SOLVING AN ADR PROBLEM USING A HIERARCHICAL MODEL WITH AN ISOGEOMETRIC ANALYSIS

Paolo Rusconi
Matricola 817294

Anno Accademico 2014-2015
Contents

1 The mathematical model 3
   1.1 Hierarchical Model (HiMod) reduction 4
      1.1.1 Geometrical analysis of the domain 4
      1.1.2 Reduction of the global model 5
      1.1.3 Well-posedness and convergence analysis 8
      1.1.4 Discretization of the reduced problem 9
      1.1.5 Algebraic formulation 10
   1.2 Isogeometric Analysis 11
      1.2.1 B-splines 11
      1.2.2 B-spline curves 14
      1.2.3 Non-uniform Rational B-splines 14

2 Implementation 17
   2.1 The IGAT00LS library 19
   2.2 Tpetra and Epetra data structures 19
   2.3 The algorithm 20
      2.3.1 A quick step-by-step explanation 20
      2.3.2 Assembling the matrix 21
      2.3.3 Assembling the rhs 23
      2.3.4 Solving the linear system and saving the results 23
   2.4 The IGAHiModAssembler class 24

3 Numerical results 26
   3.1 Conclusions 28
Chapter 1

The mathematical model

Nowadays, numerical methods and mathematical models have become indispensible instruments in many applications, from hemodynamics to fluid dynamics. However, the models used can be extremely complex and, as a consequence, the computational costs of the numerical techniques become too high. For this reason, it is crucial to create a numerical method which allows us to get the solution of the problems with a reasonable accuracy and reduces the costs.

So far, Hierarchical Model (HiMod) reduction techniques have been the answer to the problem. These models, which become particularly important for problems where the solution presents a main direction, rely on the idea of describe the axial and the transverse directions of the solution in a different way. In particular, a finite element discretization is used for the centerline while the other directions are described using a proper modal basis. Doing so, we get an enriched 1D problem from a 2D (or 3D) problem.

Even though the results obtained from HiMod methods are encouraging we want to investigate the possibility to use a different discretization for the axial dynamic. Since in many situations the geometry of the domain can be complex, we want to use basis functions which present better geometric properties. Thus, isogeometric elements come in our help. This new kind of elements presents a lot of advantages: better approximations, a new refinement strategy and lower number of degrees of freedom are needed.
1.1 Hierarchical Model (HiMod) reduction

In many engineering applications it is possible to identify a dominant direction, in particular in fluid dynamics models. In order to increase the computational efficiency Hierarchical Model (HiMod) reduction techniques have been introduced, in which the dominant and the transverse directions are treated in a different way. Generally, the former is spanned by a classical 1D piecewise polynomial basis of a finite element space, while the latter are expanded into a modal basis.

Usually the original problem is called full model and the starting point of our analysis is its weak formulation. If the mathematical model is an advection-diffusion-reaction problem the weak formulation is:

\[ \text{Find } u \in V \text{ such that: } \]
\[ a(u, v) = F(v) \quad \forall v \in V \]  

(1.1)

where \( V \) is a Hilbert space (generally \( V \subseteq H^1(\Omega) \)) chosen accordingly to the boundary conditions, \( a(\cdot, \cdot) \) is a bilinear form and \( F(\cdot) \) is a linear form. We assume that both \( a \) and \( F \) are continuous and \( a \) is coercive in order to guarantee the existence and the uniqueness of the solution thanks to Lax-Milgram lemma.

1.1.1 Geometrical analysis of the domain

Let \( \Omega \) be the domain of our problem. We assume that:

\[ \Omega = \bigcup_{x \in \Omega_{1D}} \{ x \} \times \gamma_x \]  

(1.2)

where \( \Omega_{1D} \) is the one-dimensional supporing fiber and \( \gamma_x \) is the transverse fiber associated to \( x \in \Omega_{1D} \). The dimension of \( \gamma_x \) is connected to the dimension of \( \Omega \), in fact if \( \Omega \subset \mathbb{R}^d \) then \( \gamma_x \subset \mathbb{R}^{d-1} \).

Assuming \( \Omega_{1D} \equiv [x_0, x_1] \) it is possible to split the boundary in three parts:

\[ \Gamma_0 = \{ x_0 \} \times \gamma_{x_0} \]  

(1.3a)

\[ \Gamma_1 = \{ x_1 \} \times \gamma_{x_1} \]  

(1.3b)

\[ \Gamma_* = \bigcup_{x \in \Omega_{1D}} \{ x \} \times \partial \gamma_x \]  

(1.3c)

In particular we define:

\[ \Gamma_D = \Gamma_* \quad \Gamma_N = \Gamma_1 \cup \Gamma_0 \]  

(1.4)

where \( \Gamma_D \) and \( \Gamma_N \) are the boundaries with Dirichlet and Neumann conditions respectively.
CHAPTER 1. THE MATHEMATICAL MODEL

The supporting fiber can be curved in many cases and it can be analyzed introducing a map \( \Psi : \Omega \rightarrow \hat{\Omega} \) that connects the domain \( \Omega \) with the reference domain \( \hat{\Omega} = \hat{\Omega}_{1D} \times \hat{\gamma}_{d-1} \). In particular, \( \hat{\Omega}_{1D} \) will be a straight line and \( \hat{\gamma}_{d-1} \) a reference fiber of the same dimension as \( \gamma_{d-1} \). However this kind of problems will not be studied in this work.

1.1.2 Reduction of the global model

We proceed now with the formulation of the reduced model.

First of all, it is necessary to define the functional spaces of the different directions:

1. the space \( V_{1D} \subseteq H^1(\Omega_{1D}) \) used for the mainstream direction. Its nature depends on the boundary conditions of the problem;
2. the modal basis \( \{ \varphi_k \}_{k \in \mathbb{N}} \subseteq H^1(\gamma_x) \) used on the transverse directions. These functions will be such that:

\[
\int_{\hat{\gamma}_{d-1}} \varphi_i(y)\varphi_j(y) \, dy = \delta_{ij} \quad \forall \, i, j \in \mathbb{N}
\] (1.5)

There are multiple choices for these functions: trigonometric and Legendre polynomials are the most common ones.

Combining \( V_{1D} \) with the modal functions it is possible to define the reduced space

\[
V_m = \left\{ v_m(z) = \sum_{k=1}^{m} v_k(\Psi_1(z))\varphi_k(\Psi_2(z)) \mid \text{with } v_k \in V_{1D} \right\},
\] (1.6)

where \( m \in \mathbb{N} \) is an a priori fixed number. The value \( m \) will be constant over the whole domain, even if it is possible to find other methods where \( m \) may change in some parts of the domain where more informations are needed.

Furthermore, from the \( L^2 \)-ortonormality of \( \{ \varphi_k \} \) we can evaluate the frequency coefficients

\[
v_k(\Psi_1(z)) = \int_{\gamma_x} v_m(\Psi_1(z), \Psi_2(z))\varphi_k(\Psi_2(z)) \, d\Psi_2(z) \quad \text{with } k = 1, \ldots, m
\] (1.7)

At this point, the weak formulation of the reduced model is:

For a certain \( m \in \mathbb{N} \), find \( u_m \in V_m \) such that:

\[
a(u_m, v_m) = F(v_m) \quad \forall \, v_m \in V_m.
\] (1.8)

We proceed now analyzing more in detail the uniform Hi-Mod reduction procedure on a specific differential problem. In particular, the full model will
CHAPTER 1. THE MATHEMATICAL MODEL

be a scalar advection-diffusion-reaction (ADR) problem with full homogeneous Dirichlet boundary conditions. This means $V = H^1_0(\Omega)$ and

$$a(u, v) = \int_{\Omega} \mu \nabla u \cdot \nabla v \, d\Omega + \int_{\Omega} (b \cdot \nabla u + \sigma u) v \, d\Omega$$

(1.9a)

$$F(v) = \int_{\Omega} f v \, d\Omega$$

(1.9b)

In order to guarantee the well-posedness of the weak form we will use: $f \in L^2(\Omega), \mu \in L^\infty(\Omega)$ with $\mu \geq \mu_0 > 0$ a.e. in $\Omega$, $\sigma \in L^\infty(\Omega)$, $b = (b_1, b_2)^T \in L^\infty(\Omega) \times [L^\infty(\Omega)]^{d-1}$, with $\nabla \cdot b \in L^\infty(\Omega)$ and such that $-\frac{1}{2} \nabla \cdot b + \sigma \geq 0$ is a.e. in $\Omega$.

Now we replace $u_m$ with the corresponding modal representation:

$$u_m(z) = \sum_{j=1}^{m} u_j(\Psi_1(z))\varphi_j(\Psi_2(z))$$

(1.10)

while choosing the test function as

$$v_m(z) = \theta(\Psi_1(z))\varphi_k(\Psi_2(b))$$

(1.11)

where $k = 1, \ldots, m$, $\theta$, $u_j, v \in V_{\Omega_1D} = H^1_0(\hat{\Omega}_1D)$ for $j = 1, \ldots, m$. We get

$$\sum_{j=1}^{m} \left[ \int_{\Omega} \mu(z) \nabla(u_j(\Psi_1(z))\varphi_j(\Psi_2(z))) \cdot \nabla(\theta(\Psi_1(z))\varphi_k(\Psi_2(z))) \, d\Omega 
+ \int_{\Omega} b(z) \cdot \nabla(u_j(\Psi_1(z))\varphi_j(\Psi_2(z)))\theta(\Psi_1(z))\varphi_k(\Psi_2(z)) \, d\Omega 
+ \int_{\Omega} \sigma(z) u_j(\Psi_1(z))\varphi_j(\Psi_2(z))\theta(\Psi_1(z))\varphi_k(\Psi_2(z)) \, d\Omega \right]$$

$$= \int_{\Omega} f(z)\theta(\Psi_1(z))\varphi_k(\Psi_2(z)) \, d\Omega$$

(1.12)

where $\nabla$ denotes the gradient with respect to $z$. The unknowns of the Hi-Mod reduced formulation are the coefficients $u_j \in V_{\Omega_1D}$. We proceed expanding separately all the integrals by exploiting the gradient expansion

$$\nabla(w(\Psi_1(z))\varphi(\Psi_2(z))) =

w'(\Psi_1(z))\varphi(\Psi_2(z)) \begin{bmatrix} \frac{\partial \Psi_1(z)}{\nabla x} \\ \frac{\partial \Psi_1(z)}{\nabla y} \end{bmatrix} + w(\Psi_1(z))\varphi'(\Psi_2(z)) \begin{bmatrix} \frac{\partial \Psi_2(z)}{\nabla x} \\ \frac{\partial \Psi_2(z)}{\nabla y} \end{bmatrix},$$

(1.13)

where $w'(\Psi_1(z)) = dw/d\hat{x}|_{z=\Psi_1(z)}$, $\varphi'(\Psi_2(z)) = dw/d\hat{y}|_{\hat{y}=\Psi_2(z)}$ and with $w \in \tilde{V}_{\Omega_1D}$. From now on, we will rewrite all the terms on the reference
domain, using properly the maps functions $\Psi$ and the inverse map $\Phi : \hat{\Omega} \to \Omega$. We start with the diffusive contribution:

$$\int_{\hat{\Omega}} \mu(\Phi(\hat{z})) \left\{ \left[ \left( \frac{\partial \Psi_1(\Phi(\hat{z}))}{\partial x} \right)^2 + (\nabla_y \Psi_1(\Phi(\hat{z})))^2 \right] \phi_j(y) \phi_k(y) u_j(\hat{x}) \theta(\hat{x}) 
+ \left[ \frac{\partial \Psi_1(\Phi(\hat{z}))}{\partial x} \frac{\partial \Psi_2(\Phi(\hat{z}))}{\partial x} + \nabla_y \Psi_1(\Phi(\hat{z})) \nabla_y \Psi_2(\Phi(\hat{z})) \right] 
\phi_j(y) \phi_k(y) \phi_j(y) \phi_k(y) u_j(\hat{x}) \theta(\hat{x}) \right\} \right\}$$

(1.14)

with $\mathcal{J}$ the jacobian defined thanks the map $\Psi$. The convective term is changed into

$$\int_{\hat{\Omega}} \left\{ \left[ b_1(\Phi(\hat{z})) \frac{\partial \Psi_1(\Phi(\hat{z}))}{\partial x} + b_2(\Phi(\hat{z})) \nabla_y \Psi_1(\Phi(\hat{z})) \right] \phi_j(y) \phi_k(y) u_j(\hat{x}) \theta(\hat{x}) 
+ \left[ b_1(\Phi(\hat{z})) \frac{\partial \Psi_2(\Phi(\hat{z}))}{\partial x} + b_2(\Phi(\hat{z})) \nabla_y \Psi_2(\Phi(\hat{z})) \right] \phi_j(y) \phi_k(y) u_j(\hat{x}) \theta(\hat{x}) \right\}$$

$$\mathcal{J}^{-1}(\Phi(\hat{z})) d\hat{\Omega},$$

(1.15)

while, for the reactive term, we have

$$\int_{\hat{\Omega}} \sigma(\Phi(\hat{z})) \phi_j(y) \phi_k(y) u_j(\hat{x}) \theta(\hat{x}) \mathcal{J}^{-1}(\Phi(\hat{z})) d\hat{\Omega}.$$  

(1.16)

Lastly, for the right-hand side we obtain

$$\int_{\hat{\Omega}} f(\Phi(\hat{z})) \phi_k(y) \theta(\hat{x}) \mathcal{J}^{-1}(\Phi(\hat{z})) d\hat{\Omega}.$$  

(1.17)

Immediately we observe that the diffusion term generates advective and reactive contributions in the reduced setting. The reduced convection term creates a reactive contribution, too. Combining all the parts of the integral leads to the following Hi-Mod reduced formulation for the ADR problem:

Find $u_j \in V_{\Omega,D}$ with $j = 1, \ldots, m$, such that, for any $\theta \in V_{\Omega,D}$ and $k = 1, \ldots, m$,

$$\sum_{j=1}^{m} \left\{ \int_{\Omega_{1,D}} \left[ \hat{\tau}_{kj}^{1,1}(\hat{x}) u_j(\hat{x}) \theta(\hat{x}) + \hat{\tau}_{kj}^{1,0}(\hat{x}) u_j(\hat{x}) \theta(\hat{x}) 
+ \hat{\tau}_{kj}^{0,1}(\hat{x}) u_j(\hat{x}) \theta(\hat{x}) + \hat{\tau}_{kj}^{0,0}(\hat{x}) u_j(\hat{x}) \theta(\hat{x}) \right] d\hat{x} \right\}$$

$$= \int_{\hat{\Omega}_{1,D}} \int_{\gamma_{d-1}} f(\Phi(\hat{z})) \phi_k(y) \mathcal{J}^{-1}(\Phi(\hat{z})) |d\hat{y}| \theta(\hat{x}) d\hat{x},$$

(1.18)
CHAPTER 1. THE MATHEMATICAL MODEL

where

\[ \tilde{r}_{kj}^{s,t}(\tilde{x}) = \int_{\gamma_{d-1}} r_{kj}^{s,t}(\tilde{x}, \tilde{y})|J^{-1}(\Phi(\tilde{z}))|d\tilde{y} \quad s, t = 0, 1, \quad k = 1, \ldots, m, \] (1.19)

with

\[
\begin{align*}
\tilde{r}_{kj}^{1,1}(\tilde{z}) &= \mu(\Phi(\tilde{z}))\alpha_i(\tilde{z})\varphi_j(\tilde{y})\varphi_k(\tilde{y}) \\
\tilde{r}_{kj}^{0,1}(\tilde{z}) &= \mu(\Phi(\tilde{z}))\delta(\tilde{z})\varphi_j'(\tilde{y})\varphi_k(\tilde{y}) \\
\tilde{r}_{kj}^{1,0}(\tilde{z}) &= \mu(\Phi(\tilde{z}))\delta(\tilde{z})\varphi_j(\tilde{y})\varphi_k'(\tilde{y}) + \beta_1(\tilde{z})\varphi_j(\tilde{y})\varphi_k(\tilde{y}) \\
\tilde{r}_{kj}^{0,0}(\tilde{z}) &= \mu(\Phi(\tilde{z}))\alpha_2(\tilde{z})\varphi_j'(\tilde{y})\varphi_k'(\tilde{y}) + \beta_2(\tilde{z})\varphi_j'(\tilde{y})\varphi_k(\tilde{y}) + \sigma(\Phi(\tilde{z}))\varphi_j(\tilde{y})\varphi_k(\tilde{y}),
\end{align*}
\] (1.20)

and

\[
\begin{align*}
\alpha_i(\tilde{z}) &= \left( \frac{\partial \Psi_i(\Phi(\tilde{z}))}{\partial x} \right)^2 + (\nabla_y \Psi_i(\Phi(\tilde{z})))^2 \quad i = 1, 2 \\
\beta_i(\tilde{z}) &= b_1(\Phi(\tilde{z}))\frac{\partial \Psi_i(\Phi(\tilde{z}))}{\partial x} + b_2(\Phi(\tilde{z})) \cdot \nabla_y \Psi_i(\Phi(\tilde{z})) \quad i = 1, 2 \\
\delta(\tilde{z}) &= \frac{\partial \Psi_1(\Phi(\tilde{z}))}{\partial x} \frac{\partial \Psi_2(\Phi(\tilde{z}))}{\partial x} + \nabla_{by} \Psi_1(\Phi(\tilde{z})) \cdot \nabla_y \Psi_2(\Phi(\tilde{z})).
\end{align*}
\] (1.21)

In the reduced model the dependence of the solution on the dominant and on the transverse directions is split. The Hi-Mod reduction procedure yields a \textit{enriched one-dimensional model} associated with the main curved stream, whose coefficients, \( \tilde{r}_{kj}^{s,t} \), are properly enriched to include the effects of the transverse components.

Solving the reduced problem is equivalent to solving a system of \( m \) coupled 1D problems. For this reason, if \( m \) is small enough the computational costs will be significantly lower than the full model ones. However, the choice of \( m \) represents a nontrivial task.

1.1.3 Well-posedness and convergence analysis

In order to guarantee the well-posedness and the convergence estimates of the reduced problem, it is necessary to assume more hypotheses:

1. the \textit{conformity hypothesis} which means that \( V_m \subset V \) for every choice of \( m \in \mathbb{N} \);

2. the \textit{spectral approximability hypothesis} which means that

\[
\forall \, v \in V \quad \lim_{m \to +\infty} \left( \inf_{v_m \in V_m} \|v - v_m\|_V \right) = 0 \quad (1.22)
\]
CHAPTER 1. THE MATHEMATICAL MODEL

Thanks to the first hypothesis and the well-posedness of the original problem, we obtain that the reduced problem is well-defined. The conformity hypothesis ensures also that the modelling error $e_m \in V$, defined as $e_m = u - u_m$, satisfies the modeling orthogonality property

$$a(e_m, v_m) = 0 \quad \forall v_m \in V_m$$

which leads us to the spectral optimality property

$$\|e_m\|_V \leq C \inf_{v_m \in V_m} \|u - v_m\|_V$$

where the constant $C$ depends on the constants of continuity and coercivity of the bilinear form. The convergence of $u_m$ follow thanks to the spectral approximability hypothesis.

It is important to notice that the conformity hypothesis can hold only if we make good assumptions on both the boundary conditions and the smoothness of the map $\Psi$. Furthermore, the choice of the modal basis and the regularity of $u$ set the rate of the modal convergence.

1.1.4 Discretization of the reduced problem

The next step of our analysis is the definition of the discrete reduced formulation of the problem and for this reason we start introducing a partition of $\Omega_1D$. Let $T_h$ be a subdivision of $\Omega_1D$ into subintervals $K_j = (x_{j-1}, x_j)$ of width $h_j = x_j - x_{j-1}$. We define then $h = \max_j h_j$. After this, we introduce a conforming finite element space $V_{1D}^h \subset V_{1D}$ associated with $T_h$. The dimension of this space will be $N_h < +\infty$. Before proceeding, it is necessary to add the density hypothesis: for all $u_{1D} \in V_{1D}$ we have that $\lim_{h \to 0} d_{1D}(u_{1D}, V_{1D}^h) = 0$, where we used $d_{1D}(\cdot, \cdot)$ as distance induced by the norm $\|\cdot\|_{1D} := \|\cdot\|_{H^1(\Omega_{1D})}$ in $V_{1D}$.

Now, we can write the discrete reduced formulation as follow:

Find $u_m^h \in V_m^h$ such that:

$$a(u_m^h, v_m^h) = F(v_m^h) \quad \forall v_m^h \in V_m^h$$

where

$$V_m^h = \left\{ v_m^h(z) = \sum_{k=1}^m v_k^h(\Psi_1(z))\varphi_k(\Psi_2(z)) \quad \text{with} \quad v_k^h \in V_{1D}^h \right\}$$

and, thanks to the conformity hypothesis, we obtain $V_m^h \subset V_m$.

Some convergence results are available. Let $e_m^h = u - u_m^h$ be the global error which includes both model $(u - u_m)$ and the discretization $(u_m - u_m^h)$ error contributions. In particular situations, the synergy of these errors in
fundamental in order to get the correct convergence rate of the global error. Infact, in such cases finding the optimal number $m$ of modal basis functions becomes crucial, because otherwise the model error will cause a slowdown of the global convergence rate.

Under reasonable conditions, it is possible to demonstrate the convergence of the discrete reduced solution to the solution of the full model.

### 1.1.5 Algebraic formulation

The last step will be the building of the algebraic system. Using the definition of $u_h^m \in V_h^m$ in the reduced formulation we obtain:

Find $\{u_h^j\}_{j=1}^m \in [V_{1D}^h]^m$ such that:

$$\sum_{j=1}^m a(u_h^j \varphi_j, \theta_l \varphi_k) = F(\theta_l \varphi_k) \quad (1.27)$$

with $k \in \{1, \ldots, m\}$ and $\theta_l$ is a generic basis function of the discrete space $V_{1D}^h$ (for $l \in \{1, \ldots, N_h\}$). Now, we substitute $u_h^j$ remembering that

$$u_h^j(\Psi_1(z)) = \sum_{i=1}^{N_h} u_h^{j,i} \theta_i(\Psi_1(z)) \quad (1.28)$$

in order to obtain a linear system that will be characterized by an $mN_h \times mN_h$ block matrix $A$. Figure 1.1 shows the pattern of the HiMod matrix $A$. The indeces $k$ and $j$, associated with the modes, are used for identifying the macrostructures of $A$, while $l$ and $i$ are related to discretization of the central fiber, so they are used in each block. Observing a single block $A_{kj}$ (its dimension is $N_h \times N_h$) it is easy to recognize the standard sparsity pattern of the finite element approximation used.

Figure 1.1: Sketch of the linear system corresponding to the discrete reduced formulation.
1.2 Isogeometric Analysis

As stated before, in most situations we work with extremely complex geometries and the process of translation from Computer Aided Design (CAD) files to meshes that will be used in the finite elements codes is the bottleneck that limits the efficiency of the codes. It has been estimated that 80% of the time of the analysis is spent during this passage. The idea to pass from a Finite Element Analysis (FEA) to a CAD representation is called Isogeometric Analysis.

This kind of analysis still consists in a isoparametric approach but where both the geometry and the unknown variables of the problem are described by basis functions generated from Non-Uniform Rational B-Splines (generally called NURBS) that are the standard for describing and modeling curves in computer aided design and computer graphics.

In order to understand what NURBS are it is necessary to take a step back and start from the basics concepts.

1.2.1 B-splines

The first notion we introduce are the B-spline curves because NURBS derive from them. The elements that identify a B-spline are the knot vector and the basis functions.

The knot vector

In one dimension, the knot vector is a set of non-decreasing coordinates in the parameter space:

$$\Xi = \{\xi_1, \xi_2, \ldots, \xi_{n+p+1}\}$$  \hspace{1cm} (1.29)

where $\xi_i \in \mathbb{R}$ indicates the $i$th knot, $p$ is the polynomial order and $n$ is the number of the basis functions used in order to build the B-spline curve.

There are different type of knot vectors. When all the nodes are equally distributed it is called uniform vector, otherwise it is called non uniform vector. If the first and the last nodes have a multiplicity of $p + 1$, it is called open vector. In one-dimensional cases, these vectors are frequently used because the basis functions are interpolatory at the ends of the domain.
The basis functions

Once the knot vector is defined, it is possible to build the B-splines basis functions through the following relation, known as the Cox-de Boor recursion formula, starting with piecewise constants ($p = 0$):

\[
N_{i,0}(\xi) = \begin{cases} 
1 & \text{if } \xi_i \leq \xi \leq \xi_{i+1} \\
0 & \text{otherwise}
\end{cases}
\]  

(1.30)

Then, for $p \geq 1$, the other functions are defined by

\[
N_{i,p}(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi)
\]  

(1.31)

Figure 1.2 shows that for $p = 0$ and $p = 1$ the basis functions are the same as the finite elements one, while for $p \geq 2$ we can observe some differences. In fact, we can observe that the quadratic basis functions present the same basis function for all the elements, so there is not a differentiation between internal and end nodes. This happens also with higher orders.

Figure 1.2: Basis functions of order $p = \{0, 1, 2\}$ for a uniform knot vector $\Xi = \{0, 1, 2, 3, 4, \ldots\}$. 
These basis functions present several important features. First of all, they constitute a partition of unity:

\[ \sum_{i=1}^{n} N_{i,p}(\xi) = 1 \quad \forall \xi \]  

(1.32)

Furthermore, each element of the basis is pointwise nonnegative over the entire domain. As we can observe from figure 1.3, this property is not satisfied by the finite element basis functions with \( p \geq 2 \). Another important characteristic of B-spline basis functions is that each of them has \( p - 1 \) continuous derivatives within the knots and increasing the order \( p \) their supports grow. This leads to the common misconception that the increasing of the support of the functions cause the growth of the bandwidth in a numerical method, but actually this does not happen. In fact, each function shares support with (including itself) \( 2p + 1 \) elements of the basis.

![Figure 1.3: Bandwidth comparison for \( C^0 \) FEA cubics and \( C^2 \) B-spline cubic functions. The bandwidth of the resulting matrices is \( 2p + 1 = 7 \).](image)

In particular circumstances non-uniform knot vectors allow us to obtain better results. For instance, modifying the multiplicity of the nodes it is possible to make the basis functions interpolatory only for specific elements of the mesh.
CHAPTER 1. THE MATHEMATICAL MODEL

Generally, each basis function has \( p - m_i \) continuous derivatives, where \( m_i \) indicates the multiplicity of the \( i^{th} \) node. If \( m_i = p \) the function becomes interpolatory while if \( m_i = p + 1 \) the function becomes discontinuous and we obtain the border of the patch.

The derivatives of the B-spline basis functions are defined through polynomials of lower orders. Given the polynomial order \( p \) and the knot vector \( \Xi \), the derivative of the \( i^{th} \) basis function is

\[
\frac{d}{d\xi} N_{i,p}(\xi) = \frac{p}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) - \frac{p}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi)
\]

This relation can be generalized in order to evaluate derivatives of higher orders.

1.2.2 B-spline curves

In \( \mathbb{R}^d \), B-spline curves are obtained with the linear combination of the basis functions introduced in the previous pages. The values of these functions in the nodes are called control points and they represent the counterparts of the nodal values of the finite elements.

Given the \( n \) basis and the corresponding control point \( B_i \in \mathbb{R} \), we define a polynomial B-spline curve using the following relation:

\[
C(\xi) = \sum_{i=1}^{n} N_{i,p}(\xi) B_i
\]

The properties of the curves are consequence of the characteristics of the basis functions. For example, a B-spline curve of order \( p \) will have \( p - 1 \) continuous derivatives if all nodes are not repeated. These curves can be refined in several ways: increasing the number of elements, increasing the order of the basis and increasing the continuity of the basis. Now I will describe each technique shortly.

1.2.3 Non-uniform Rational B-splines

Now that the main properties of the B-spline curves have been discussed, it is possible to introduce the notion of Non-Uniform Rational B-Spline (NURBS). The passage from B-spline to NURBS is fundamental because we will be able to represent a lot of geometric objects that normally polynomials cannot represent. In particular, we introduce NURBS through a geometric analysis and then an algebraic one.

The geometric point of view

It is possible to obtain a NURBS in \( \mathbb{R}^d \) through a projective transformation of a B-spline entity in \( \mathbb{R}^{d+1} \). For example, using piecewise quadratic curves
we can construct exactly the main conic sections. The B-spline \( C_w(\xi) \) is called projective curve while \( B^w_i \) are called projective control points. The terms curve and control points are used only for the NURBS objects \( C(\xi) \) and \( B_i \). We obtain the control points using these relations:

\[
(B_i)_j = \frac{(B^w_i)}{w_i} \quad j = 1, \ldots, d \tag{1.35}
\]

\[
w_i = (B^w_i)_{d+1} \tag{1.36}
\]

where \((B_i)_j\) indicates the \( j^{th} \) component of the vector \( B_i \) and \( w_i \) indicates the \( i^{th} \) weight. Almost in any situation these values are considered to be positive.

![Figure 1.4: (a) Projective transformation of projective control point \( B^w_i \) yields control point \( B_i \). (b) Projective transformation of the piecewise quadratic B-spline curve \( C^w(\xi) \) yields the NURBS curve \( C(\xi) \) (circle in \( \mathbb{R}^2 \)).](image)

Before defining a NURBS curve, it is necessary to introduce the weighting function:

\[
W(\xi) = \sum_{i=1}^{n} N_{i,p}(\xi) w_i \tag{1.37}
\]
In particular, in $\mathbb{R}^3$ we will have $W(\xi) = z(\xi)$, which means the weight will be the height of the curve.

Now, we can define a NURBS curve as:

$$ (C(\xi))_j = \left(\frac{C^w(\xi)}{W(\xi)}\right)_j \quad j = 1, \ldots, d \quad (1.38) $$

Clearly, the curve obtained is a piecewise rational function because both $C^w(\xi)$ and $W(\xi)$ are piecewise polynomial functions.

Lastly, it is important to notice that if we multiply all the projective control points by a constant, the NURBS curve obtained will be the same.

**The algebraic point of view**

In order to build a NURBS curve more efficiently we want to define a basis for the NURBS space directly from the knot vector. Once we have that, it is possible to define all the curves from the linear combination of these new basis functions and the control points.

First of all, we define the basis functions of the NURBS space using this formula:

$$ R^p_i(\xi) = \frac{N_{i,p}(\xi)w_i}{W(\xi)} = \frac{N_{i,p}(\xi)w_i}{\sum_{j=1}^{n} N_{j,p}(\xi)w_j} \quad (1.39) $$

Now, if we use this relation with the one that defines the control points we obtain the equation which is commonly used for the construction of NURBS curves:

$$ C(\xi) = \sum_{i=1}^{n} R^p_i(\xi)B_i \quad (1.40) $$

All the properties, such as continuity of the functions, follow directly from the knot vector. Furthermore, these basis are a partition of the unity and are pointwise nonnegative, too.

We can easily point out that B-splines are a special case of NURBS. Infact, if all the weights are the same we have $R^p_i(\xi) = N_{i,p}(\xi)$.

Moreover, since NURBS basis functions are constructed from B-spline, it is possible to define the derivatives using this relation:

$$ \frac{d}{d\xi} R^p_i(\xi) = w_i \frac{W(\xi)N'_{i,p}(\xi) - W'(\xi)N_{i,p}(\xi)}{(W(\xi))^2} \quad (1.41) $$

where:

$$ N'_{i,p}(\xi) = \frac{d}{d\xi} N_{i,p}(\xi) \quad W'(\xi) = \sum_{j=1}^{n} N'_{j,p}(\xi)w_i \quad (1.42) $$

Now that we have faced all the theoritecal features of the numerical method, we can start discussing about its implementation.
Chapter 2

Implementation

The objective of this work is the implementation of a Hierarchical Model reduction method with an isogeometric approach and to test it on a 3D advection-diffusion-reaction problem. In particular, it will be a problem with homogeneous Neumann conditions on the inflow and outflow sides while on the lateral faces of the domain homogeneous Dirichlet conditions are imposed. Furthermore, all the parameters of the problem are constant. The code has been created in LifeV and it can be found inside the branch IGA_HiMOD.

First of all, we used cmake for the compilation. It is a cross-platform, open-source build system. It is mainly used for the control of software compilation processes. cmake is widely used because it generates native makefiles and workspaces that can be used in the compiler environment. In this framework, we used a particular version: TriBITS. The name means Tribal Build, Integrate, and Test System. It is generally used with large software development projects involving multiple independent development teams and multiple source repositories.

One of the first things we have to do is the setup of all the Task Parallel Libraries (TPL) necessary for the project. All the informations are written in the file TPLsList.cmake. In the following picture are showed all the libraries used. They are written in a specific order, accordingly to their dependencies.

```cmake
SET( LifeV_TPLS_FINDMODS_CLASSIFICATIONS
    MPI "${PROJECT_NAME}_TRIBITS_DIR}/tpls/
    BLAS "cmake/TPLs/
    LAPACK "cmake/TPLs/
    Boost "cmake/TPLs/
    ParMETIS "cmake/TPLs/
    HDF5 "cmake/TPLs/
    QHull "cmake/TPLs/
    Trilinos "cmake/TPLs/
    IGATools "cmake/TPLs/
)
```

17
The last two letters of each line are used for the classification of the libraries. In fact PS means *Primary Stable* and it is used for all the essential libraries for LifeV development and precheckin testing; while on the other hand, SS means *Secondary Stable* and it is for those TPLs that are not required in order to be able to develop and test LifeV before checkins. There are other two possible labels for the classification, however in this framework are not used.

Another important point of the setup, which can be considered relevant for the analysis, is the *CMakeLists.txt* file used for the two examples.

```cmake
INCLUDE(TribitsAddExecutableAndTest)
INCLUDE(TribitsCopyFilesToBinaryDir)

INCLUDE_DIRECTORY(${CMAKE_SOURCE_DIR})

TRIBITS_ADD_EXECUTABLE_AND_TEST(HM_IGA
    SOURCES main.cpp
    ARGS -c
    NUM_MPI_PROCS 2
    COMM serial mpi
)

TRIBITS_COPY_FILES_TO_BINARY_DIR(data_HM_IGA
    SOURCE_FILES data
    SOURCE_DIR ${CMAKE_CURRENT_SOURCE_DIR}
)

TRIBITS_COPY_FILES_TO_BINARY_DIR(ParameterList_HM_IGA
    SOURCE_FILES SolverParamList.xml
    SOURCE_DIR ${CMAKE_CURRENT_SOURCE_DIR}
)
```

We remark that inside the TriBITS directory the lines

```cmake
IF (${PROJECT_NAME}_ENABLE_CPP11)
    PREPEND_CMNDLINE_ARGS(GENERAL_BUILD_FLAGS "-std=c++11")
ENDIF()

IF (${PROJECT_NAME}_ENABLE_CPP14)
    PREPEND_CMNDLINE_ARGS(GENERAL_BUILD_FLAGS "-std=c++1y")
ENDIF()
```

have been added since we are going to use IGATOOLS library and it needs both the *boost 1.59* library and some features of *c++14*, such as *decltype* (auto).
CHAPTER 2. IMPLEMENTATION

2.1 The IGATOOLS library

Since this project aims the implementation of isogeometric elements for the discretization of the centerline in a HiMod framework, a specific library which allows us to create such elements becomes indispensable.

In particular, the IGATOOLS library has been used. All the codes can be easily downloaded and installed from the site \url{https://code.google.com/p/igatools/}.

This library is an open source used for solving partial differential equations using isogeometric spaces (such as B-Splines and NURBS). Clearly, this one is not the only library that could be used for the implementation of this kind of finite elements, however it presents good data structures such as Tpetra vectors and matrix which can easily be inserted inside the LifeV environment and it uses Trilinos for its linear algebra. Trilinos is a big library with a lot of algorithms and it is widely used for the solution of large-scale, complex multi-physics engineering and scientific problems.

2.2 Tpetra and Epetra data structures

The crucial point for the synergy of these two libraries (LifeV and IGATOOLS) consists in the communication between Tpetra and Epetra objects when the code will run on multiple processors. The first are used by IGATOOLS while the latter by LifeV.

Tpetra allows the implementation of linear algebra objects, such as sparse matrices and dense vectors. It is defined as hybrid parallel because uses at least two kinds of parallelism: MPI and various shared-memory parallel programming models within each MPI process, such as OpenMP, POSIX Threads (Pthreads) and Nvidia’s CUDA.

Furthermore, Tpetra differs from its predecessor Epetra at least under three different aspects:

1. Tpetra has native support for solving very large problem (with over 2 billion unknowns);
2. Tpetra lets you construct matrices and vectors with different kinds of data, such as floating-point types of different precision, or complex-values types. Epetra objects can only support double-precision floating-point data (of type double);
3. Tpetra can exploit many different kinds of hybrid parallelism and has optimizations for shared-memory parallel systems with nonuniform memory access (NUMA), while Epetra can only support OpenMP shared-memory parallelism for a few kernels;
On the other hand, Epetra is based on objects called Maps that contain the details of the distribution of the data over MPI processes. Each Map instance represents a particular data distribution and it can be seen as a representation of a vector space.

Even though this project presents only a serial version of the code, the implementation of these types of data structures is crucial for future studies for a parallelization of the code.

2.3 The algorithm

Now that we have shown the main features, we can start analyzing the main code.

The original codes are the ones written by Sofia Guzzetti and Massimiliano Lupo Pasini for the resolution of a classic 3D advection-diffusion-reaction problem using the classical HiMod approach.

Clearly, the main objects inserted by them cannot be used anymore for many reasons. All the library of LifeV is structured around the use of linear finite elements, so many classes cannot be used since we are going to replace FEM with B-spline basis functions using IGATOOLS.

Fortunately, we can still use all the code available for the creation of the modal basis which is crucial because now the only missing step will be the creation of a B-Spline space from the grid of the domain and to use the basis functions for the assembling of the system matrix and the right hand side (rhs).

2.3.1 A quick step-by-step explanation

First of all, thanks to GetPot we acquire all the informations such as dimension of the domain, the number of modes $m$, the number of elements, etc. Clearly, this has been done in order to customize the parameters without the need of modifying directly the code.

Once the parameters of the problem are defined we can proceed with the definition of the grid and space objects using specific methods of the classes iga::CartesianGrid and iga::BSplineSpace.

```cpp
// Definition of the grid and of the B-Spline space
auto grid = iga::CartesianGrid<dim>::create(n_knots);
auto space = iga::BSplineSpace<dim>::create(degree, grid);
```

After that, we can proceed with the creation of the ModalSpace object like in the original code, since all the informations about it are independent from the discretization technique chosen for the centerline. The following lines show the creation of the modal basis for a problem with a domain characterized by a rectangular section.
// Definition of the modal basis
const Real Ly = dataFile("himod/ly", 1.);
const Real Lz = dataFile("himod/lz", 1.);
boost::shared_ptr<ModalSpaceRectangular> MB (new ModalSpaceRectangular(Ly,Lz,mtot));

Obviously, once the modal basis is defined it is necessary to add the boundary conditions.

// Applying lateral homogeneous boundary conditions
MB->addSliceBCY("dir", "dir");
MB->addSliceBCZ("dir", "dir");

// Computing the basis
MB->evaluateBasis();

In the main code there are also the grid HMgrid and the functional space HMSpace which are needed for the creation of the global matrix and of the rhs vector.

Now that all the pieces are ready we can start with the assembling of the matrix final_mat calling the method assembly_matrix and then with the assembling of the rhs final_rhs using assembly_rhs. These two functions are methods of the new class IGAHiModAssembler and they will be presented in detail later.

2.3.2 Assembling the matrix

As mentioned before, the matrix we want to build is composed by $m \times m$ blocks and each of this blocks presents the pattern of the finite element discretization technique chosen. For example, if we use linear FE then each block will have the classic tridiagonal pattern. The function assembles the global matrix creating a singular block matrix and then adding the informations in the global matrix in the correct rows and columns.
CHAPTER 2. IMPLEMENTATION

// Iteration on the blocks
for (UInt k=0; k < M_modalbasis->mtot(); ++k)
    for (UInt l=0; l < M_modalbasis->mtot(); ++l)
    {
        // Evaluation of HiMod coeffs
        // Creation of local matrix

        for ( ; elem!=elem_end ; ++elem)
        {
            // Evaluation of element basis, gradient
            // and weights on quadrature nodes

            // Local assembling
        }
    }

But what happens during the local assembling? Every local matrix is built adding the values of each element, for this reason we need other for cycles.

// Local assembling
for (int i=0; i < n_basis; ++i)
{
    // Values of the i-th basis function
    // Values of the i-th gradient

    for (int j=0; j < n_basis; ++j)
    {
        // Values of the j-th basis function
        // Values of the j-th gradient

        for (int qp=0; qp < n_qp; ++qp)
        {
            // Saving the results using
            // a reasonable quadrature rule
        }
    }
}

We want to emphasize one thing about the quadrature rule: the number n_qp is not fixed. Generally, in an isogeometric context this number is defined once the polynomial degree of the basis function is chosen: if the order of the basis is p, then the number of nodes for the quadrature rule will be p + 1. This guarantees the exactness of the quadrature formula.

Once the (j,l)-th block is built, it is necessary to add the informations in the global matrix. In order to do it we simply need to know the IDs of the rows and of the columns and then we use the method add_block. Lastly, we call the method fill_complete in order to complete the assembling of the matrix.
2.3.3 Assembling the rhs

The right hand side vector of the problem is built following the same philosophy of the global matrix. The vector is composed of \( m \) blocks and each one has the dimension of the local functional space. As we can see from the following scheme, there is only one external \texttt{for} cycle for the block while and one internal \texttt{for} cycle. We evaluate the informations element after element and using the same quadrature rule of the previous function.

```cpp
// Iteration on the block
for (UInt k=0; k < MB->tot(); ++k)
{
    // Evaluation of HiMod coeffs
    // Creation of local vector
    for ( ; elem!=elem_end; ++elem)
    {
        // Evaluation of element basis, gradient
        // and weights
        for (int i=0; i < n_basis; ++i)
        {
            // Values of the i-th basis function
            for (int qp=0; qp < n_qp; ++qp)
            {
                // Saving the results using
                // a reasonable quadrature rule
            }
        }
    }
}
```

At this moment, it is not possible to use non-constant forcing term because we need the values of the coefficients of the Fourier series on the quadrature nodes. However, LifeV environment relies on a two-points quadrature rule so when cannot have a suitable interpolated vector for the forcing term. Clearly it is a problem that can be solved, substituting the actual quadrature rule with a more dynamic version that uses a number of quadrature nodes which depends on the degree of the basis functions used by the user.

2.3.4 Solving the linear system and saving the results

Now that we have both the global matrix and the rhs vector of the linear system we can solve the problem using the \texttt{belos} library and choosing proper method like coniugate gradient \texttt{CG}.
CHAPTER 2. IMPLEMENTATION

The last step missing is the conversion of the solution. Once the linear problem is solved, we have a vector with all the coefficients however, as mentioned in the first section, the B-Spline basis functions are not interpolatory so it is necessary to interpolate the values over the grid points. This is possible using some lines of code of the writer class of IGATOOLS library.

2.4 The IGAHiModAssembler class

For the resolution of the 3D ADR problem we have implemented a new class called IGAHiModAssembler. Since a 3D domain can present different kind of sections, such as a rectangular or a circular one, it can be useful to define this class as a template one. The template parameter is an int value that indicates the section geometry. If the user uses 0 it means he is dealing with a rectangular section, otherwise, if he uses 1, with a circular one.

The introduction of this class is indispensable because all the other versions of HiModAssembler rely on the use of finite element spaces which are not be used in this context.

So far, it is possible to treat two types of section: rectangular and circular. For this reason there are the classes IGAHiModAssemblerRectangular and IGAHiModAssemblerCircular which are fully described in the following files:

- IGAHiModAssemblerRectangular_decl.hpp
- IGAHiModAssemblerRectangular_def.hpp
- IGAHiModAssemblerCircular_decl.hpp
- IGAHiModAssemblerCircular_def.hpp

In the decl header files there is the declaration of the classes with a specific section (which is suggested by the name of the file). In each file there are also the declaration of the methods which we will describe. The methods are:

- assembly_matrix. This method, as the name suggests, is used in order to assemble the matrix of the linear system. It has only an input parameter which is an empty igamatrix_ptrType (a pointer to a Tpetra) matrix which will be filled during the assembling. As mentioned few pages ago, the assembling of the matrix is possible using several for cycles. This is necessary because we will move on each block and within each block we will move on each element of the grid. During each block iteration we initialize an empty iga::DenseMatrix, we update the coefficients of the modal basis functions and of the boundary conditions. Instead, during the internal cycles, we use proper methods of the iga library for the evaluation of the basis functions (get_basis_values) and the respective gradients (get_basis_gradients) on each element.
in the quadrature nodes in order to evaluate the coefficients of the matrix with the best precision possible. The number of quadrature points and weights depends on the polynomial degree of the basis: if the user wants to use $p$ as polynomial order then the algorithm will use $p + 1$ quadrature nodes.

Once a block is assembled, we transfer the informations contained in the dense matrix into the Tpetra global matrix, given as a parameter to the method.

- **assembly_rhs.** This method imitates the previous one but it assembles the global right hand side of the linear system. The philosophy of the assembling is the same of the previous method with the only difference that it does not cycle globally on two indeces, but on only one. For now, this method can be used using only a constant forcing term. This limit is due to the fact the actual interpolating methods are based on two points quadrature rules, since they are enough for linear basis functions. But in an isogeometric context the codes must be updated with a more dynamic version that uses $p + 1$ nodes, where $p$ is the polynomial order of the basis.

- **solve.** Once the matrix and the vector of the system are prepared, we call this method in order to solve the algebraic problem. First we define a linear solver of the belos library and then we obtain the solution setting a proper method such as the conjugate gradient CG (however it is possible to use other methods).

- **grid_evaluationData and grid_evaluation.** The first method is used directly in the main code and it gets as input parameters the main geometric informations of the problem and of the domain. Inside this method we call the the second one where we interpolate the solution on the grid points. This step is crucial since the B-spline basis functions are not interpolatory. Once the interpolated coefficients are obtained we evaluate the solution over the 3D grid and then we use the exporter in order to plot the results.

- **showMe.** This is the simplest among all the methods and it used only for showing some informations to the user such as the number of degrees of freedom, the global dimension of the linear system and the parameters of the model.
Chapter 3

Numerical results

Once the implementation is complete, it is fundamental to test if the code works properly. Since specific setups are not available yet, we cannot use models with an analytic solution in order to evaluate the approximation errors. Thus, we tested the code on a 3D Poisson problems with $\mu = 1$ and $f = 1$.

\[
\begin{align*}
-\mu \Delta u &= f \quad \text{in } \Omega \\
\mu \nabla u \cdot \mathbf{n} &= 0 \quad \text{on } \partial \Omega_{in} \cup \partial \Omega_{out} = \partial \Omega_{io} \\
u &= 0 \quad \text{on } \partial \Omega \setminus \partial \Omega_{io}
\end{align*}
\]

We used a FreeFem++ script in order to get at least a full finite element $L^2$-norm of the solution for a comparison with the results obtained from our method and the classical HiMod formulation.

For the rectangular case, the full finite element solution presents a $L^2$-norm equal to 0.04126 while both the HiMod formulations return a solution with a $L^2$-norm equal to 0.04115. Clearly, it is not possible to observe any difference between using finite elements or isogeometric ones, since in this context the solution is independent from the axial coordinate. We tried simulations with $p = 1$ (which is equivalent to using FEM since the basis functions are the same) and $p = \{2, 3, 4\}$.

On the other hand, the results obtained using the circular section are less accurate. In fact the $L^2$-norm of the classical solution is equal to 0.255765, while the norm of the solutions we got from the HiMod formulations are equal to 0.3104. Since these results seems to be independent from the choice of the discretization, the error is probably due to the modal basis chosen. However, in the following page it is possible to observe that the solution seems to be a reasonable approximation.
Figure 3.1: Graphic solutions of the 3D Poisson problem on both rectangular and circular domains.
3.1 Conclusions

This is just the first and little step toward a full implementation of this new kind of finite elements with the Hierarchial Model reduction framework. With this work we have just tested the terrain, because we wanted to see if there was a way to link different libraries in order to use the best instruments from both of them and the answer seems to be affirmative.

However the road ahead is still pretty long. There are still many options that can be implemented, such as the customization of the centerline of the domain in terms of length and profile and the possibility to use non-constant parameters $\mu$, $b$, $\sigma$ and $f$. Clearly, a user will be interested in boundary conditions, too. So, there is still much work to do, however well begun half in done.
Bibliography


