archives-ouvertes.fr

High Performance Computing for the Reduced Basis Method. Application to Natural Convection

Elisa Schenone, Stéphane Veys, Christophe Prud'Homme

To cite this version:

Elisa Schenone, Stéphane Veys, Christophe Prud'Homme. High Performance Computing for the Reduced Basis Method. Application to Natural Convection. ESAIM: Proceedings, EDP Sciences, 2013, 43, pp.255-273. <10.1051/proc/201343016>. <hal-00786560>

HAL Id: hal-00786560

https://hal.archives-ouvertes.fr/hal-00786560

Submitted on 8 Feb 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

ESAIM: PROCEEDINGS, Vol. ?, 2013, 1-10
Editors: Will be set by the publisher

HIGH PERFORMANCE COMPUTING FOR THE REDUCED BASIS METHOD. APPLICATION TO NATURAL CONVECTION

Elisa Schenone[1,2], Stéphane Veys[3] and Christophe Prud'homme[4]

Abstract. In this paper, we are interested in studying of the reduced basis methodology (RBM) applied to steady-state natural convection problems. The latter has applications in many engineering domains and being able to apply the RBM would allow to gain huge computation savings when querying the model the reduced model for many parameter evaluations. In this work, we focus on the order reduction of the model — in particular the handling of the non-linear terms, — as well as the design of the RBM computational framework and the requirements on high performance computing to treat 3D models using Feel++, a C++ open source library to solve partial differential equations. Numerical experiments are presented on a 2D and 3D models.

Introduction

Nowadays, in many application fields, engineering problems require accurate, reliable, and efficient evaluation of quantities of interest. Often, these quantities of interest depend on the solution of a parametrized partial differential equation where the — *e.g.* physical or geometrical — parameters are inputs of the model and the evaluation of quantities of interest are outputs — *e.g.* average values. — In a real-time or many-query context, the reduced basis method (RBM) offers a rapid and reliable evaluation of the input-output relationship (see [PRV+02, VPP03, VPRP03, PP04, QRM11, RHP07] for the methodology) for a large class of problems.

In this paper, we are interested in studying the RBM applied to steady-state natural convection problems parametrized by the Grashof and Prandtl numbers. Natural convection has applications in many engineering domains and being able to apply the RBM would allow to gain huge computation savings when querying the model the reduced model for many parameter evaluations. In this work, we focus on the order reduction of the model — in particular the handling of the non-linear terms, — as well as the design of the RBM computational framework and the requirements on high performance computing (HPC) to treat 3D models. Even though the model considered remains simple with respect to industrial applications, we tackle some of the main difficulties namely order reduction and computational costs.

In order to solve Finite Elements (FE) or Reduced Basis (RB) problems, we use an open-source library called Feel++ for *Finite Element Embedded Library and Language in C++* ([PCD+12, Pru06]). Feel++ is a library to solve problems arising from partial differential equations (PDEs) with Galerkin methods, standard or generalized, continuous or discontinuous, from 1D to 3D, for low to high order approximations (including geometry). Among the many other Feel++ features, it provides a seamless programming environment with

[1] Laboratoire Jacques Louis Lions, UPMC, 4 Place Jussieu, 75005 Paris, France
[2] Inria Paris-Rocquencourt, Domaine de Voluceau B.P. 105, 78153 Le Chesnay Cedex, France - elisa.schenone@inria.fr
[3] Laboratoire Jean Kuntzmann, Université Joseph Fourier Grenoble 1, BP53 38041 Grenoble Cedex 9, France - stephane.veys@imag.fr
[4] Université de Strasbourg, IRMA UMR 7501, 7 rue René-Descartes, 67084 Strasbourg Cedex, France - prudhomme@unistra.fr
© EDP Sciences, SMAI 2013

respect to parallel computing using MPI, see section 4.1. Feel++ enjoys an implementation of the RBM, see [DVTP13], which can deal with a wide range of problems: elliptic or parabolic models, coercive or non-coercive models, linear or non-linear models. It can handle coupled non-linear multiphysic problems such as the thermo-electric problems in [VDPT12, VCD+12]. It is important that such an environment hides as many implementation details as possible and let the user worry only about his/her model and the high level aspects of the FEM and RBM.

The organisation of the paper is as follows: in section 1 we describe the 2D and 3D steady-state natural convection models; in section 2, we present the finite element discretization and the solution strategy; in section 3, we apply the RBM and focus in particular on the non-linear terms handling; in section 4, we present the computational framework for FEM and RBM as well as some implementation aspects; finally in section 5, we display some numerical experiments in 2D and 3D.

1. Problem setting

We start with the description of a standard natural convection model. We consider a heated fluid in a squared or cubical cavity, the fluid circulates towards the low temperature under the action of density and gravity differences. We introduce the adimensionalized steady-state incompressible Navier-Stokes equations coupled with the heat equation and we consider the problem in a two and a three dimensional tank, see *e.g.* Figure 1: find (u, p, T) such that

$$\begin{cases} \mathbf{u}\cdot\nabla\mathbf{u} + \nabla p - \dfrac{1}{\sqrt{Gr}}\triangle\mathbf{u} &= T\mathbf{e_2} \quad , \text{ in } \Omega \\ \nabla\cdot\mathbf{u} &= 0 \quad , \text{ in } \Omega \\ \mathbf{u}\cdot\nabla T - \dfrac{1}{\sqrt{Gr}Pr}\triangle T &= 0 \quad , \text{ in } \Omega \\ \mathbf{u} &= \mathbf{0} \quad , \text{ on } \partial\Omega \\ T &= 0 \quad , \text{ on } \Gamma_1 \\ \dfrac{\partial T}{\partial \mathbf{n}} &= 0 \quad , \text{ on } \partial\Omega\setminus(\Gamma_1\cup\Gamma_3) \\ \dfrac{\partial T}{\partial \mathbf{n}} &= 1 \quad , \text{ on } \Gamma_3. \end{cases} \tag{1}$$

where $\Omega \subset \mathbb{R}^d$, $d = 2, 3$, $\mathbf{u}$, p and T are respectively the adimensionalized velocity, pressure and temperature, Gr and Pr are the Grashof and the Prandtl numbers, and $\mathbf{e_2}$ is the inward-pointing normal vector of a border $\Gamma_2 \subset \partial\Omega$. The 2D domain consists in a rectangular tank of height 1 and length W, in the 3D case we consider a rectangular cuboid of height 1, length W and depth 1. A heat flux is imposed on the "right" border Γ_3 while the temperature is fixed on the "left" border Γ_1 and the remaining walls are insulated. Similar boundary conditions apply in 3D. No-slip boundary conditions are set for the fluid velocity in the tank.

The parameters are the Grashof and the Prandtl numbers and we consider the Nusselt number — the average temperature on Γ_3 — as the output. As the Grashof and/or Prandtl numbers increase the Nusselt is decreasing, see *e.g.* figure 5(a) or figure 6(a).

2. Finite element formulation

In this section, we write the weak Galerkin formulations associated to (1) and we propose an iterative method to solve this problem in case of FE space and a resolution with the RBM.

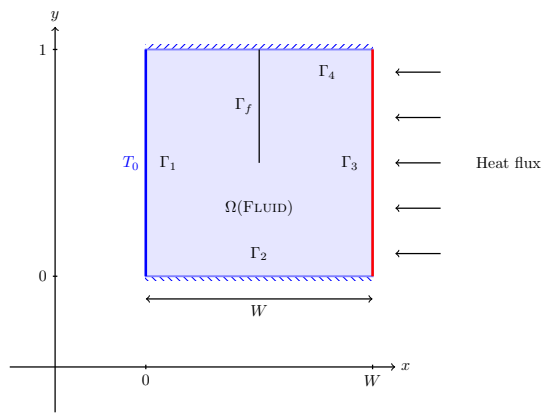


FIGURE 1. Geometry of the 2D model. Consider an extrusion of this geometry in 3D case is the extrusion in z axis of length 1.

The weak formulation associated to problem (1) writes

$$\begin{aligned} a(\mathbf{u},\mathbf{u},\mathbf{v}) - b(p,\mathbf{v}) + \frac{1}{\sqrt{Gr}}c(\mathbf{u},\mathbf{v}) - d(T,\mathbf{v}) &= 0 \quad ,\forall \mathbf{v}\in V \\ b(q,\mathbf{u}) &= 0 \quad ,\forall q \in Q \\ e(T,\mathbf{u},\xi) + \frac{1}{\sqrt{Gr}Pr}g(T,\xi) - \frac{1}{\sqrt{Gr}Pr}h(\xi) &= 0 \quad ,\forall \xi \in \Xi \end{aligned} \tag{2}$$

where $V \equiv [H_0^1(\Omega)]^d$, $Q \equiv L^2(\Omega)$, $\Xi \equiv \{\xi \in H^1(\Omega) \text{ s.t. } \xi|_{\Gamma_1} = 0\}$, and we define the tri-linear forms $a : V \times V \times V \to \mathbb{R}$ and $e : \Xi \times V \times \Xi \to \mathbb{R}$ as

$$a(\mathbf{u},\mathbf{w},\mathbf{v}) = \int_\Omega (\mathbf{w}\cdot\nabla\mathbf{u})\cdot\mathbf{v}, \quad \forall \mathbf{u},\mathbf{w},\mathbf{v}\in V \tag{3}$$

$$e(T,\mathbf{v},\xi) = \int_\Omega (\mathbf{v}\cdot\nabla T)\,\xi, \quad \forall \mathbf{v}\in V, T,\xi\in\Xi \tag{4}$$

the bi-linear forms $b : Q \times V \to \mathbb{R}$, $c : V \times V \to \mathbb{R}$, $d : \Xi \times V \to \mathbb{R}$ and $g : \Xi \times \Xi \to \mathbb{R}$ as

$$b(q,\mathbf{v}) = \int_\Omega q\nabla\cdot\mathbf{v}, \ \forall \mathbf{v}\in V, q\in Q \tag{5}$$

$$c(\mathbf{w},\mathbf{v}) = \int_\Omega \nabla\mathbf{w} : \nabla\mathbf{v}, \ \forall \mathbf{w},\mathbf{v}\in V \tag{6}$$

$$d(\xi,\mathbf{v}) = \int_\Omega \xi\,\mathbf{e_2}\cdot\mathbf{v}, \ \forall \mathbf{v}\in V, \ \xi\in\Xi \tag{7}$$

$$g(T,\xi) = \int_\Omega \nabla T\cdot\nabla\xi, \ \forall T, \ \xi\in\Xi \tag{8}$$

and the linear operator $h : \Xi \to \mathbb{R}$ as

$$h(\xi) = \int_{\Gamma_3} \xi, \ \forall \xi\in\Xi. \tag{9}$$

We introduce now a FE discretization of (2). We define the discrete spaces $V_h \subset V$, $Q_h \subset Q$, $\Xi_h \subset \Xi$ and the associated Galerkin projection of the solution $(\mathbf{u}, p, T)$ of (2) by

$$V_h \equiv \text{span}\{\boldsymbol{\phi}_1, \dots, \boldsymbol{\phi}_{N_u}\} \ , \quad \mathbf{u} \simeq \sum_{j=1}^{N_u} \mathbf{u}_j \boldsymbol{\phi}_j$$
$$Q_h \equiv \text{span}\{\psi_1, \dots, \psi_{N_p}\} \ , \quad p \simeq \sum_{j=1}^{N_p} p_j \psi_j$$
$$\Xi_h \equiv \text{span}\{\xi_1, \dots, \xi_{N_T}\} \ , \quad T \simeq \sum_{j=1}^{N_T} T_j \xi_j .$$

The discrete formulation of (2) now reads

$$\begin{aligned}
\sum_{i,j=1}^{N_u} \mathbf{u}_i \mathbf{u}_j a(\boldsymbol{\phi}_i, \boldsymbol{\phi}_j, \boldsymbol{\phi}_{k_1}) + \sum_{i=1}^{N_u} \frac{1}{\sqrt{Gr}} \mathbf{u}_i c(\boldsymbol{\phi}_i, \boldsymbol{\phi}_{k_1}) - \sum_{i=1}^{N_p} p_i b(\psi_i, \boldsymbol{\phi}_{k_1}) - \sum_{i=1}^{N_T} T_i d(\xi_i, \boldsymbol{\phi}_{k_1}) = 0 \ , \quad k_1 = 1, \dots, N_u \\
\sum_{i=1}^{N_u} \mathbf{u}_i b(\psi_{k_2}, \boldsymbol{\phi}_i) = 0 \ , \quad k_2 = 1, \dots, N_p \\
\sum_{i=1}^{N_T} \sum_{j=1}^{N_u} T_i \mathbf{u}_j e(\xi_i, \boldsymbol{\phi}_j, \xi_{k_3}) + \frac{1}{\sqrt{Gr} Pr} \sum_{i=1}^{N_T} T_i g(\xi_i, \xi_{k_3}) - \frac{1}{\sqrt{Gr} Pr} h(\xi_{k_3}) = 0 \ , \quad k_3 = 1, \dots, N_T
\end{aligned} \tag{10}$$

Due to its strong non-linearities, when the Grashof or Prandtl numbers are high, a robust iterative method is required to solve this problem. We apply here a Newton Method. For a given parameter $\boldsymbol{\mu} = (\mu_1, \mu_2) = (Gr^{-1/2}, Pr^{-1})$ and an initial guess $(\mathbf{u}^0, p^0, T^0)$, at each Newton sub-iteration $n = 1, \dots, n_{max}$ we look for $(\mathbf{u}^{n+1}, p^{n+1}, T^{n+1}) \in \mathbb{R}^{N_u} \times \mathbb{R}^{N_p} \times \mathbb{R}^{N_T}$ such that

$$J(\mathbf{u}^n, p^n, T^n; \boldsymbol{\mu}) \left[(\mathbf{u}^{n+1}, p^{n+1}, T^{n+1}) - (\mathbf{u}^n, p^n, T^n) \right] = R(\mathbf{u}^n, p^n, T^n; \boldsymbol{\mu}) \tag{11}$$

where $J = J(\mathbf{u}, p, T; \boldsymbol{\mu})$ is the Jacobian matrix, $R = R(\mathbf{u}, p, T; \boldsymbol{\mu})$ is the residual vector.

In the case of problem (10) the terms of the Jacobian matrix can be easily calculated. For each row $k_1 = 1, \dots, N_u$ they write

$$\begin{aligned}
J_{k_1 i}(\mathbf{u}, p, T; \boldsymbol{\mu}) &= \sum_{j=1}^{N_u} \mathbf{u}_j a(\boldsymbol{\phi}_i, \boldsymbol{\phi}_j, \boldsymbol{\phi}_{k_1}) + \sum_{j=1}^{N_u} \mathbf{u}_j a(\boldsymbol{\phi}_j, \boldsymbol{\phi}_i, \boldsymbol{\phi}_{k_1}) + \mu_1 c(\boldsymbol{\phi}_i, \boldsymbol{\phi}_{k_1}) \ , && i = 1, \dots, N_u \\
J_{k_1 N_u + i}(\mathbf{u}, p, T; \boldsymbol{\mu}) &= -b(\psi_i, \boldsymbol{\phi}_{k_1}) \ , && i = 1, \dots, N_p \\
J_{k_1 N_u + N_p + i}(\mathbf{u}, p, T; \boldsymbol{\mu}) &= -d(\xi_i, \boldsymbol{\phi}_{k_1}) \ , && i = 1, \dots, N_T
\end{aligned} \tag{12}$$

for all row $k_2 = N_u + k$, $k = 1, \dots, N_p$

$$\begin{aligned}
J_{k_2 i}(\mathbf{u}, p, T; \boldsymbol{\mu}) &= b(\psi_k, \boldsymbol{\phi}_i) \ , && i = 1, \dots, N_u \\
J_{k_2 N_u + i}(\mathbf{u}, p, T; \boldsymbol{\mu}) &= 0 \ , && i = 1, \dots, N_p \\
J_{k_2 N_u + N_p + i}(\mathbf{u}, p, T; \boldsymbol{\mu}) &= 0 \ , && i = 1, \dots, N_T
\end{aligned} \tag{13}$$

and for all line $k_3 = N_u + N_p + k$, $k = 1, \dots, N_T$

$$
\begin{aligned}
J_{k_3 i}(\mathbf{u}, p, T; \boldsymbol{\mu}) &= \sum_{j=1}^{N_T} T_j e(\xi_j, \boldsymbol{\phi}_i, \xi_k) \ , && i = 1, \dots, N_u \\
J_{k_3 N_u + i}(\mathbf{u}, p, T; \boldsymbol{\mu}) &= 0 \ , && i = 1, \dots, N_p \\
J_{k_3 N_u + N_p + i}(\mathbf{u}, p, T; \boldsymbol{\mu}) &= \sum_{j=1}^{N_u} \mathbf{u}_j e(\xi_i, \boldsymbol{\phi}_j, \xi_k) + \mu_1 \mu_2 g(\xi_i, \xi_k) \ , && i = 1, \dots, N_T.
\end{aligned} \tag{14}
$$

In the same way, each term of the residual $R(\mathbf{u}, p, T; \boldsymbol{\mu}) \in \mathbb{R}^{N_u + N_p + N_T}$ can be calculated by

$$
\begin{aligned}
R_k(\mathbf{u}, p, T; \boldsymbol{\mu}) &= -a(\mathbf{u}, \mathbf{u}, \boldsymbol{\phi}_k) + b(p, \boldsymbol{\phi}_k) - \mu_1 c(\mathbf{u}, \boldsymbol{\phi}_k) + d(T, \boldsymbol{\phi}_k) \ , && k = 1, \dots, N_u \\
R_{N_u + k}(\mathbf{u}, p, T; \boldsymbol{\mu}) &= -b(\psi_k, \mathbf{u}) \ , && k = 1, \dots, N_p \\
R_{N_u + N_p + k}(\mathbf{u}, p, T; \boldsymbol{\mu}) &= \mu_1 \mu_2 h(\xi_k) - e(T, \mathbf{u}, \xi_k) - \mu_1 \mu_2 g(T, \xi_k) \ , && k = 1, \dots, N_T
\end{aligned}
$$

Remark 2.1. *For high Gr and Pr numbers the Newton method might be insufficient. We propose in that case to use the continuation algorithm 1. Note however that, for the range of Grashof and Prandtl, we didn't need to apply this continuation method.*

Algorithm 1 Continuation strategy for high Gr and Pr numbers.

Fix parameters Gr and Pr
Fix minimal values of parameters $Gr_{min} = 1$ and $Pr_{min} = 10^{-2}$
Fix tolerance tol and max number of iteration n_{max} for Newton
Calculate number of intermediary parameters:
$N = \max\left\{1; \max\left\{\lceil \log(Gr/Gr_{min}) \rceil), \lceil (\log(Pr/Pr_{min}) \rceil\right\}\right\}$
for i=1:N **do**
Logarithmic scale for intermediary parameters:
$Gr(i) = exp\{\log(Gr_{min}) + i(\log(Gr/Gr_{min}))/N)$
$Pr(i) = exp\{\log(Pr_{min}) + i(\log(Pr/Pr_{min}))/N)$
Fix Newton initial guess $u^0 = u_{old}$
while $||R|| \geq tol$ **or** $n \leq n_{max}$ **do**
find u^n s.t. $J(u^{n-1})(u^n - u^{n-1}) = R(u^{n-1})$
end while
$u_{old} = u^n$
end for
return Solution $u = u^n$

3. Reduced basis formulation

Let us now investigate a reduced basis formulation to solve the heat convection problem introduced above. We first propose a general approach that can be applied to any quadratic problem affine in parameters. The technique is based on the idea to store the more information as possible in the reduced space. So, tensors are introduced and projected as well as matrices and vectors. After an overview on this method we deal with application to the heat convection equations.

3.1. Reduced Basis for a general quadratic problem

We now turn to the generalization of the RBM to a quadratic problem, in abstract form it reads

$$a(u,u,v;\boldsymbol{\mu}) + b(u,v;\boldsymbol{\mu}) = f(v;\boldsymbol{\mu}),\ \forall v \in V \tag{15}$$

where the solution is $u = u(\boldsymbol{\mu}) \in V$, $\boldsymbol{\mu} \in R^Q$ indicates Q scalars parameters, V is an Hilbert space defined on a domain $\Omega \subset \mathbb{R}^d$, with $d = 1,2,3$, $a : V \times V \times V \to \mathbb{R}$ is a tri-linear form, $b : V \times V \to \mathbb{R}$ is a bi-linear form, and $f : V \to \mathbb{R}$ is a linear operator.

We first introduce the FE approximation of (15) that leads to the RB approximation. The RB formulation of the general problem (15) applies then to (2).

Let us define a FE space $V_{\mathcal{N}} \equiv \text{span}\{v_1,\dots,v_{\mathcal{N}}\} \subset V$, and the approximated solution of (15) as

$$u(\boldsymbol{\mu}) \simeq \sum_{i=1}^{\mathcal{N}} u_i(\boldsymbol{\mu}) v_i$$

then the discrete formulation of (15) writes

$$\sum_{i,j=1}^{\mathcal{N}} u_i u_j a(v_i,v_j,v_k;\boldsymbol{\mu}) + \sum_{i=1}^{\mathcal{N}} u_i b(v_i,v_k;\boldsymbol{\mu}) = f(v_k;\boldsymbol{\mu}),\quad \forall k = 1,\dots,\mathcal{N} \tag{16}$$

where the solution $\mathbf{u} = \mathbf{u}(\boldsymbol{\mu}) = [u_1 \cdots u_{\mathcal{N}}]^T \in \mathbb{R}^{\mathcal{N}}$ for each parameter $\boldsymbol{\mu} \in \mathbb{R}^Q$.

In order to describe the RB approximation applied to problems such as (15), we need to define some discrete objects. In particular, we introduce the tensor $A = A(\boldsymbol{\mu}) \in \mathbb{R}^{\mathcal{N}\times\mathcal{N}\times\mathcal{N}}$, the matrix $B = B(\boldsymbol{\mu}) \in \mathbb{R}^{\mathcal{N}\times\mathcal{N}}$, and the vector $\mathbf{f} = \mathbf{f}(\boldsymbol{\mu}) \in \mathbb{R}^{\mathcal{N}}$ defined by

$$(A)_{ijk} = a(v_i,v_j,v_k;\boldsymbol{\mu}),\quad (B)_{ki} = b(v_i,v_k;\boldsymbol{\mu}),\quad (\mathbf{f})_k = f_k = f(v_k;\boldsymbol{\mu})$$

for $i,j,k = 1,\dots,\mathcal{N}$. Note that the tensor A can be considered as a vector of matrices, *i.e.* for each $k = 1,\dots,\mathcal{N}$, we define the matrix $A_k = A_k(\boldsymbol{\mu}) \in \mathbb{R}^{\mathcal{N}\times\mathcal{N}}$ as $(A_k)_{ij} = (A)_{ijk}$, $i,j = 1,\dots.\mathcal{N}$. Using this notation (16) now reads

$$\mathbf{u}^T A_k \mathbf{u} + (B\mathbf{u})_k = f_k,\ \forall k = 1,\dots,\mathcal{N} \tag{17}$$

where $(B\mathbf{u})_k$ and f_k are respectively the k-th term of vectors $B\mathbf{u}$ and $\mathbf{f}$.

As described in Section 2 we can treat the non-linearity of the first term in (17) using a Newton Method. For a given parameter $\boldsymbol{\mu} \in \mathbb{R}^Q$, each iteration $n = 1,2,\dots$ of the Newton algorithm reads

$$J_{row(k)}(\mathbf{u}^n;\boldsymbol{\mu})(\mathbf{u}^{n+1} - \mathbf{u}^n) = R_k(\mathbf{u}^n;\boldsymbol{\mu}),\ \forall k = 1,\dots,\mathcal{N} \tag{18}$$

where $J_{row(k)}(\mathbf{u}^n;\boldsymbol{\mu}) \in \mathbb{R}^{\mathcal{N}}$ and $R_k(\mathbf{u}^n;\boldsymbol{\mu})$ are respectively the k-th row of the Jacobian matrix $J = J(\mathbf{u};\boldsymbol{\mu}) \in \mathbb{R}^{\mathcal{N}\times\mathcal{N}}$ and the k-th term of the residual $R = R(\mathbf{u}^n;\boldsymbol{\mu}) \in \mathbb{R}^{\mathcal{N}}$ defined by

$$R_k(\mathbf{u}^n;\boldsymbol{\mu}) = f_k - (\mathbf{u}^n)^T A_k \mathbf{u}^n - (B\mathbf{u}^n)_k,\ k = 1,\dots,\mathcal{N}. \tag{19}$$

For the sake of notation, the dependency of a, b, f, and all associated operators on the parameters $\boldsymbol{\mu}$ is removed.

Now, we just need to calculate the Jacobian matrix $J = J(\mathbf{u}; \boldsymbol{\mu})$. We can write each of its elements as

$$\begin{aligned} J_{ki}(\mathbf{u}) &= \frac{\partial a}{\partial u_i}(\mathbf{u}, \mathbf{u}, v_k) + \frac{\partial b}{\partial u_i}(\mathbf{u}, v_k) - \frac{\partial f}{\partial u_i}(v_k) = \\ &= \sum_{l=1}^{\mathcal{N}} \sum_{j=1}^{\mathcal{N}} \frac{\partial}{\partial u_i}\big(u_l u_j a(v_l, v_j, v_k)\big) + \sum_{j=1}^{\mathcal{N}} \frac{\partial}{\partial u_i}\big(b(v_j, v_k) u_j\big) = \\ &= \sum_{l=1}^{\mathcal{N}} \sum_{j=1}^{\mathcal{N}} \frac{\partial}{\partial u_i}\big(u_l A_{ljk} u_j\big) + \sum_{j=1}^{\mathcal{N}} \frac{\partial}{\partial u_i}\big(B_{kj} u_j\big) = \\ &= \sum_{j=1}^{\mathcal{N}} u_j A_{jik} + \sum_{j=1}^{\mathcal{N}} A_{ijk} u_j + B_{ki}. \end{aligned} \tag{20}$$

In general A is never assembled as it is of size $\mathcal{N}^3$, where $\mathcal{N}$ is the dimension of the underlying discretization space. If this is indeed the case for FE discretization, recall however that here we deal with reduced order approximation which enables the computation of A explicitly. Let us then introduce a reduced space $V_N = \text{span}\{\varphi_1, \ldots, \varphi_N\}$, with $N \ll \mathcal{N}$ and define the projection $\widetilde{\mathbf{u}} = \widetilde{\mathbf{u}}(\boldsymbol{\mu}) \in \mathbb{R}^N$of the solution $\mathbf{u}$ in V_N as

$$\widetilde{\mathbf{u}} = \sum_{i=1}^{N} \widetilde{u}_i \varphi_i = \Phi^T \mathbf{u}, \tag{21}$$

where $\Phi = [\varphi_1 \ldots \varphi_N] \in \mathbb{R}^{\mathcal{N} \times N}$, $\Phi_{ji} = \hat{\varphi}_{i,j} = (\varphi_i, v_j)_{V_\mathcal{N}}$, the coefficients $\widetilde{u}_i$ are

$$\widetilde{u}_i = (u, \varphi_i)_{V_\mathcal{N}} = (u, \sum_{j=1}^{\mathcal{N}} \hat{\varphi}_{i,j} v_j)_{V_\mathcal{N}} = \sum_{j=1}^{\mathcal{N}} u_j \hat{\varphi}_{i,j}$$

and $(\cdot, \cdot)_{V_\mathcal{N}}$ is the scalar product associated to $V_\mathcal{N}$.

We observe that the Newton method defined in (18) is in fact generic with respect to the discrete spaces and that we can replace V_N by V_N which corresponds to projections of the terms in (18) onto V_N. We now prove this statement.

Following the procedure of (21) we start by defining the projection of the source term $\mathbf{f}$ as $\widetilde{\mathbf{f}} = \widetilde{\mathbf{f}}(\boldsymbol{\mu}) \in \mathbb{R}^N$

$$\widetilde{\mathbf{f}} = \Phi^T \mathbf{f}, \tag{22}$$

then the matrix $\widetilde{B} = \widetilde{B}(\boldsymbol{\mu}) \in \mathbb{R}^{N \times N}$ as well is the projection of B into the reduced space V_N

$$\widetilde{B} = \Phi^T B \Phi, \tag{23}$$

and finally we denote the reduced size tensor $\widetilde{A} = \widetilde{A}(\boldsymbol{\mu}) \in \mathbb{R}^{N \times N \times N}$. We now prove that it is in fact the projection of the tensor $A \in \mathbb{R}^{\mathcal{N} \times \mathcal{N} \times \mathcal{N}}$ in V_N

$$\begin{aligned} \widetilde{A}_{ijk} = a(\varphi_i, \varphi_j, \varphi_k) &= \sum_{l,m,h=1}^{\mathcal{N}} \hat{\varphi}_{i,l} \hat{\varphi}_{j,m} \hat{\varphi}_{k,h} a(v_l, v_m, v_h) = \\ = \sum_{l,m,h=1}^{\mathcal{N}} \hat{\varphi}_{i,l} \hat{\varphi}_{j,m} \hat{\varphi}_{k,h} A_{lmh} &= \sum_{h=1}^{\mathcal{N}} \hat{\varphi}_{k,h} (\Phi^T A_h \Phi)_{ij} = \sum_{h=1}^{\mathcal{N}} \hat{\varphi}_{k,h} (\widetilde{A}_h)_{ij} \end{aligned} \tag{24}$$

$i, j, k = 1, \ldots, N$, where $\widetilde{A}_k = \Phi^T A_k \Phi$ is the projection of $A_k \in \mathbb{R}^{\mathcal{N}\times\mathcal{N}}$ in V_N for all $k = 1, \ldots, \mathcal{N}$. Let us define the tensorial product $\star : \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}$ as

$$\mathbf{a}^T \star (\mathbf{b}^T A\mathbf{c}) := \sum_{h=1}^{\mathcal{N}} \mathbf{a}_h (\mathbf{b}^T A_h \mathbf{c}), \quad \forall \mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{R}^{\mathcal{N}}, \forall A \in \mathbb{R}^{\mathcal{N}\times\mathcal{N}\times\mathcal{N}}. \tag{25}$$

Then

$$\widetilde{A}_{ijk} = \sum_{h=1}^{\mathcal{N}} \hat{\varphi}_{k,h} (\varphi_i^T A_h \varphi_j) = \varphi_k^T \star (\varphi_i^T A \varphi_j), \quad i, j, k = 1, \ldots, N. \tag{26}$$

Using equations (23) and (26) we obtain the projection of the Jacobian $\widetilde{J} \in \mathbb{R}^{N\times N}$ at each iteration $n = 1, 2, \ldots$, and it reads

$$\widetilde{J}_{ik}(\widetilde{\mathbf{u}}^n; \boldsymbol{\mu}) = (\widetilde{A}_k|_{col(i)})^T \widetilde{\mathbf{u}}^n + (\widetilde{A}_k|_{row(i)}) \widetilde{\mathbf{u}}^n + \widetilde{B}_{ik} \tag{27}$$

where $\widetilde{A}_k|_{col(i)} = [\widetilde{A}_{jik}]_{j=1}^N \in \mathbb{R}^N$ is the i-th column of the matrix $\widetilde{A}_k$ for each $k = 1, \ldots, N$ and $\widetilde{A}_k|_{row(i)} = [\widetilde{A}_{ijk}]_{j=1}^N \in \mathbb{R}^N$ is the i-th row of the matrix $\widetilde{A}_k$ for each $k = 1, \ldots, N$. If we use the tensorial product defined by (25) we simply write

$$\widetilde{A}_k|_{col(i)} = \begin{bmatrix} \varphi_k^T \star (\varphi_1^T A \varphi_i) & \cdots & \varphi_k^T \star (\varphi_N^T A \varphi_i) \end{bmatrix}$$
$$\widetilde{A}_k|_{row(i)} = \begin{bmatrix} \varphi_k^T \star (\varphi_i^T A \varphi_1) & \cdots & \varphi_k^T \star (\varphi_i^T A \varphi_N) \end{bmatrix}$$

$i, k = 1, \ldots, N$. Similarly, each term of the reduced residual $\widetilde{R} = \widetilde{R}(\widetilde{\mathbf{u}}; \boldsymbol{\mu}) \in \mathbb{R}^N$ reads

$$\widetilde{R}_k(\widetilde{\mathbf{u}}^n; \boldsymbol{\mu}) = \widetilde{f}_k - (\widetilde{\mathbf{u}}^n)^T \widetilde{A}_k \widetilde{\mathbf{u}}^n - (\widetilde{B}\widetilde{\mathbf{u}}^n)_k, \quad k = 1, \ldots, N. \tag{28}$$

3.2. Application to heat convection problem

Let us now apply this reduction technique to the natural convection problem introduced in Section 2. The FE formulation of each sub-iteration $n = 1, \ldots, n_{max}$ of the Newton Method applied to equations (2) writes

$$J(\mathbf{u}^n, p^n, T^n) \left[(\mathbf{u}^{n+1}, p^{n+1}, T^{n+1}) - (\mathbf{u}^n, p^n, T^n) \right] = R(\mathbf{u}^n, p^n, T^n) \tag{29}$$

with Jacobian and residual terms defined as in Section 2. In order to apply the technique described in the previous paragraph, we introduce matrices and tensors associated to (10). We define the tensors $A \in \mathbb{R}^{N_u\times N_u\times N_u}$ and $E \in \mathbb{R}^{N_T\times N_u\times N_T}$ as

$$A = [A_{ijk},\ i, j, k = 1, \ldots, N_u], \quad (A_k)_{ij} = A_{ijk} = a(\boldsymbol{\phi}_i, \boldsymbol{\phi}_j, \boldsymbol{\phi}_k)$$
$$E = [E_{ijk},\ i, k = 1, \ldots, N_T, j = 1, \ldots, N_u], \quad (E_k)_{ij} = E_{ijk} = e(\xi_i, \boldsymbol{\phi}_j, \xi_k)$$

the matrices $B \in \mathbb{R}^{N_p\times N_u}$, $C \in \mathbb{R}^{N_u\times N_u}$, $D \in \mathbb{R}^{N_u\times N_T}$, $G \in \mathbb{R}^{N_T\times N_T}$ as

$$B = [B_{ij},\ i = 1, \ldots, N_u, j = 1, \ldots, N_p], \quad B_{ij} = b(\psi_j, \boldsymbol{\phi}_i)$$
$$C = [C_{ij},\ i, j = 1, \ldots, N_u], \quad C_{ij} = c(\boldsymbol{\phi}_j, \boldsymbol{\phi}_i)$$
$$D = [D_{ij},\ i = 1, \ldots, N_u, j = 1, \ldots, N_T], \quad D_{ij} = d(\xi_j, \boldsymbol{\phi}_i)$$
$$G = [G_{ij},\ i = 1, \ldots, N_T], \quad G_{ij} = g(\xi_j, \xi_i)$$

and the vector $H \in \mathbb{R}^{N_T}$ as

$$H = [H_i,\ i = 1, \ldots, N_T], \quad H_i = h(\xi_i).$$

Then, using notation introduced in Section 3.1, the Jacobian matrix writes

$$J(\mathbf{u},p,T;\boldsymbol{\mu}) = \begin{bmatrix} J^{Nl_1}(\mathbf{u}) + \mu_1 C & -B & -D \\ B^T & \mathbf{0} & \mathbf{0} \\ J^{Nl_2}(T) & \mathbf{0} & J^{Nl_3}(\mathbf{u}) + \mu_1\mu_2 G \end{bmatrix} \tag{30}$$

where each element of the non-linear submatrices $J^{Nl_1}(\mathbf{u})$, $J^{Nl_2}(T)$, $J^{Nl_3}(\mathbf{u})$ are

$$\begin{aligned} J^{Nl_1}_{ki}(\mathbf{u}) &= \sum_{j=1}^{N_u}((A_k)_{ij} + (A_k)_{ji})\mathbf{u}_j = ((A_k|_{col(i)})^T + (A_k|_{row(i)}))\mathbf{u}\ ,\ k,i = 1,\dots,N_u \\ J^{Nl_2}_{ki}(T) &= \sum_{j=1}^{N_T}(E_k)_{ij}T_j = (E_k|_{row(i)})T\ ,\ k = 1,\dots,N_T, i = 1,\dots,N_u \\ J^{Nl_3}_{ki}(\mathbf{u}) &= \sum_{j=1}^{N_u}(E_k)_{ji}\mathbf{u}_j = (E_k|_{col(i)})^T\mathbf{u}\ ,\ k,i = 1,\dots,N_T \end{aligned} \tag{31}$$

whereas each element of the residual vector of the Newton algorithm (29) writes

$$\begin{aligned} R_k(\mathbf{u},p,T;\boldsymbol{\mu}) &= -\,\mathbf{u}^T A_k\mathbf{u} + B_{row(k)}p - \mu_1 C_{row(k)}\mathbf{u} + D_{row(k)}T\ ,\ k = 1,\dots,N_u \\ R_k(\mathbf{u},p,T;\boldsymbol{\mu}) &= -\,(B_{col(k)})^T u\ ,\ k = N_u+1,\dots,N_u+N_p \\ R_k(\mathbf{u},p,T;\boldsymbol{\mu}) &= \mu_1\mu_2 H_k - \mathbf{u}^T E_k T - \mu_1\mu_2 G_{row(k)}T \end{aligned} \tag{32}$$

To apply efficiently the RB methodology, a key ingredient is the affine decomposition of the terms in the Newton Method which is readily available for our problem. The Jacobian matrix writes as

$$J(\mathbf{u},p,T;\boldsymbol{\mu}) = \sum_{q=1}^{Q_J}\theta_J^q(\boldsymbol{\mu})J^q(\mathbf{u},p,T) \tag{33}$$

where for the considered example $Q_J = 4$ and the coefficients θ_J are

$$\theta_J^1(\boldsymbol{\mu}) = \mu_1 = \frac{1}{\sqrt{Gr}},\quad \theta_J^2(\boldsymbol{\mu}) = \mu_1\mu_2 = \frac{1}{\sqrt{Gr}Pr},\quad \theta_J^3(\boldsymbol{\mu}) = \theta_J^4(\boldsymbol{\mu}) = 1. \tag{34}$$

Each sub-matrix can be described using the notation introduced above

$$J^1(\mathbf{u},p,T) = \begin{bmatrix} C & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix},\qquad J^2(\mathbf{u},p,T) = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & G \end{bmatrix}, \tag{35}$$

$$J^3(\mathbf{u},p,T) = \begin{bmatrix} \mathbf{0} & -B & -D \\ B^T & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix},\quad J^4(\mathbf{u},p,T) = \begin{bmatrix} J^{Nl_1}(\mathbf{u}) & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ J^{Nl_2}(T) & \mathbf{0} & J^{Nl_3}(\mathbf{u}) \end{bmatrix}.$$

As to the residual, it is readily decomposed as

$$R(\mathbf{u},p,T;\boldsymbol{\mu}) = \sum_{q=1}^{Q_R}\theta_R^q(\boldsymbol{\mu})R^q(\mathbf{u},p,T) \tag{36}$$

with $Q_R = 3$ terms, we have

$$\theta_R^1(\boldsymbol{\mu}) = \mu_1 = \frac{1}{\sqrt{Gr}}, \quad \theta_R^2(\boldsymbol{\mu}) = \mu_1\mu_2 = \frac{1}{\sqrt{Gr}Pr}, \quad \theta_R^3(\boldsymbol{\mu}) = 1 \tag{37}$$

and

$$R^1(\mathbf{u}, p, T) = \begin{bmatrix} -C\mathbf{u} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \; R^2(\mathbf{u}, p, T) = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ H - GT \end{bmatrix}, \tag{38}$$
$$R^3(\mathbf{u}, p, T) = \begin{bmatrix} -(\mathbf{u}^T A_k \mathbf{u})_{k=1}^{N_u} + Bp + DT \\ B^T\mathbf{u} \\ -((\mathbf{u}^T E_k T)^T)_{k=1}^{N_T} \end{bmatrix}.$$

The last step consists in projecting these equations in a reduced space. As shown for the general problem (15), the reduced Newton method at each iteration $n = 1, \ldots, n_{max}$ writes

$$\widetilde{J}(\widetilde{\mathbf{u}}^n, \widetilde{p}^n, \widetilde{T}^n) \left[(\widetilde{\mathbf{u}}^{n+1}, \widetilde{p}^{n+1}, \widetilde{T}^{n+1}) - (\widetilde{\mathbf{u}}^n, \widetilde{p}^n, \widetilde{T}^n) \right] = \widetilde{R}(\widetilde{\mathbf{u}}^n, \widetilde{p}^n, \widetilde{T}^n) \tag{39}$$

where the $\sim$ indicates the projection onto the reduced space V_N. The computation of vectors and matrices is done in a "classical" way, the terms are calculated in the FE space $V_{\mathcal{N}}$ and projected into the reduced one by the matrix Φ defined by the basis function. Each basis function is the solution of problem (10) for a fixed parameter $\boldsymbol{\mu}$, possibly orthonormalized. For tensors A and E a different approach is needed because of their high dimensions.

Let us consider a general tensor $A \in R^{\mathcal{N}^3}$ defined by a trilinear form $a : V_{\mathcal{N}}^3 \to \mathbb{R}$, $A_{ijk} = a(v_i, v_j, v_k)$, $i, j, k = 1, \ldots, \mathcal{N}$ as in problem (15). Its projection on a reduced space $V_N = \text{span}\{\varphi_1, \ldots, \varphi_N\}$, as shown in Section 3.1, is defined by $\widetilde{A} \in \mathbb{R}^{N^3}$, $\widetilde{A}_{ijk} = \varphi_k^T \star (\varphi_i^T A \varphi_j)$, $i, j, k = 1, \ldots, N$. We observe that $\widetilde{A}_{ijk} = a(\varphi_i, \varphi_j, \varphi_k)$ where $\varphi_i, \varphi_j, \varphi_k \in V_{\mathcal{N}}$.

Let us then define an hybrid tensor $\Lambda \in \mathbb{R}^{\mathcal{N}\times\mathcal{N}\times N}$ whose elements are

$$\Lambda_{ijk} = (\Lambda_k)_{ij} = a(v_i, v_j, \varphi_k), \tag{40}$$

$i, j = 1, \ldots, \mathcal{N}$, $k = 1, \ldots, N$. We can then use this tensor to redefine the reduced tensor $\widetilde{A}$

$$\widetilde{A}_{ijk} = a(\varphi_i, \varphi_j, \varphi_k) = \sum_{l,m=1}^{\mathcal{N}} \hat{\varphi}_{i,l}\hat{\varphi}_{j,m} a(v_l, v_m, \varphi_k) =$$
$$= \sum_{l,m=1}^{\mathcal{N}} \Phi_{li}\Phi_{mj}(\Lambda_k)_{lm} = (\Phi^T \Lambda_k \Phi)_{ij}. \tag{41}$$

So, we proved that

$$\widetilde{A}_k = \Phi^T \Lambda_k \Phi \; . \tag{42}$$

That implies that we can calculate only N matrices Λ_k, $k = 1, \ldots, N$, and project them into the reduced space to obtain the reduced tensor $\widetilde{A}$.

4. Computational framework

In this section we give an overview of the computational framework and its implementation to solve the heat convection problem introduced in section 2. All the routines described are part of the FE or RB frameworks

of Feel++ library and available in Feel++ source code. We first introduce the main principles of Feel++, then we illustrate the FE implementation of problem (1) and finally we introduce the associated RB framework.

4.1. Feel++ principles and design

Feel++ is a C++ library that provides a clear and easy to use interface to solve complex PDE systems. It aims at bringing the scientific community a tool for the implementation of advanced numerical methods and high performance computing.

Feel++ relies on a so-called *Domain Specific Embedded Language* (DSEL) designed to closely match the Galerkin mathematical framework. In computer science DS(E)Ls are used to partition complexity. In Feel++ the DSEL splits low level mathematics and computer science on one side, and high level mathematics as well as physical applications to the other side. This difference between disciplines is reflected on users and developers tasks and allows easily improvements on both sides. Furthermore, it enables using Feel++ for teaching purposes, solving complex problems with multiple physics and scales or rapid prototyping of new methods, schemes or algorithms.

The DSEL on Feel++ provides access to powerful tools such as interpolation, with a simple and seamless interface, and allows clear translation of a wide range of variational formulations into the variational embedded language. Combined with this robust engine, it lies also arbitrary order finite elements, high order quadrature formulas and robust nodal configuration sets. The tools at the user's disposal grant the flexibility to implement numerical methods that cover a large combination of choices from meshes, function spaces or quadrature points using the same integrated language and control at each stage of the solution process of the numerical approximations.

In this paper, we use recent developments which allow to operate on large-scale parallel infrastructures. The general strategy used is *parallel data* framework using MPI and thanks to DSEL the MPI communications are seamless to the user: (*i)* we start with automatic mesh partitioning using Gmsh [GR09] (Chaco/Metis) — adding information about ghost cells with communication between neighbor partition;— (*ii)* the Feel++ parallel data structures such as meshes, (elements of) function spaces — create a parallel degrees of freedom table with local and global views; — (*iii)* and finally we use the library PETSc [BBB+12b,BBB+12a,BGMS97] which provides access to a Krylov subspace solvers(KSP) coupled with PETSc preconditioners such as Block-Jacobi, ASM, GASM. A complete description of the Feel++ high performance framework is available in the thesis [Cha13].

Remark 4.2. *The last preconditioner is an additive variant of the Schwarz alternating method for the case of many subregions, see [SBG04]. For each sub-preconditioners (in the subdomains),* PETSc *allows to choose a wide range of sequential preconditioners such as LU, ILU, JACOBI, ML. Moreover, precondioner ASM or GASM can be used with or without an algebraic overlap. Other parallel preconditioners are available in* PETSc *but not used here. In particular we would like to mention the* MUMPS *direct parallel solver [ADL00]. We use it both as solver and preconditioner for iterative solves.* FieldSplit *preconditioners are also of notice for the applications we have: they allow to exploit the structure of block matrix.*

4.3. Finite element model for the reduced basis framework

In this section we briefly introduce the organization of the RB framework of Feel++ (see figure 2), and then we see how the user deals with the FE model, needed to interface with the RB framework.

4.3.1. *Reduced basis framework*

The offline/online strategy developed in the RBM is implemented in the class `CRBTrilinear`.

As the offline step of the method can be very expensive, scalar products resulting from the projection of matrices or vectors on the RB are saved in a database. To save objects in the database, we use the concept of "serialization" introduced by the set of libraries for the C++ programming language Boost.

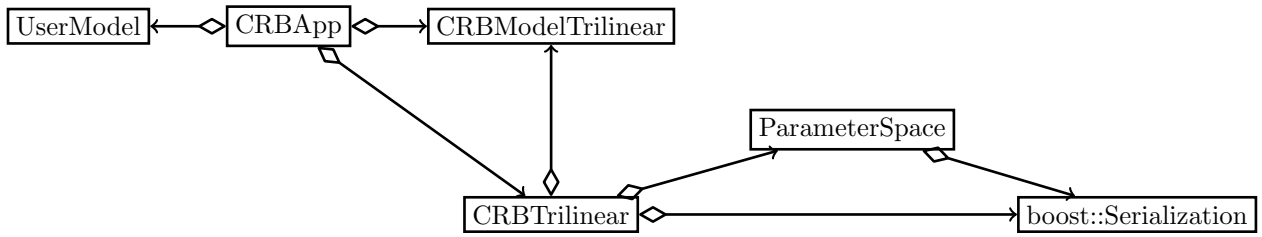


FIGURE 2. Class diagram for the FEEL++ RB framework. Arrows represent instantiations of template classes.

The class `CRBApp` is the driver for the RB framework. The class `ParameterSpace` manages the construction of the parameter space. First, if no existing database is found the offline step of RBM is run in order to build a basis, while if a database already exists the basis can be enriched or the algorithm passes to the online step.

To solve a linear or nonlinear problem with RB method in FEEL++ the user has to define in his model the variational formulation of the problem. At the state of the art, the variation form must be defined in accord with the affine decomposition, future revisions of the code will introduce an automated affine decomposition. Note that using FEEL++ the user model can be as well a 1D, 2D or 3D model. It is important to keep in mind that to interface with the RB framework the user has to provide only the FE model (*i.e.*parameter space, geometry, variational formulation), it corresponds to `UserModel` in figure 2 whose interface derives from `CRBModelTrilinear`.

The FEEL++ RB framework support parallel architectures using the MPI technology. As in the FE framework an automatic mesh partitioning using `Gmsh` (Chaco/Metis) is computed, while every data associated to the reduced basis (scalars, vectors and dense matrices, parameter space samplings,...) are duplicated on each processor. However note that since the mesh is partitioned according the number of processors, finite element approximations and thus the reduced basis functions are in fact spread on all processors. Currently the basis functions are saved in the RB database with their associated partitioning. If they are required for visualization purposes or reduced basis space enrichment, the same data partition as in the initial computations must be used. Another particular attention must be paid to parameter space sampling generation: we must ensure that all processors hold the same samplings. To this end, there are generated in a sequential way by only one processor and then broadcasted to other processors.

4.3.2. *Finite element model*

Let us illustrate the implementation of the FE solution strategy for the problem (1) using FEEL++, we give the main ideas of the solver function, with the continuation algorithm described above, and the computation of Newton method terms.

In listing 1, we display a snippet of code showing the code describing for a given μ, the solution process to retrieve $(\boldsymbol{u}(\mu), p(\mu), T(\mu))$: *(i)* compute the coefficients of the affine decomposition; *(ii)* assemble the linear terms; *(iii)* solve the non linear problem where `updateJacobian` and `updateResidual` are computing the jacobian and residual respectively during the Newton iterations. Note that `updateJacobian` need not be called at every iterations. Moreover the code is seamless with respect to geometrical dimension (2D or 3D) and parallel computing.

LISTING 1. Implementation of algorithm 1

```
void solve( parameter_type const& mu, element_ptrtype& T )
{
  this->computeThetaQ( mu ); // Update θ coefficients
  this->update( mu ); // Update affine decomposition of the linear terms
```

```
  // nonlinear iterative solver, solve for U=(u,p,T)
  backend()->nlSolve( _jacobian=updateJacobian,
                      _residual=updateResidual,
                      _solution=U );
}
```

As mentioned earlier, at each Newton iteration, the Jacobian matrix and the residual vector are updated. We start with the linear terms which can be precomputed, see section 3.2. We show, as an example, the Jacobian sub-matrices assembly. The vector `thetaAq[q]` represents the coefficients θ_J defined in (34), while the matrices `Aq[0]`, `Aq[1]`, `Aq[2]` are respectively the matrices J^1, J^2 and J^3 defined in (35). The assembly of those matrices is displayed in listing 2.

Listing 2. Jacobian Linear terms assembly

```
// Definition of mesh and FE space Xh = V_h × Q_h × Ξ_h
mesh_ptrtype mesh;
// N polynomial order
Vh = Pch<N+1, Vectorial>( mesh );
Qh = Pch<N>( mesh );
Xih = Pch<N+1>( mesh );
Xh = Vh * Qh * Xix;
// Definition of functions
element_type U( Xh, "u" );
element_type V( Xh, "v" );
// Velocity function and test function
element_0_type u = U. element<0>(); element_0_type v = V. element<0>();
// Pression function and test function
element_1_type p = U. element<1>(); element_1_type q = V. element<1>();
// Temperature function and test function
element_2_type t = U. element<2>(); element_2_type s = V. element<2>();

// 1) Fluid equations
// 1.1) Velocity diffusion: Aq[0] = C = ∫_Ω ∇u : ∇v
form2(_test=Xh,_trial=Xh,_matrix=Aq[0] ) =
    integrate( _range=elements(mesh), _expr=trace(gradt(u)*trans(grad(v))) );
// 1.2) Heat diffusion: Aq[1] = G = ∫_Ω ∇t · ∇s
form2( _test=Xh,_trial=Xh,_matrix=Aq[1] ) =
    integrate( _range=elements( mesh ), _expr=gradt(t)*trans(grad(s)) );
// 1.3) Pressure-velocity terms: Aq[2] = -B = ∫_Ω −p ∇ · v
form2( _test=Xh,_trial=Xh,_matrix=Aq[2] ) =
    integrate ( _range=elements( mesh ), _expr= - idt(p) * div(v) );
//      Aq[2] += B^t = ∫_Ω q ∇ · u
form2( _test=Xh,_trial=Xh,_matrix=Aq[2] ) +=
    integrate ( _range=elements( mesh ), _expr=divt(u) * id(q) );
// 2) Temperature equation
// 2.1) Buyoancy forces: Aq[2] += D = ∫_Ω t e_2 v
form2( _test=Xh,_trial=Xh,_matrix=Aq[2] ) +=
    integrate( elements( mesh ), -idt(t)*( trans(vec(cst(0.),cst(1.0)))*id(v) ) );
// B.C. ...
```

`form2(Xh,Xh,M)` builds a bilinear form $X_h \times X_h \to \mathbb{R}$ whose algebraic contribution will be stored in the matrix `M`. We remark that the finite element space `Xh` used to solve the problem is a composite space: `Vh` is a finite element space of degree 3 for vectorial functions (velocity), `Qh` is a scalar function space of degree 2 (pression) and `Xih` a scalar space of degree 3 (temperature). Again this is seamless for the user with respect to parallel computing and quite expressive with respect to the mathematical formulation.

Finally we add the contribution of the non-linear terms to the jacobian. The main operations required to this update are showed in listing 3. We remark that for a given $\boldsymbol{\mu}$ the linear part of the jacobian stored in `Jlin` does not need to be updated, its implementation is computed once for all using the code displayed in listing 2.

Listing 3. Non-linear terms assembly in jacobian

```
void Convection::updateJacobian( const vector_ptrtype& X, sparse_matrix_ptrtype& J)
{
    //Definition of mesh and FE space Xh, and definition of functions
    //are done in the same way as previously

    Aq[3][0]->zero(); // initialization

    // 1) Fluid equations - fluid convection derivatives:
    //     Aq[3] = u^T*A + A*u = ∫_Ω(u ∇·v_i v_j + v_i ∇·u v_j)
    form2( _test=Xh,_trial=Xh,_matrix=Aq[3] ) +=
            integrate ( _range=elements(mesh), trans( id(v) )*( gradv(u) )*idt(u) );
    form2( _test=Xh,_trial=Xh,_matrix=Aq[3] ) +=
            integrate ( _range=elements(mesh), trans( id(v) )*( gradt(u) )*idv(u) );

    // 2) Temperature equation - heat convection:
    //     Aq[3] += u^T*E + E*T = ∫_Ω(u·∇(s_i)s_j + u_i·∇(T)s_j)
    form2( _test=Xh,_trial=Xh,_matrix=Aq[3] ) +=
            integrate ( elements(mesh), grad(s)*( idv(t)*idt(u) ) );
    form2( _test=Xh,_trial=Xh,_matrix=Aq[3] ) +=
            integrate ( elements(mesh), grad(s)*( idt(t)*idv(u) ) );

    // B.C. ...

    // Jacobian = linear terms of Affine Decomposition + Nonlinear term Aq[3]
    J->zero(); J->addMatrix(1.,Jlin); J->addMatrix(1.,Aq[3]);
}
```

5. Numerical Experiments

We present some numerical results: first we compare flow profiles obtained using FEM and RBM, and associated errors, in both 2D and 3D cases. Then, computational times and performances varying model parameters are show in both FEM and CRB cases are displayed. Finally, we compare the Nusselt number obtained in both cases.

Thanks to the Feel++ framework, the finite element and reduced basis models are available both in 2D and 3D.

Regarding the 2D case the FEM simulations refer to 6×10^4 degrees of freedom while in the 3D tank we have more than 2×10^5 dof. We consider polynomial basis functions of degree 3 for velocity and temperature variables, and degree 2 for pressure variable. Continuation algorithm (see Algorithm 1) is not used for FEM simulation and RBM as well, a simple Newton Method without parameters continuation is run. The RB used for all results presented here contains 28 elements non orthonormalized. The basis functions are solutions of the FE problem evaluated for Prandtl fixed to 1 while Grashof varies from 1 to 10^6. Each 2D simulation is run in parallel on 10 processors, whereas each 3D simulation is run on 24 processors. In both cases we use the solver GMRES and the Additive Schwarz Method (GASM) as preconditioner.

5.1. **Flow profiles**

First we look at the solutions obtained with RBM and compare them with the FEM ones. We investigate the relative error between the two resolutions for increased turbulent flow, *i.e.* high parameters.

Figure 3 shows 2D results for different Grashof values (1, 10^3, 10^5). On the left there are RBM solutions, we show the velocity magnitude (top) and temperature (bottom) profiles. We observe faster flow for increased Grashof values as expected. On the right, relative errors for velocity (top) and temperature (bottom) are shown. As the flow is more turbulent for increasing parameters, the relative errors increase as well. Furthermore, errors is at least 2×10^{-3} for velocity magnitude and 10^{-2} for temperature profile, which are satisfactory values (we do not consider values for $Gr = 1$ because of it corresponds to a parameter used to build the basis, the error is as expected 10^{-16}).

We analyze the error in 2D and 3D simulations, respectively Figures 5(c) and 6(c), increasing Grashof values. We observe in both cases that the error increases for Grashof between 1 and 10^3 and it stabilizes for values greater than 10^3. Also, we remark that the solution in a finite 3D tank for high Grashof value represents a flow with a complex pattern (Figure 4).

5.2. **Performances**

We now compare the RBM computational gain with respect to FEM. Figures 5(b) and 6(b) show the computational times for increased Grashof in both FEM and RBM cases, respectively in the 2D and 3D tanks. In both cases, we show the log-log plot of the computational times (in seconds) vs Gr.

As expected the FEM solution costs is several order more expensive than the RBM one. In particular, we have a factor 6×10^2 for small Grashof and we reach a factor 1.5×10^3 for high Grashof in 2D case, and a factor from 10^3 to 10^4 occurs in 3D case. Furthermore, we observe that the computational time increases in parameters in the case of FEM, while it is almost constant in RBM computations. In both 2D and 3D case we find few parameters that lead to a higher computational time in RB, this is due to increased number of online Newton iterations, however it still yield good results when inspecting the Nusselt number, see Figure 5(a) and Figure 6(a).

We remark that we do not apply the continuation algorithm neither for FEM nor for RBM. However it may become necessary for FEM resolution in case of non-convergence of the Newton solver, for higher parameter values or more complex geometry. As initial guess of the Newton Algorithm used to solve the online RB problems, the nearest known solution is used, whereas in FEM the initial guess is taken as the zero — although we could also use the nearest basis function as initial guess. —

5.3. **Outputs**

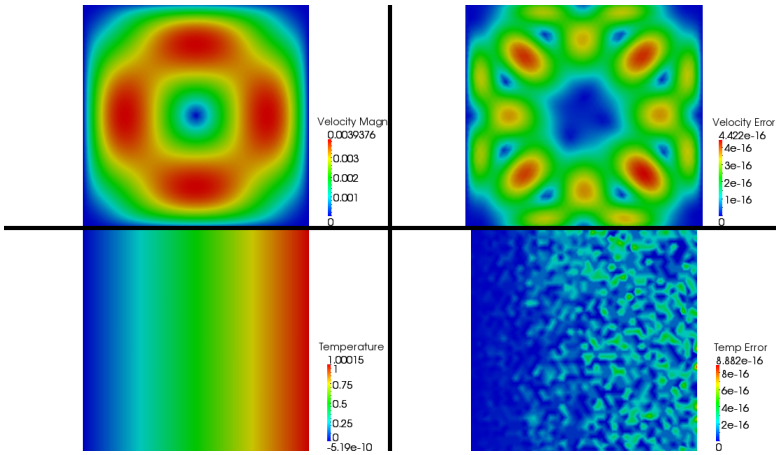
To conclude we look at the Nusselt number, *i.e.* the mean temperature on boundary $\Gamma_3 = \bar{\Omega} \cap \{x = 1\}$ (see Figure 1 for the 2D tank)

$$Nu = \int_{\Gamma_3} T d\sigma. \tag{43}$$

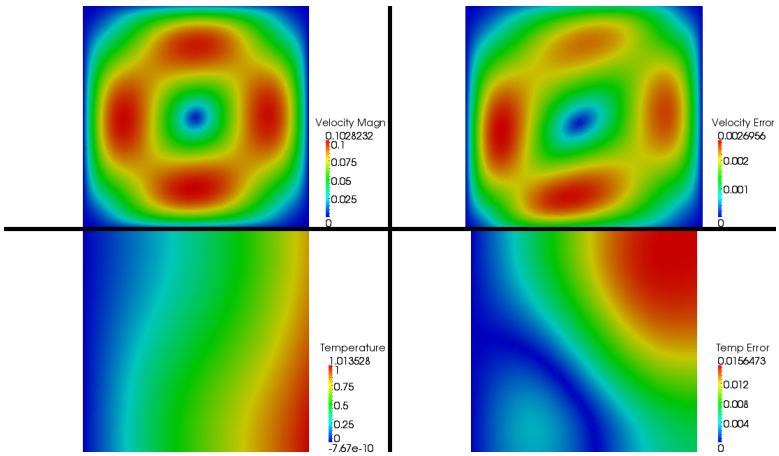
This quantity decreases with increasing Grashof because of faster fluid flows that remove the heat for Γ_3. In Figures 5(a) and 6(a), respectively 2D and 3D cases, the logarithmic output curves show that the RBM solutions follow the same behavior of the FEM ones with same Nu values. These results confirm the good approximation obtained with RBM in both 2D and 3D simulations.

6. Conclusions and perspectives

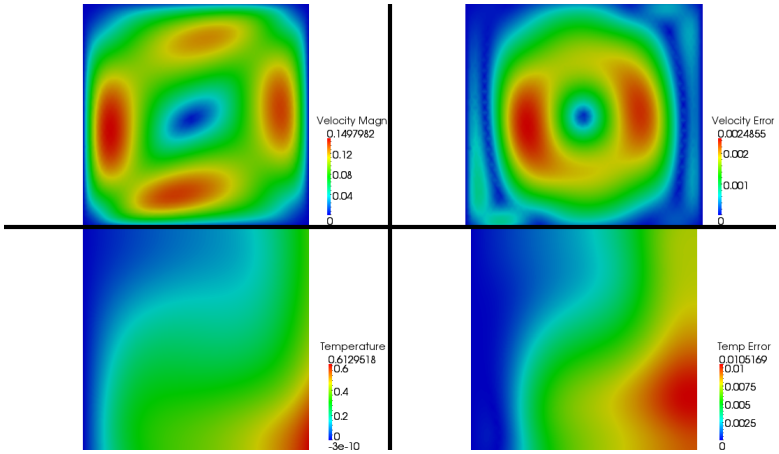
We have presented a mathematical and computational treatment of the RBM applied to steady-state natural convection. This can be readily applied to other types of problems with quadratic non-linearities. The proposed method gives results that are accurate and efficient. In particular, the efficiency of this technique is remarkable in the case of flows with complex patterns both in 2D and 3D. From a framework point of view, the (offline) database handling raises interesting challenges when dealing with large parallel data.



(a) Gr=1.e+00

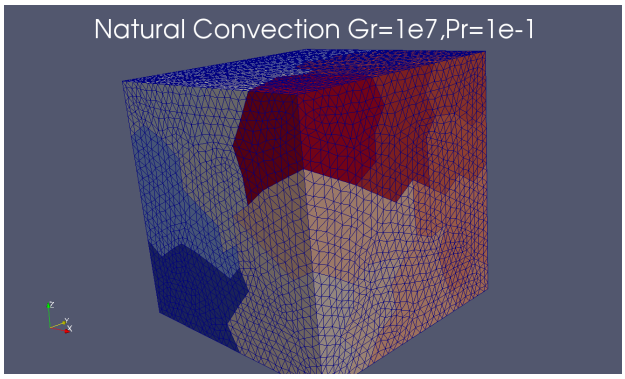


(b) Gr=1.e+03

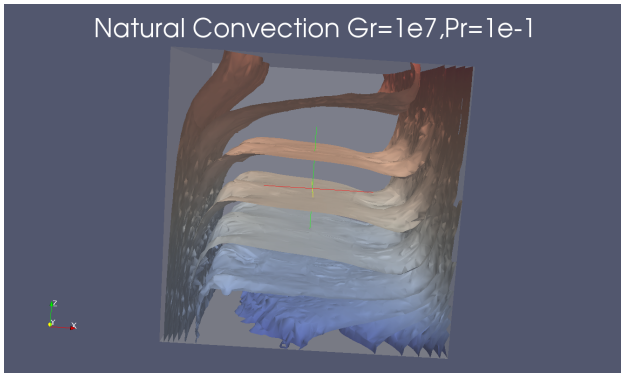


(c) Gr=1.e+05

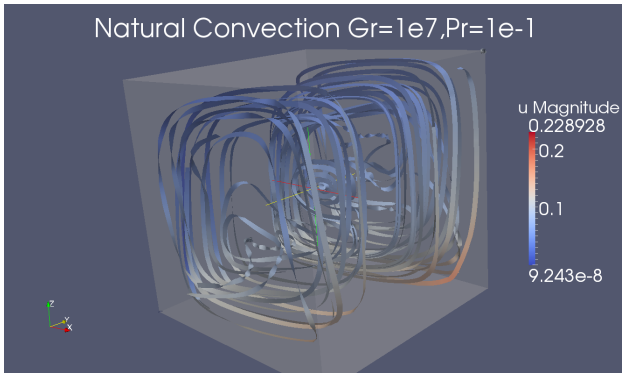
FIGURE 3. Comparisons of FEM and RBM solutions for $Gr = (1, 10^3, 10^5)$ and $Pr = 1$. On the left velocity flow magnitude (top) and temperature profile (bottom) for RBM solutions. On the right velocity error (top) and temperature error (bottom).



(a) mesh partitioning (24 processors)

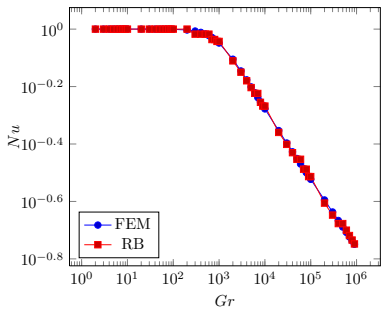


(b) temperature isosurfaces

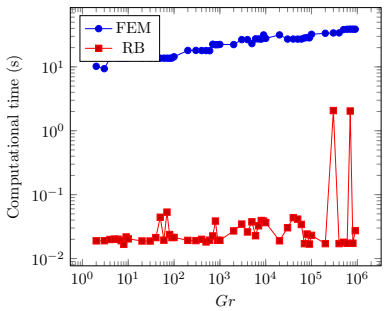


(c) Stream lines

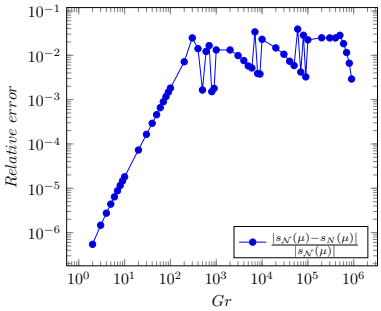
FIGURE 4. 3D computations for $Gr = 1e7, Pr = 0.1$



(a) Nu versus Gr



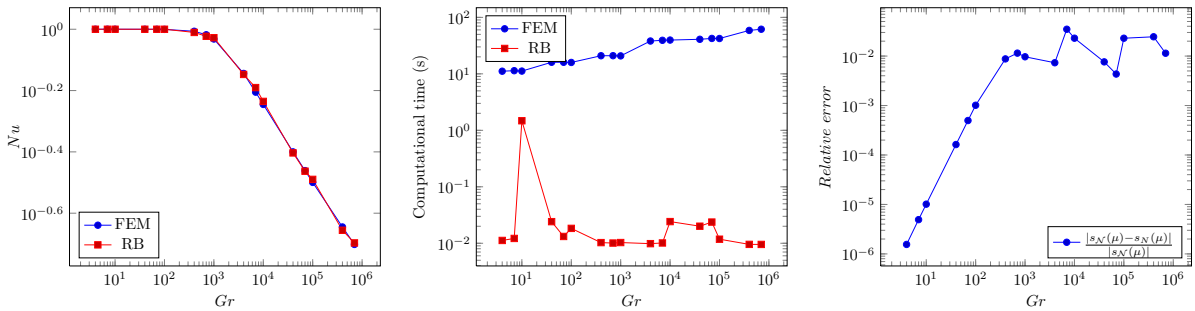
(b) Computational time versus Gr



(c) Error versus Gr

FIGURE 5. 2D computations for $Gr \in [1; 1e6]$, $Pr = 1$.

In terms of perspectives, a posteriori error estimation is a first step not only to assess the quality of the RB approximation but also to guide the RB space construction using greedy strategies [Yan12, VPRP03]. More complex applications can be considered in particular including geometrical parameters. However it is foreseen that we may require hp-RBM approximations, see *e.g.* [EHKP12]. Finally, we dealt with the steady-state of



(a) Nu versus Gr (b) Computational time versus Gr (c) Error versus Gr

FIGURE 6. 3D computations for $Gr \in [1; 1e6]$, $Pr = 1$.

the natural convection flow, the transient state is also of interest but requires much more involved mathematical and computational framework, see *e.g.* [Yan12].

Acknowledgements

The authors wish to thank Vincent Chabannes, Ranine Tarabay and Céline Caldini-Queiros for fruitful discussions during the Cemracs 2012. Stéphane Veys and Christophe Prud'homme acknowledge the financial support of the project ANR HAMM ANR-2010-COSI-009. Finally the authors wish to thank the Cemracs 2012 and its organizers.

References

[ADL00] P.R. Amestoy, I.S. Duff, and J.-Y. L'Excellent. Multifrontal parallel distributed symmetric and unsymmetric solvers. *Computer Methods in Applied Mechanics and Engineering*, 184(2-4):501 – 520, 2000.

[BBB+12a] S. Balay, J. Brown, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. Curfman McInnes, B. F. Smith, and H. Zhang. *PETSc Users Manual*, 2012.

[BBB+12b] S. Balay, J. Brown, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. Curfman McInnes, B. F. Smith, and H. Zhang. *PETSc Web page*, 2012.

[BGMS97] S. Balay, W. D. Gropp, L. Curfman McInnes, and B. F. Smith. *Efficient Management of Parallelism in Object Oriented Numerical Software Libraries*, 1997.

[Cha13] V. Chabannes. *Vers la simulation des écoulements sanguins*. PhD thesis, Université Joseph Fourier, Grenoble, 2013.

[DVTP13] C. Daversin, S. Veys, c. Trophime, and C. Prud'Homme. A reduced basis framework : Application to nonlinear multiphysics problem. Submitted to ESAIM proceedings, 2013.

[EHKP12] J.L. Eftang, D.B.P. Huynh, D.J. Knezevic, and A.T. Patera. A two-step certified reduced basis method. *Journal of Scientific Computing*, 51:28–58, 2012.

[GR09] C. Geuzaine and J.-F. Remacle. *Gmsh: a three-dimensional finite element mesh generator with built-in pre-and post-processing facilities*, 2009.

[PCD+12] C. Prud'Homme, V. Chabannes, V. Doyeux, M. Ismail, A. Samake, and G. Pena. Feel++: A computational framework for galerkin methods and advanced numerical methods. In *ESAIM: PROCEEDINGS*, pages 1–10, 2012.

[PP04] C. Prud'homme and A. T. Patera. Reduced-basis output bounds for approximately parameterized elliptic coercive partial differential equations. *Computing and Visualization in Science*, 6(2-3):147–162, 2004.

[Pru06] C. Prud'homme. A domain specific embedded language in C++ for automatic differentiation, projection, integration and variational formulations. Scientific Programming, 14(2):81-110, 2006.

[PRV+02] C. Prud'homme, D. V. Rovas, K. Veroy, L. Machiels, Y. Maday, A. T. Patera, and G. Turinici. Reliable real-time solution of parametrized partial differential equations: Reduced-basis output bound methods. *Journal of Fluids Engineering*, 124(1):70–80, 2002.

[QRM11] A. Quarteroni, G. Rozza, and A. Manzoni. Certified reduced basis approximation for parametrized partial differential equations and applications. *Journal of Mathematics in Industry*, 1(1):1–49, 2011.

[RHP07] G. Rozza, D.B.P. Huynh, and A.T. Patera. Reduced basis approximation and a posteriori error estimation for affinely parametrized elliptic coercive partial differential equations. *Archives of Computational Methods in Engineering*, 15(3):1–47, 2007.

[SBG04] B. Smith, P. Bjorstad, and W. Gropp. *Domain decomposition: parallel multilevel methods for elliptic partial differential equations*. Cambridge University Press, 2004.

[VCD+12] S. Veys, R. Chakir, C. Daversin, C. Prud'homme, and C. Trophime. A computational framework for certified reduced basis methods: application to a multiphysic problem. In *Conference Record of the 6th European Congress on Computational Methods in Applied Sciences and Engineering*, 2012.

[VDPT12] S. Veys, C. Daversin, C. Prud'homme, and C. Trophime. Reduced order modeling of high magnetic field magnets. In *Conference Record of the 11th International Workshop on Finite Elements for Microwave Engineering*, 2012.

[VPP03] K. Veroy, C. Prud'homme, and A. T. Patera. Reduced-basis approximation of the viscous Burgers equation: Rigorous *a posteriori* error bounds. *C. R. Acad. Sci. Paris, Série I*, 337(9):619–624, November 2003.

[VPRP03] K. Veroy, C. Prud'homme, D. V. Rovas, and A. T. Patera. *A posteriori* error bounds for reduced-basis approximation of parametrized noncoercive and nonlinear elliptic partial differential equations (AIAA Paper 2003-3847). In *Proceedings of the 16th AIAA Computational Fluid Dynamics Conference*, June 2003.

[Yan12] Masayuki Yano. A space-time petrov-galerkin certified reduced basis method : Application to the boussinesq equations. Submitted to SIAM Journal on Scientific Computing, December 2012.