

# Riduzione Gerarchica di Modello in 3D

con le basi istruite

Matteo Aletti & Andrea Bortolossi

Politecnico di Milano

16 Ottobre 2013



## 1 Fondamenti teorici

- Hierarchical Model Reduction in 3D
- Basi istruite

## 2 Implementazione

- Basis1DAbstract
- ModalSpace
- HiModAssembler

## 3 Risultati numerici

## 1 Fondamenti teorici

- Hierarchical Model Reduction in 3D
- Basi istruite

## 2 Implementazione

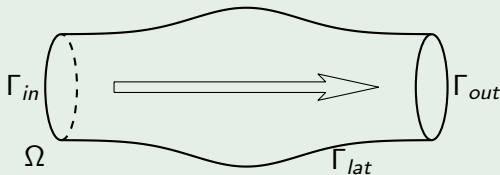
- Basis1DAbstract
- ModalSpace
- HiModAssembler

## 3 Risultati numerici

# Motivazione

esistenza di una direzione dominante

Vogliamo risolvere un certo tipo di problemi: quelli che presentano una direzione preferenziale



## Modello 3D

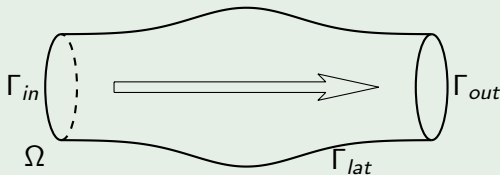
Alta precisione 😊

Costoso 😞

# Motivazione

esistenza di una direzione dominante

Vogliamo risolvere un certo tipo di problemi: quelli che presentano una direzione preferenziale



## Modello 1D

Bassa precisione ☹️

Economico 😊

## Modello 3D

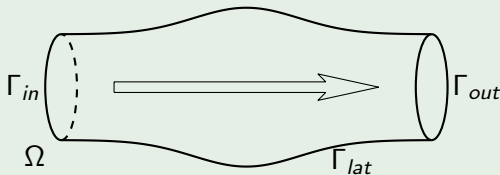
Alta precisione 😊

Costoso ☹️

# Motivazione

esistenza di una direzione dominante

Vogliamo risolvere un certo tipo di problemi: quelli che presentano una direzione preferenziale



## Modello 1D

Bassa precisione ☹️

Economico 😊

HiMod

## Modello 3D

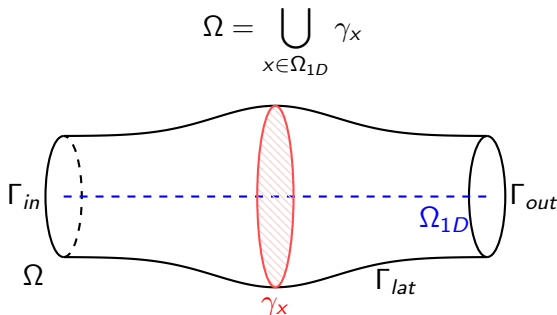
Alta precisione 😊

Costoso ☹️

# Impostazione geometrica

il dominio

- Fibra di supporto rettilinea  $\Omega_{1D}$  dove avviene la dinamica dominante.
- Suddivisione del dominio in slices  $\gamma_x$  ortogonali alla fibra di supporto.



- mappare  $\Omega$  in un dominio di riferimento  $\hat{\Omega}$  in modo che

$$\gamma_x \mapsto \hat{\gamma}_{\hat{x}} = \hat{\gamma} \quad \forall \hat{x} \in \hat{\Omega}_{1D} \implies \hat{\Omega} = \hat{\Omega}_{1D} \times \hat{\gamma}$$

- espandere, in direzione trasversale, la soluzione rispetto alla base di Fourier generalizzata

$$\{\hat{\varphi}_k(\hat{y}, \hat{z})\}_{k \in \mathbb{N}}$$

**Notazione:** lavoreremo già sul riferimento, non useremo quindi i cappelli.

### Spazi in direzione trasversale

$$V_{\gamma}^{\infty} = \left\{ v(y, z) = \sum_{k=1}^{\infty} v_k \varphi_k(y, z) \right\}$$
$$V_{\gamma}^m = \left\{ v(y, z) = \sum_{k=1}^m v_k \varphi_k(y, z) \right\}$$



# Processo di riduzione

## Spazi ridotti

Usiamo uno spazio  $V_{1D}$  di tipo  $H^1(\Omega_{1D})$  lungo la **direzione principale**. Possiamo ora definire gli spazi ridotti come **spazi prodotto**

### Spazi ridotti

$$V^\infty(\Omega) = V_{1D} \otimes V_\gamma^\infty := \left\{ v(x, y, z) = \sum_{k=1}^{\infty} v_k(x) \varphi_k(y, z), v_k \in V_{1D} \right\}.$$

$$V^m(\Omega) = V_{1D} \otimes V_\gamma^m := \left\{ v(x, y, z) = \sum_{k=1}^m v_k(x) \varphi_k(y, z), v_k \in V_{1D} \right\}.$$

È lo spazio delle **combinazioni lineari a coefficienti in  $V_{1D}$**  delle funzioni di base della fibra trasversale.

# Il problema

Caso condizioni di Dirichlet

Il **problema** che vogliamo risolvere ...

$$\begin{cases} -\mu \Delta u + \beta \cdot \nabla u + \sigma u = f & \text{in } \Omega \\ u = 0 & \text{su } \partial\Omega \end{cases}$$

... e la sua **formulazione debole**

Trovare  $u \in H_0^1(\Omega)$  tale che

$$\int_{\Omega} \mu \nabla u \nabla v + \beta \cdot \nabla uv + \sigma uv d\Omega = \int_{\Omega} f v d\Omega \quad \forall v \in H_0^1(\Omega).$$

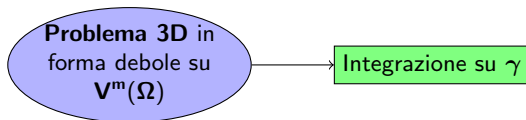
# Modelli ridotti

Problemi 1D accoppiati

**Problema 3D** in  
forma debole su  
 $\mathbf{V}^m(\Omega)$

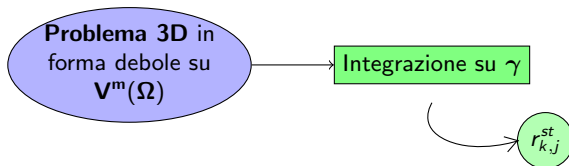
# Modelli ridotti

Problemi 1D accoppiati



# Modelli ridotti

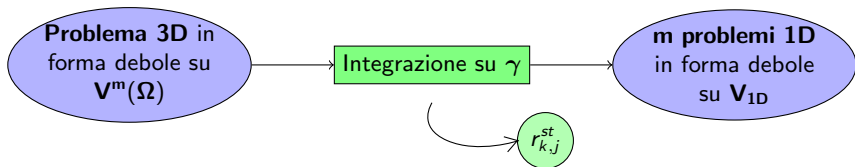
## Problemi 1D accoppiati



I coefficienti  $r_{k,j}^{st}$  accoppiano i problemi 1D: comprimono le informazioni nella fibra trasversale tramite opportuni integrali su  $\gamma$ .

# Modelli ridotti

## Problemi 1D accoppiati



I coefficienti  $r_{k,j}^{st}$  accoppiano i problemi 1D: comprimono le informazioni nella fibra trasversale tramite opportuni integrali su  $\gamma$ .

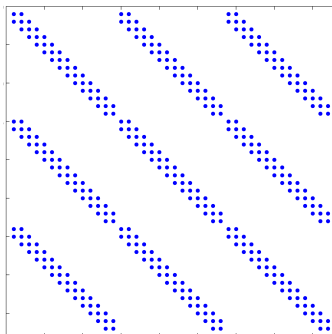
Trovare  $\{u_k\}_{k=1}^m$  con  $u_k \in V_{1D} \forall k = 1 \dots m$  tale che

$$\sum_{k=1}^m \int_{\Omega_{1D}} \left[ \underbrace{\hat{r}_{k,j}^{11} \frac{\partial u_k}{\partial x} \frac{\partial \theta_j}{\partial x}}_{\text{Diffusione}} + \underbrace{\hat{r}_{k,j}^{10} \frac{\partial u_k}{\partial x} \theta_j}_{\text{Trasporto}} + \underbrace{\hat{r}_{k,j}^{00} u_k \theta_j}_{\text{Reazione}} dx \right] = \int_{\Omega_{1D}} \underbrace{\theta_j f_k}_{\text{Forzante}} dx. \quad \forall j = 1 \dots m \quad \theta_j \in V_{1D}$$

# Struttura algebrica

## Pattern di sparsità

Per  $V_{1D}$  utilizziamo una discretizzazione elementi finiti P1 ottenendo il seguente **pattern di sparsità**.



14 elementi, 3 modi.

**Nota:** i blocchi si riferiscono alle **frequenze**, ogni blocco ha il pattern proprio degli **elementi finiti**

# Scelta della base

## Basi trigonometriche e polinomi di Legendre

In **letteratura** è stato considerato il problema di **Dirichlet** in **2D**. Le basi scelte sono state

- Funzioni **trigonometriche**:  $\sin(\pi kx)$  in  $(0, 1)$  [Prof. Perotto]
- Polinomi di **Legendre** moltiplicati per un fattore  $(1 - x^2)$  e ortogonalizzati alla Gram-Schmidt [Prof. Blanco, LNCC]

### In questo progetto

abbiamo esteso l'approccio **trigonometrico** al **caso 3D** con condizioni al bordo omogenee di tipo piú **generale**.



## 1 Fondamenti teorici

- Hierarchical Model Reduction in 3D
- Basi istruite

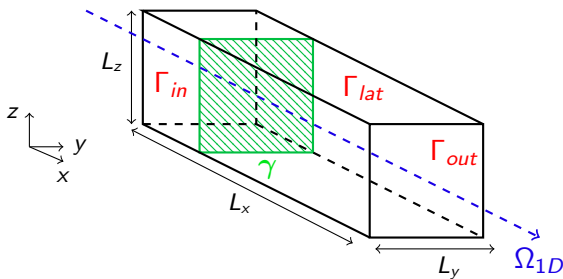
## 2 Implementazione

- Basis1DAbstract
- ModalSpace
- HiModAssembler

## 3 Risultati numerici

# Ipotesi geometriche

Dominio parallelepipedo



**Ipotesi** per le condizioni al bordo su  $\Gamma_{lat}$

- omogenee
- di ogni tipo ma con **coefficienti costanti**

# Base teorica

## Teorema spettrale per forme bilineari

Sia  $V$  uno spazio di tipo  $H^1$ . Sia  $a(\cdot, \cdot)$  una **forma bilineare** in  $V$ , continua, **simmetrica** e debolmente coerciva ( $a(u, u) \geq \alpha \|u\|_V^2 + \lambda_0 \|u\|_{L^2}^2$ ). Allora:

- (a) L'insieme degli autovalori è numerabile ed è una successione  $\{\lambda_m\}_{m \geq 1}$  tale che  $\lambda_m \rightarrow +\infty$ ;
- (b) se  $u, v$  sono autofunzioni corrispondenti ad autovalori differenti, allora

$$a(u, v) = 0 = (u, v)_{L^2}.$$

Inoltre,  $L^2$  ha una base ortonormale  $\{u_m\}_{m \geq 1}$  di autofunzioni di  $a$ ;

- (c) la successione  $\{u_m / \sqrt{\lambda_0 + \lambda_m}\}_{m \geq 1}$  è anche una base ortonormale in  $V$ , rispetto al prodotto scalare

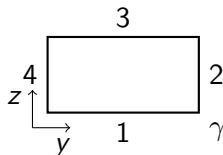
$$((u, v)) = a(u, v) + \lambda_0 (u, v)_{L^2}.$$

# Problema agli autovalori ausiliario

## Due sottoproblemi agli autovalori

**Problema agli autovalori su  $\gamma$ :** un caso con condizioni al bordo miste Dirichlet e Robin

$$\begin{cases} -\Delta u = \lambda u & \text{in } \gamma \\ u = 0 & \text{su } 1, 3, 4 \\ \mu \frac{\partial u}{\partial y} + \chi u = 0 & \text{su } 2 \end{cases}$$



**Ipotesi:**  $u(y, z) = Y(y)Z(z)$

$$\begin{cases} -\frac{Y''}{Y} - \frac{Z''}{Z} = \lambda & \longrightarrow -Y'' = K_y Y, \quad -Z'' = K_z Z, \quad \lambda = K_z + K_y \\ u = 0 \text{ su } 1, 3 & \longrightarrow Y(y)Z(1) = 0, \quad Y(y)Z(0) = 0 \quad \forall y \in (0, L_y) \\ u = 0 \text{ su } 4 & \longrightarrow Y(0)Z(z) = 0 \quad \forall z \in (0, L_z) \\ \mu \frac{\partial u}{\partial y} + \chi u = 0 \text{ su } 2 & \longrightarrow \mu \frac{\partial Y}{\partial y}(1)Z(z) + \chi Y(1)Z(z) = 0 \quad \forall z \in (0, L_z) \end{cases}$$

# Soluzione (semi-)analitica del problema agli autovalori

Soluzione dei due sottoproblemi e combinazione dei risultati

$$\begin{cases} -Y'' = K_y Y \text{ in } (0, L_y) \\ Y(0) = 0 \\ \mu Y'(L_y) + \chi Y(L_y) = 0 \end{cases}$$

$$\begin{cases} -Z'' = K_z Z \text{ in } (0, L_z) \\ Z(0) = 0 \\ Z(L_z) = 0 \end{cases}$$

$$\implies \{\varphi_{y,p}(y), K_{y,p}\}_{p=1}^{\infty}$$

$$\implies \{\varphi_{z,q}(z), K_{z,q}\}_{q=1}^{\infty}$$

**Combinando** queste successioni otteniamo le soluzioni del **problema 2D** agli autovalori su  $\gamma$

$$\{\varphi_k(y, z), \lambda_k\} = \{\varphi_{y,p}(y)\varphi_{z,q}(z), K_{y,p} + K_{z,q}\}$$

Due questioni da affrontare...

- Come risolvere il singolo problema agli autovalori ?
- Siamo interessati ad ordinare le  $\{\varphi_k(y, z)\}$  rispetto al valore di  $\{\lambda_k\}$ : come è possibile farlo a partire da  $\{K_{y,p}\}$  e  $\{K_{z,q}\}$  ?

# Risoluzione sottoproblema agli autovalori

Ricerca degli zeri

Forma della soluzione generale

$$\phi_{y,k}(y) = A_k \sin(w_{y,k}y) + B_k \cos(w_{y,k}y), \quad w_{y,k}^2 = K_{y,k}$$

<b>3</b> incognite	$\longleftrightarrow$	<b>2</b> condizioni di bordo
$A_k, B_k, w_{y,k}$		<b>1</b> normalizzazione

- Note:**
1. ortogonalità garantita dal teorema spettrale
  2. equazione spesso non lineare in  $w_{y,k}$

# Il problema della gerarchia

Esempio caso condizioni di Dirichlet

**Esempio:** condizioni di Dirichlet su  $\Gamma_{lat}$  con  $L_y = \pi$  e  $L_z = 3\pi/2$ .

Si ha che

$$K_{y,p} = p^2, K_{z,q} = (2/3q)^2.$$

$K_{y,p}$	1	p=1	4	p=2	9	p=3	1	p=1	4	p=2	...
$K_{z,q}$	4/9	q=1	4/9	q=1	4/9	q=1	16/9	q=2	16/9	q=2	...
$\lambda_k$	1.44	k=1	4.44	k=3	9.44	k=5	2.77	k=2	5.77	k=4	...

Per il problema dell'ordinamento abbiamo sviluppato l'algoritmo implementato in **EigensProvider**.

## 1 Fondamenti teorici

- Hierarchical Model Reduction in 3D
- Basi istruite

## 2 Implementazione

- Basis1DAbstract
- ModalSpace
- HiModAssembler

## 3 Risultati numerici

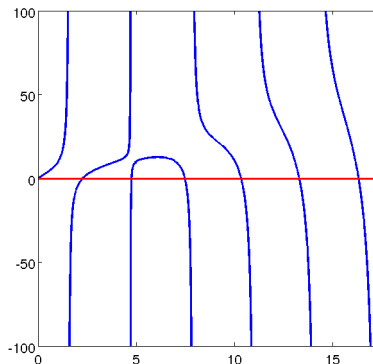


# Basis1DAbstract

Una classe astratta per risolvere i sottoproblemi agli autovalori

Il problema viene sempre rimappato nell'intervallo  $(0, 1)$

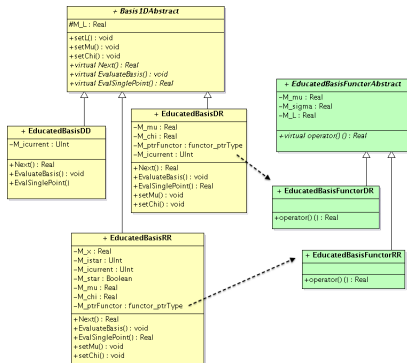
- Calcola gli autovalori  
→ `Next()`
- Valuta le funzioni di base  
→ `EvaluateBasis(...)`



Ricerca degli zeri con condizioni di Robin

# Polimorfismo su Basis1DAbstract...

... e su EducatedBasisFuncorAbstract



Per gestire le diverse classi figlie di Basis1DAbstract abbiamo utilizzato una **factory**.

## 1 Fondamenti teorici

- Hierarchical Model Reduction in 3D
- Basi istruite

## 2 Implementazione

- Basis1DAbstract
- **ModalSpace**
- HiModAssembler

## 3 Risultati numerici

# ModalSpace

Una classe che gestisce la costruzione dell'intera base modale

- Gestisce le formule di quadratura sulla slice
- Istanza i corretti generatori di base  
→ `AddSliceBC(...)`
- Calcola gli autovalori del problema 2D  
→ `EigensProvider(...)`
- Valuta le funzioni di base e le loro derivate nei nodi di quadratura  
→ `EvaluateBasis(...)`
- Calcola i coefficient  $r_{k,j}^{st}$   
→ `Compute_*(...)`

# AddSliceBC(...)

due metodi uno per ogni direzione

---

```
void ModalSpace::
AddSliceBCY (const string& left , const string& right , const
             Real& mu, const Real& chi)
{
// Creation of the correct basis generator
M_genbasisY = Basis1DFactory::instance().createObject(left+
              right);
// Setting of the parameters
M_genbasisY->setL(M_Ly);
M_genbasisY->setMu(mu);
M_genbasisY->setChi(chi);
return;
}
```

---

# EigenProvider()

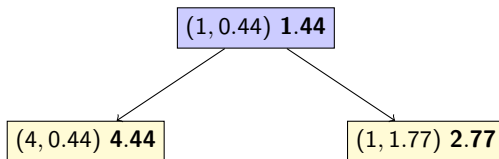
Ordinare correttamente gli autovalori non è facile

(1, 0.44) **1.44**

□ Nodi già salvati in `M_eigenvalues`, è un vector il cui indice, `k` è già associato all'`k`-esima **frequenza**.

# EigenProvider()

Ordinare correttamente gli autovalori non è facile



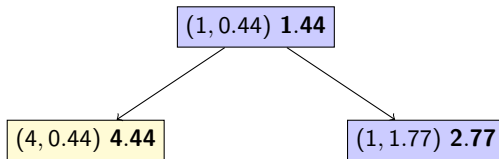
- Nodi già salvati in `M_eigenvalues`, è un vector il cui indice, `k` è già associato all'`k`-esima **frequenza**.

Nodi temporaneamente salvati in un set.

- E esso, grazie ad un'opportuna **relazione d'ordine**, è in grado di evitare i doppioni, senza rimuovere gli autovalori doppi e mantenendo sempre in prima posizione l'autovalore più piccolo pronto per l'estrazione.

# EigenProvider()

Ordinare correttamente gli autovalori non è facile



- Nodi già salvati in `M_eigenvalues`, è un vector il cui indice, `k` è già associato all'`k`-esima **frequenza**.

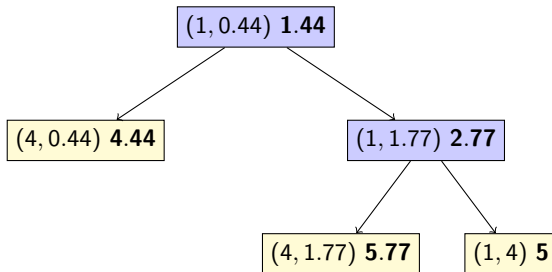
Nodi temporaneamente salvati in un set.

- E esso, grazie ad un'opportuna **relazione d'ordine**, è in grado di evitare i doppioni, senza rimuovere gli autovalori doppi e mantenendo sempre in prima posizione l'autovalore più piccolo pronto per l'estrazione.



# EigenProvider()

Ordinare correttamente gli autovalori non è facile



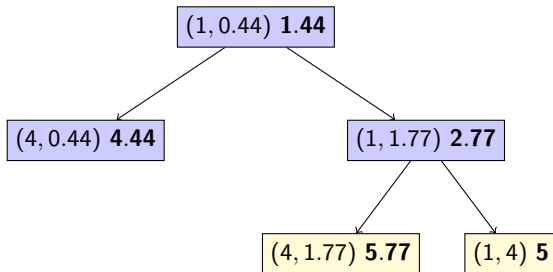
- Nodi già salvati in `M_eigenvalues`, è un vector il cui indice, `k` è già associato all `k`-esima **frequenza**.

Nodi temporaneamente salvati in un set.

- E esso, grazie ad un'opportuna **relazione d'ordine**, è in grado di evitare i doppioni, senza rimuovere gli autovalori doppi e mantenendo sempre in prima posizione l'autovalore più piccolo pronto per l'estrazione.

# EigenProvider()

Ordinare correttamente gli autovalori non è facile



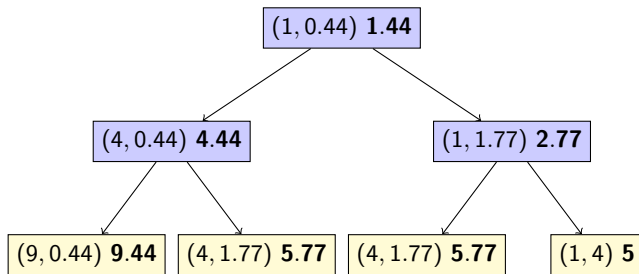
- Nodi già salvati in `M_eigenvalues`, è un vector il cui indice, `k` è già associato all'`k`-esima **frequenza**.

Nodi temporaneamente salvati in un set.

- E esso, grazie ad un'opportuna **relazione d'ordine**, è in grado di evitare i doppioni, senza rimuovere gli autovalori doppi e mantenendo sempre in prima posizione l'autovalore più piccolo pronto per l'estrazione.

# EigenProvider()

Ordinare correttamente gli autovalori non è facile



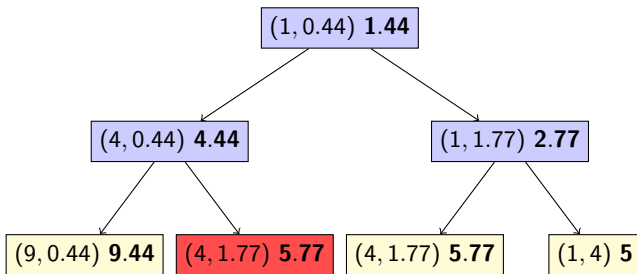
- Nodi già salvati in `M_eigenvalues`, è un vector il cui indice, `k` è già associato all'`k`-esima **frequenza**.


Nodi temporaneamente salvati in un set.

- E esso, grazie ad un'opportuna **relazione d'ordine**, è in grado di evitare i doppioni, senza rimuovere gli autovalori doppi e mantenendo sempre in prima posizione l'autovalore più piccolo pronto per l'estrazione.


# EigenProvider()


Ordinare correttamente gli autovalori non è facile



 Nodi già salvati in `M_eigenvalues`, è un vector il cui indice,  $k$  è già associato all  $k$ -esima **frequenza**.

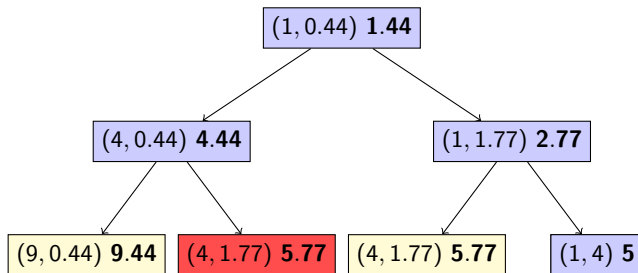
Nodi temporaneamente salvati in un set.

 E esso, grazie ad un'opportuna **relazione d'ordine**, è in grado di evitare i doppioni, senza rimuovere gli autovalori doppi e mantenendo sempre in prima posizione l'autovalore più piccolo pronto per l'estrazione.

 Nodi doppi **esclusi** dal set perché già presenti nell'albero.

# EigenProvider()

Ordinare correttamente gli autovalori non è facile



Nodi già salvati in `M_eigenvalues`, è un vector il cui indice, `k` è già associato all `k`-esima **frequenza**.

Nodi temporaneamente salvati in un set.

E esso, grazie ad un'opportuna **relazione d'ordine**, è in grado di evitare i doppioni, senza rimuovere gli autovalori doppi e mantenendo sempre in prima posizione l'autovalore più piccolo pronto per l'estrazione.

Nodi doppi **esclusi** dal set perché già presenti nell'albero.

# EvaluateBasis(...)

Valutare le funzioni di base sui nodi di quadratura

---

```
void ModalSpace::
EvaluateBasis()
{
    // Compute all the eigens from the 1D problems
    EigensProvider();

    // Fill the matrices with the evaluation of
    // the monodimensional basis and its derivatives
    M_genbasisY->EvaluateBasis (M_phiy, M_dphiy,
                                M_eigenvaluesY, M_quadruleY);
    M_genbasisZ->EvaluateBasis (M_phiz, M_dphiz,
                                M_eigenvaluesZ, M_quadruleZ);
}
```

---

# Compute\*(..)

Sfruttando la separazione di variabili

```
Real ModalSpace::
Compute_PhiPhi (const UInt& j, const UInt& k) const
{
    // ...
    // ... (Date le frequenze estrae i sottoindici p e q)
    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;
}
```

**WIP:** Abbiamo aggiunto dei metodi per considerare **coefficienti non costanti** che si occupano di calcolare i coefficienti  $r_{j,k}^{st}$  **senza separare le variabili**.

Sono ancora da testare e da ottimizzare.

## 1 Fondamenti teorici

- Hierarchical Model Reduction in 3D
- Basi istruite

## 2 Implementazione

- Basis1DAbstract
- ModalSpace
- HiModAssembler

## 3 Risultati numerici



- **Membri principali**

- `M_modalbasis`
- `M_etfespace`

- **Metodi per l'assemblaggio**

- `AddADRProblem(...)`
- `interpolate(...)`
- `Addrhs(...)`
- `AddDirichletBC_In(...)`

- **Metodi per l'analisi**

- `evaluateBase3DGrid(...)` *//HiMod vector*
- `evaluateBase3DGrid(...)` *//Function type*
- `normL2(...)`

- **Metodi per l'export**

- `ExportStructuredVTK(...)`
- `ExportFunctionVTK(...)`

# AddADRProblem(...)

Utilizzo del pacchetto ETA per assemblare i problemi 1D

```
for j {  
  for k {  
    VectorSmall<5> Coeff;  
    Coeff[0] = M_modalbasis->Compute_PhiPhi (j , k);  
    // ...  
    { using namespace ExpressionAssembly;  
      // ...  
      integrate(  
        elements ( M_etfespace->mesh() ) ,  
        M_fespace->qr() ,  
        M_etfespace ,  
        M_etfespace ,  
        mu*Coeff[0]*dot ( grad ( phi_i ) , grad ( phi_j ) )  
        // ...  
      )  
      >>(systemMatrix->block(k , j));  
    }  
    // ...  
  }  
  return ;  
}
```

## 1 Fondamenti teorici

- Hierarchical Model Reduction in 3D
- Basi istruite

## 2 Implementazione

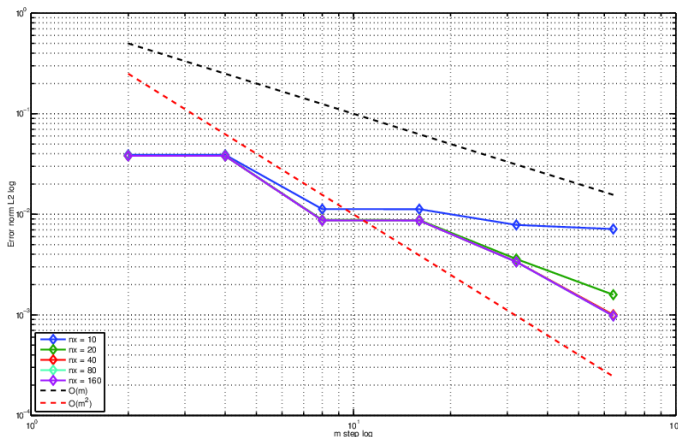
- Basis1DAbstract
- ModalSpace
- HiModAssembler

## 3 Risultati numerici

# Convergenza DRDR

$$C_1(x, y) = \frac{70}{1+x} \left( -\frac{2}{3L_y} y^3 + y^2 \right) \quad C_2(x, z) = \frac{70}{1+x} \left( -\frac{2}{3L_z} z^3 + z^2 \right)$$

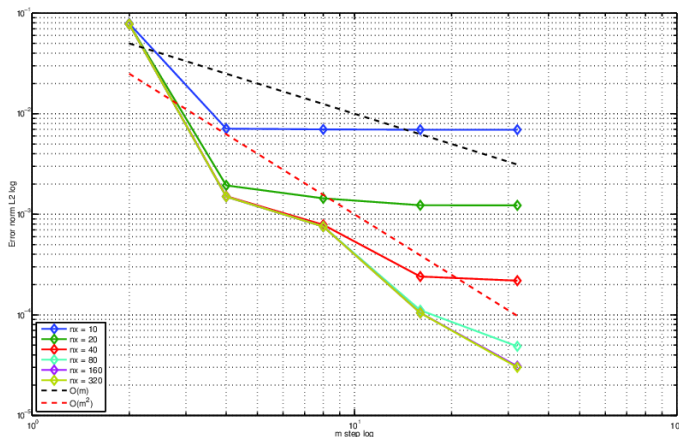
$$u_{es}(x, y, z) = 10^5 \exp \left( -\frac{\chi}{\mu L_y} (y - L_y/2)^2 + C_1 - \frac{\chi}{\mu L_z} (z - L_z/2)^2 + C_2 \right) (L_x - x)^2$$



# Convergenza RRRR

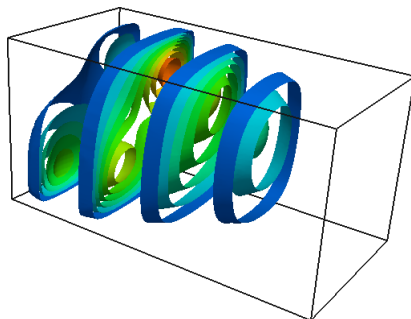
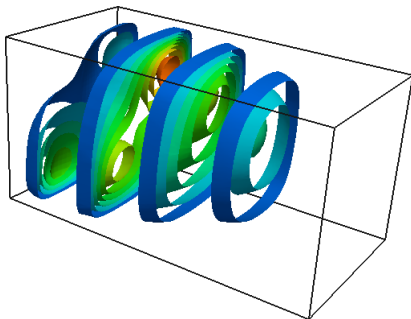
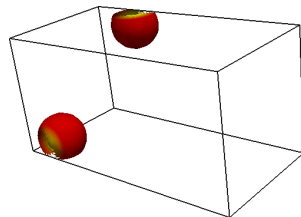
$$C_1(x, y) = \frac{70}{1+x} \left( -\frac{2}{3L_y} y^3 + y^2 \right) \quad C_2(x, z) = \frac{70}{1+x} \left( -\frac{2}{3L_z} z^3 + z^2 \right)$$

$$u_{es}(x, y, z) = 10^5 \exp \left( -\frac{\chi}{\mu L_y} (y - L_y/2)^2 + C_1 - \frac{\chi}{\mu L_z} (z - L_z/2)^2 + C_2 \right) (L_x - x)^2$$

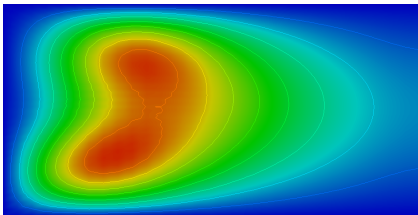


# Test camini

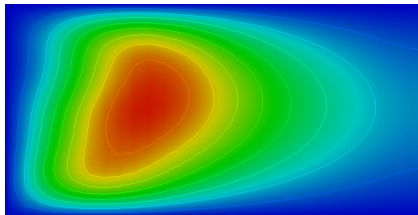
$$\begin{cases} -\mu\Delta u + b \cdot \nabla u + \sigma u = f & \text{in } \Omega \\ u = u_{in} & \text{su } \Gamma_{in} \\ \frac{\partial u}{\partial n} = 0 & \text{su } \Gamma_{out} \\ u = 0 & \text{su } \Gamma_{vaso} \end{cases}$$



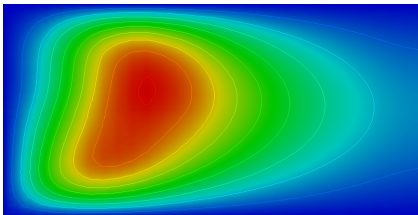
# Slice XY 2D



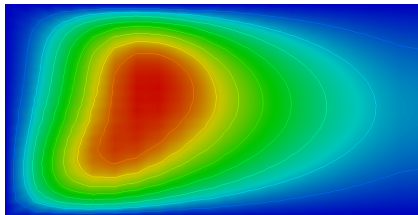
$m=9$



$m=16$

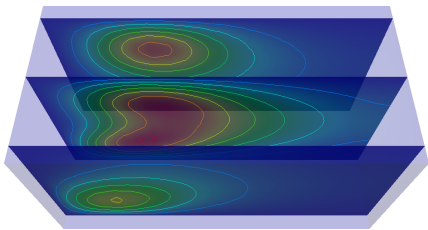


$m=25$

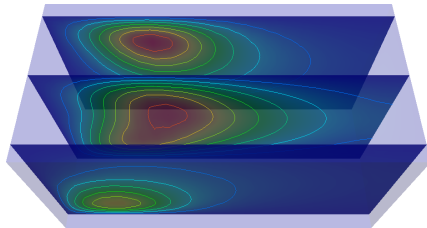


FEM

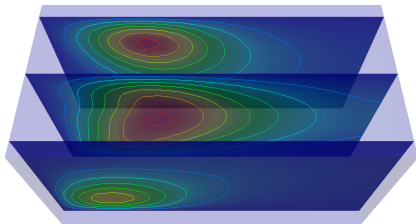
# Slice XY



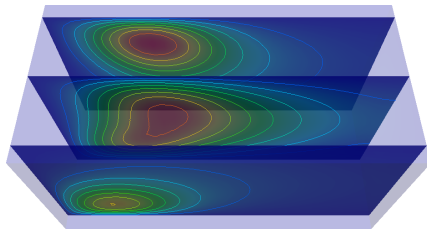
$m=9$



$m=16$



$m=25$



FEM



