^{L_y} \eta_j(y)^2\, dy \int_0^{L_z} \xi_j(z)^2\, dz \\ &= \int_0^1 \eta_j(L_y\hat{y})^2 L_y\, d\hat{y} \int_0^1 \xi_j(L_z\hat{z})^2 L_z\, d\hat{z} \\ &= L_yL_z \int_0^1 \hat{\eta}_j(\hat{y})^2\, d\hat{y} \int_0^1 \hat{\xi}_j(\hat{z})^2\, d\hat{z} \qquad \neq 1 \end{aligned} \tag{3.3}$$

Da questi semplici passaggi deduciamo che per essere mantenere la normalizzazione, la base che stiamo cercando avrà la seguente forma:

$$\varphi_j(y,z) = (L_yL_z)^{-\frac{1}{2}}\hat{\eta}_j(yL_y^{-1})\hat{\xi}_j(zL_z^{-1}) \tag{3.4}$$

In conclusione, nei conti che verranno proposti si utilizzerà sempre questa forma della base modale.

Procediamo con la descrizione delle strutture dati:

- `EigenContainer M_eigenvalues`, contiene le sottofrequenze e gli indici corrispondenti, viene prodotta in fase di setting dello spazio modale tramite la funzione membro `EigensProvider()`, chiamata da `EvaluateBasis()`. Il tipo è un `vector<EigenMap>` dove:

```
struct EigenMap
{
  Real wp;   //subfrequency y
  Real wq;   //subfrequency z
  UInt p;
  UInt q;
```

```
  static EigenMap make_eigenmap(const Real& _wp,const Real&
     _wq,const UInt& _p,const UInt& _q)
  {
    EigenMap a;
    a.wp = _wp;
    a.wq = _wq;
    a.p = _p;
    a.q = _q;
    return a;
  }
};
```

L'ordinamento gerarchico degli autovalori e la corrispondenza delle sottofrequenze con i sottoindici è fondamentale, approfondiremo in seguito il metodo `EigensProvider()`.

- **MBMatrix_type M_phiy**, è un `vector<vector<Real> >` che raccoglie la valutazione di $\hat{\eta}_j(\hat{y})\ \forall j$ e per ogni nodo di quadratura lungo $\hat{y} \in [0,1]$.
- **MBMatrix_type M_phiz**, è un `vector<vector<Real> >` che raccoglie la valutazione di $\hat{\xi}_j(\hat{y})\ \forall j$ e per ogni nodo di quadratura lungo $\hat{z} \in [0,1]$.
- **MBMatrix_type M_dphiy**, è un `vector<vector<Real> >` che raccoglie la valutazione di $\frac{\partial \hat{\eta}_j}{\partial \hat{y}}\ \forall j$ e per ogni nodo di quadratura lungo $\hat{y} \in [0,1]$.
- **MBMatrix_type M_dphiz**, è un `vector<vector<Real> >` che raccoglie la valutazione di $\frac{\partial \hat{\xi}_j}{\partial \hat{z}}\ \forall j$ e per ogni nodo di quadratura lungo $\hat{z} \in [0,1]$.

Metodi di calcolo

Approfondiamo ora i metodi che si occupano di calcolare i coefficienti della matrice di sistema.

- `Real Compute_PhiPhi(const UInt& j, const UInt& k)`
  $\int_{\gamma_x} \varphi_j(y,z)\varphi_k(y,x)\,dydz$
- `Real Compute_DyPhiPhi(const UInt& j,const UInt& k)`
  $\int_{\gamma_x} \partial_y\varphi_j(y,z)\varphi_k(y,x)\,dydz$
- `Real Compute_DzPhiPhi(const UInt& j,const UInt& k)`
  $\int_{\gamma_x} \partial_z\varphi_j(y,z)\varphi_k(y,x)\,dydz$

- `Real Compute_DyPhiDyPhi(const UInt& j,const UInt& k)`

  $\int_{\gamma_x} \partial_y \varphi_j(y,z) \partial_y \varphi_k(y,x)\, dydz$

- `Real Compute_DzPhiDzPhi(const UInt& j,const UInt& k)`

  $\int_{\gamma_x} \partial_z \varphi_j(y,z) \partial_z \varphi_k(y,x)\, dydz$

- `Real Compute_Phi(const UInt& k)`

  $\int_{\gamma_x} \varphi_k(y,x)\, dydz$

- `vector<Real> FourierCoefficients (const function_Type& g) const`,

  data una funzione indipendente da x questo metodo restituisce i coefficienti di Fourier (in numero pari ad `M_mtot`) rispetto alla base modale scelta.

- `Real Coeff_fk ( const Real& x,const function_Type& f,const UInt& k ) const`,

  restituisce il k-esimo coefficiente di Fourier di una generica funzione 3D valutato nel punto x, rispetto alla base modale.

Date le premesse risulta ora semplice risolvere gli integrali scritti qui sopra, vediamo ad esempio che aspetto ha `Compute_PhiPhi()`:

```
Real ModalSpace::
Compute_PhiPhi(const UInt& j, const UInt& k) const
{
  Real coeff_y = 0.0;
  Real coeff_z = 0.0;
  UInt p_j = M_eigenvalues[j].p-1;
  UInt p_k = M_eigenvalues[k].p-1;
  UInt q_j = M_eigenvalues[j].q-1;
  UInt q_k = M_eigenvalues[k].q-1;

  Real normy = 1.0 / sqrt(M_Ly);
  Real normz = 1.0 / sqrt(M_Lz);

  for(UInt n = 0; n < M_quadruleY->nbQuadPt();++n)
  {
    coeff_y +=  M_phiy[p_j][n] * normy *
                M_phiy[p_k][n] * normy *
                M_Ly * M_quadruleY->weight(n);
  }

    for(UInt n = 0; n < M_quadruleZ->nbQuadPt();++n)
  {
    coeff_z +=  M_phiz[q_j][n] * normz *
                M_phiz[q_k][n] * normz *
                M_Lz * M_quadruleZ->weight(n);
  }

  return coeff_y*coeff_z;
```

```
}
```

Gli ultimi due metodi citati sono indispendabili ed il loro impiego sarà noto una volta che tratteremo la classe `HiModAssembler`.

`EigensProvider()`

Abbiamo deciso di dedicare una sezione solamente a questo metodo, poiché la ricerca degli autovalori occupa un ruolo fondamentale nella struttura del codice. Il metodo viene chiamato da `EvaluateBasis()` dunque dopo che sono stati settati i generatori di basi. La funzione deve preoccuparsi di definire la struttura dati `M_eigenvalues`, tuttavia il procedimento non è scontato. Per comprendere le difficoltà occorre ragionare sulla struttura del problema. Separando le variabili della slice 2D si sono ottenuti due problemi agli autovalori 1D. Ognuno di questi genera una successione ordinata crescente di autovalori, determinata dalla ricerca degli zeri di una data funzione. Definiamo la successione di autovalori in y con $\{K_y\}_p$ e quella in z con $\{K_z\}_q$. Le precedenti successioni definiscono univocamente la successione degli autovalori del problema di partenza 2D e sono in relazione con essa nel seguente modo:

$$\lambda_j = (K_y^p)^2 + (K_z^q)^2 \tag{3.5}$$

Anche $\{\lambda\}_j$ è una successione crescente di autovalori, ma il suo ordinamento, dato quello dei sottoautovalori, non è immediato. Due sono le difficoltà che si presentano:

1. Ogni sottoautovalore è il risultato di una ricerca di zeri di una funzione non lineare.

2. L'utente stabilisce il numero massimo di modi sul problema 2D e non sui sotto-problemi 1D.

Le due problematiche sono strettamente legate, difatti non siamo interessati a cercare più sottoautovalori del necessario. Si poteva partire calcolando ad esempio 10 sottoautovalori in y e altrettanti in z, ordinare la succession 2D e procedere eventualmente nella ricerca. Questo metodo tuttavia presenta due difetti: è poco efficiente ed inoltre può cadere in errore. Infatti l'algoritmo si dovrebbe fermare una volta raggiunti un numero di autovalori 2D pari ad `M_mtot`, ma così facendo nessuno ci assicura che nel gruppo successivo di 10 autovalori non vi sia almeno uno minore dell'ultimo autovalore calcolato.

La soluzione è stata quella di procedere un passo alla volta, con l'accortezza di salvare i sotto-autovalori ancora non utlizzati. Per fare questo il metodo di ricerca degli zeri (`Next()` che approfondiremo nella sezione `Basis1DAbstract`) fornisce progressivamente uno zero alla volta. Infine abbiamo analizzato il seguente albero delle scelte:

$$ALBERODELLESCELTE \tag{3.6}$$

3.2 Basis1dAbstract

Questa classe si occupa di definiere dei generatori di base seguendo la teoria delle Educated Basis (E.B.). Introduciamo con degli esempi l'algoritmo di ricerca delle basi ortonormali, senza entrare nel dettaglio della teoria (per riferimenti si guardi ....):

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:

$$\begin{cases} -\Delta u(y,z) = 0 & \text{in } \gamma_x \\ \mu \nabla u(y,z) \cdot \mathbf{n}_{\gamma_x} + \chi u(y,z) = 0 & \text{su } \Gamma_{vaso} \end{cases} \tag{3.7}$$

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:

$$\begin{cases} -\varphi(y)'' = K_y \varphi(y) & \\ \mu\varphi(y)' + \chi\varphi(y) = 0 & \text{per } y = L_y \\ -\mu\varphi(y)' + \chi\varphi(y) = 0 & \text{per } y = 0 \end{cases} \tag{3.8}$$

$$\begin{cases} -\vartheta(z)'' = K_z \vartheta(z) & \\ \mu\vartheta(z)' + \chi\vartheta(z) = 0 & \text{per } z = L_z \\ -\mu\vartheta(z)' + \chi\vartheta(z) = 0 & \text{per } z = 0 \end{cases} \tag{3.9}$$

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

Esempio - RRRR

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

$$\begin{aligned} \varphi(y) &= Asin(\sqrt{K_y}y) + Bcos(\sqrt{K_y}y) \\ \vartheta(z) &= Asin(\sqrt{K_z}z) + Bcos(\sqrt{K_z}z) \end{aligned} \tag{3.10}$$

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 $w = \sqrt{K_z}$):

$$\begin{aligned} f(x) &= 2\mu x + tan(L_y x)\left(\chi - \tfrac{\mu^2 x^2}{\chi}\right) \\ f(w) &= 2\mu w + tan(L_z w)\left(\chi - \tfrac{\mu^2 w^2}{\chi}\right) \end{aligned} \tag{3.11}$$

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:

$$\begin{aligned} K_y &= \left(\tfrac{\pi p}{L_y}\right)^2 \quad p = 1, ..., m_y \\ K_z &= \left(\tfrac{\pi q}{L_z}\right)^2 \quad q = 1, ..., m_z \end{aligned} \tag{3.12}$$

Dunque é 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.

3.3 HiModAssembler

La classe HiModAssembler gestisce lo spazio ridotto presentato nella teoria e si occupa di assemblare la matrice a blocchi del sistema. Analizziamo i membri di cui è composta:

- `modalbasis_ptrType M_modalbasis`
- `fespace_ptrType M_fespace`
- `etfespace_ptrType M_etfespace`

Chiaramente lo spazio ridotto è composto da uno spazio modale e da uno elementi finiti. Il terzo oggetto

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'é 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 leggibilitá).

Ecco le uilitá di HiModAssembler:

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

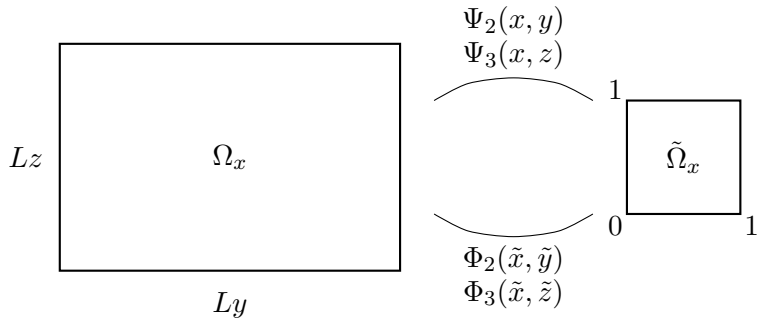
Ecco invece le utilitá di HiModView:

- 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)

Capitolo 4

Conti 3D

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



Consideriamo una mappa della seguente forma:

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

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

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

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.

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

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:

$$\int_\Omega \left(\mu \nabla u_m \nabla v_m + \mathbf{b}\nabla u_m v_m + \sigma u_m v_m\right)\, d\Omega = \int_\Omega f v\, dxdy \tag{4.4}$$

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

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

Procediamo con i calcoli sapendo che:

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

$$\tilde{\nabla}\varphi_j := \begin{bmatrix} \partial_{\tilde{y}}\varphi_j \\ \partial_{\tilde{z}}\varphi_j \end{bmatrix} \quad \partial_i \mathbf{\Psi_2} := \begin{bmatrix} \partial_i \Psi_2 \\ \partial_i \Psi_3 \end{bmatrix} \tag{4.7}$$

Riscriviamo quindi la formulazione come segue:

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