Département de Mathématiques & Modélisation

Research Project

Reduced Basis Methods for Parametrized PDEs

June 6, 2020

Students:

Pierre-Jean	Bénard	benard@etud.insa-toulouse.fr
Landry	Duguet	duguet@etud.insa-toulouse.fr
Simon	Treillou	treillou@etud.insa-toulouse.fr

Tutor:

Pascal Noble *Mathematics professor* pascal.noble@math.univ-toulouse.fr

Keywords: Reduced basis method; Burgers equation; PDE; Greedy algorithm; POD

Abstract:

Partial differential equations (PDEs) are now one of the most used mathematical tools which allow us to model real world and physical phenomena. A physical problem can be parametrized depending on materials or environment for instance. Optimizing these parameters leads to select best materials or conditions for some purpose. One of the major issues of this method is computation cost; it needs to solve the same problem for different parameters a great number of times, rendering it unpractical for real time solutions. To reduce optimization cost, one can develop a method to learn solving faster a parametrized PDE. This is the purpose of the reduced basis method (RBM). Our goal was to test reduced basis methods especially for Burgers' equation in the fluid mechanics domain. To do so, we first developed the method for heat transfer equations to compare Greedy algorithm and proper orthogonal decomposition (POD), two common algorithms for basis selection. We also applied the method to the Cole-Hopf transform of Burgers' equation; this transformation already provided a major reduction in computation time. Our results show the robustness of the reduced basis methods and illustrate theoretical drawback and advantages of POD and Greedy algorithm. Further research should focus on stabilising the Burgers implementation and reducing time cost with machine learning or parallel computation.

Contents

1 Introduction and method explanation

1.1 Scientific context

PDEs involving multiple parameters are essential for resolving many problems in scientific and engineering fields including acoustics, elasticity, and finance. In order to optimize these parameters or to compute the solutions in real-time, "truth" solutions cannot be used because of their computation costs. This is why there is a need to create reduced basis methods which provide quick resolution times, with a certified and small error.

1.2 Overview

In this paper, we shall study how the reduced basis method works through two main parts. First, in order to use a reduced basis, we compare the two kinds of algorithms used in this case, which are the Proper Orthogonal Decomposition (POD) and the Greedy algorithm. We explain how these work, analyze their relative advantages and disadvantages such as convergence rate, computation time etc. All of this will be done by using a heat equation as a ground base. Secondly, we implement the RBM on a nonlinear PDE and with no exact solution under a simple form in order to demonstrate that the RBM is applicable to real-life problems. When converted by the Cole-Hopf transform, we obtain a linear PDE, thus implementing the RBM becomes easier. Finally, we discuss any future ideas and/or possible ways to pursue the implementation of the reduced basis method.

1.3 Method steps

To familiarize the reader with the RBM, the main steps of the method are described as follows:
Offline:

- Discretize parameter space;
- Generate reduced basis with POD or Greedy algorithm for instance;
- Precompute quantities from the affine assumption (as described in 2.1.6).

Online:

- Assemble operators from precomputed quantities for a given parameter;
- Solve (quickly) the reduced problem.

The most important aspect of the method we must keep in mind is that we want to certify the error. The basis choice is therefore a compromise between reduction for faster computation and error certification.

1.4 Libraries and coding methods

All algorithms and tests were implemented using Python on Jupyter Notebook (`https://jupyter.org/`). We used some Python libraries, listed as follows:

- NumPy (`https://numpy.org/`);
- SciPy (`https://www.scipy.org/`);
- FEniCS, an open-source computing platform for solving partial differential equations. One can refer to [1]. As we did, one can learn through FEniCS tutorials [4]. (`https://fenicsproject.org/`).

We also used time indicators to check for time efficiency:

- `timeit`, Jupyter Notebook function providing wall time with accuracy using loops;
- `time`, Jupyter Notebook function providing wall time but without loops.

In each part, we made all numerical measures on the same computer thus all comparisons are coherent, as we do not compare sections among themselves. Norms used for comparisons will be detailed in the appropriate sections.

2 Compare algorithms with the heat equation

In order to compare the two main algorithms, we analyse them on a heat equation, as used in [2]. We present the problem and prove quickly its well-posedness, then we implement the reduced basis method with POD and Greedy algorithm for this problem, before comparing both.

2.1 Problem presentation

2.1.1 Physical description

The problem we consider is a steady heat conduction in a two-dimensional domain $\Omega = [0,1] \times [0,1]$. We split the boundary $\partial\Omega$ into four parts as follows: $\Gamma_{bottom} = [0,1] \times \{0\}$, $\Gamma_{top} = [0,1] \times \{1\}$ and $\Gamma_{sides} = \{0\} \times [0,1] \bigcup \{1\} \times [0,1]$. Then, we split the domain Ω in two parts: Ω_0 which is a disk centered at the middle of Ω, with radius $r = 0.3$, and $\Omega_1 = \Omega \setminus \overline{\Omega_0}$ the rest of the domain. The geometrical set-up of this domain is represented in Fig. 1. The thermal conductivity is defined as $\kappa = \kappa_0 \mathbb{1}_{\Omega_0} + \mathbb{1}_{\Omega_1}$, with κ_0 constant and $\mathbb{1}$ the characteristic function of Ω sub-domains.

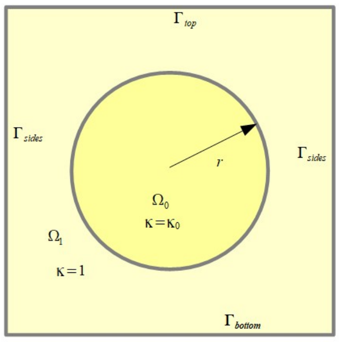

Figure 1: Geometrical representation of heat equation problem domains

The variable κ is significant because it allows us to introduce parameters and so to consider a space parameter $\mathbf{P} = [\mu_1^{min}, \mu_1^{max}] \times [\mu_2^{min}, \mu_2^{max}]$. We can define parameters as follows: $\kappa_0 = \mu_1$ and μ_2 the constant heat flux over Γ_{base}. So we have $\kappa_\mu = \mathbb{1}_{\Omega_1} + \mu_1 \mathbb{1}\Omega_0$.

2.1.2 Strong formulation

Thus we have $u(\mu)$ the scalar field variable satisfying Poisson's equation in Ω, with the Dirichlet condition on Γ_{top} (temperature at 0), the Neumann condition on Γ_{sides} (no flux on the sides) and the parametrized Neumann condition on Γ_{bottom} (a μ_2 flux). We can write the strong formulation of this parametrized problem as:

For some parameter μ in $\mathbf{P}$, find $u(\mu)$ in $\mathbf{V}$ such that :

$$\begin{cases} \nabla.\kappa_\mu \nabla u(\mu) & = 0 \quad in \ \ \Gamma \\ u(\mu) & = 0 \quad on \ \ \Gamma_{top} \\ \kappa_\mu \nabla u(\mu).n & = 0 \quad on \ \ \Gamma_{sides} \\ \kappa_\mu \nabla u(\mu).n & = \mu_2 \quad on \ \ \Gamma_{bottom} \end{cases}$$

with $\mathbf{V} = \{v \in H^1(\Omega) | v_{|\Gamma_{top}} = 0\}$ the space where solutions exists.

2.1.3 Weak formulation

From this strong formulation, we can find the weak formulation which reads as follows:

For some parameter $\mu \in \mathbf{P}$, find $u(\mu) \in \mathbf{V}$ such that:

$$a(u(\mu), v; \mu) = f(v; \mu), \quad v \in \mathbf{V}$$

with $a(v, w; \mu) = \int_\Omega \kappa_\mu \nabla w . \nabla v$ and $f(w; \mu) = \mu_2 \int_{\Gamma_{bottom}} v$, for all $v, w \in \mathbf{V}$.

We assume that $\mu_1^{min} > 0$ so $\kappa = min(1, \mu_1) > 0$. In this case, coercivity of the bilinear form a can be proved. As for the continuity and the linearity of forms a and f, they can be obtained without major problems, using the Cauchy-Schwarz inequality. Once these properties have been proved, we use the Lax-Milgram theorem to demonstrate the well-posedness of the problem: for one parameter $\mu = (\mu_1, \mu_2)$ given, there exists a unique solution $u(\mu)$ whose existence and unicity are guaranteed.

We chose the parameter space as

$$\mathbf{P} = [1, 10] \times [-1, 1]$$

2.1.4 Discretization

We can proceed to discretization, using $\mathbf{V}_\delta \subset \mathbf{V}$, with $\mathbf{V}_\delta$ a finite-dimensional space, here a standard finite elements space. We have the discrete weak formulation of the problem:

For some parameter $\mu \in \mathbf{P}$, find $u_\delta(\mu) \in \mathbf{V}_\delta$ such that:

$$a(u_\delta(\mu), v_\delta; \mu) = f(v_\delta; \mu), \quad v_\delta \in \mathbf{V}$$

2.1.5 Some graph outputs

For the rest of the discussion, we will assume that this finite-elements solution is the truth solution, with 2500 nodes. Some representative solutions of this example are represented in Fig. 2, 3, 4 and 5.

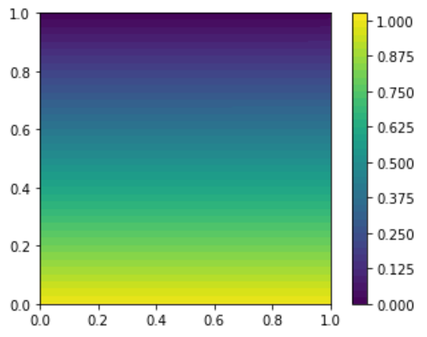

Figure 2: Heat diffusion for $(\mu_1, \mu_2) = (1, 1)$

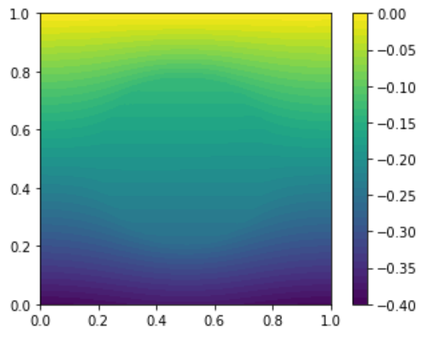

Figure 4: Heat diffusion for $(\mu_1, \mu_2) = (3, 0.5)$

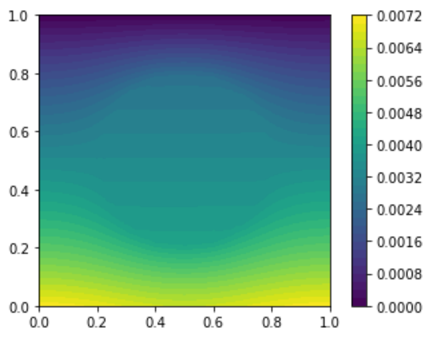

Figure 3: Heat diffusion for $(\mu_1, \mu_2) = (6, 0.01)$

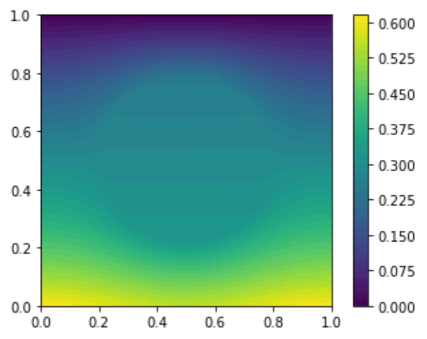

Figure 5: Heat diffusion for $(\mu_1, \mu_2) = (10, 0.0)$

2.1.6 Affine assumption

In order to have an efficient reduced basis method, we have to use the affine assumption. Indeed, without this assumption, we would have to build the reduced basis solution matrix $\mathbf{A}^{\mu}_{rb}$ for each μ as:

$$\mathbf{A}^{\mu}_{rb} = B^T \mathbf{A}^{\mu}_{\delta} B$$

with $\mathbf{B}$ the matrix interpolating space $\mathbf{V}_{\delta}$ to space $\mathbf{V}_{rb}$. Doing this, the computation would depend on N_{δ} and severely limit the interest of the reduced basis method. That is why we assume that the problem is affine refering to the parameters. As in [2], we define:

$$a(w, v; \mu) = \sum_{q=1}^{Q_a} \theta_a^q(\mu) a_q(w, v)$$

$$f(v; \mu) = \sum_{q=1}^{Q_f} \theta_{\mathrm{f}}^q(\mu) f_q(v)$$

where each form

$$a_q : \mathbb{V} \times \mathbb{V} \to \mathbb{R}, \quad f_q : \mathbb{V} \to \mathbb{R}$$

is independent of the parameter value μ and the coefficients

$$\theta_{\mathrm{a}}^q : \mathbb{P} \to \mathbb{R}, \quad \theta_{\mathrm{f}}^q : \mathbb{P} \to \mathbb{R}$$

and thus compute the matrix $\mathbf{A}^{\mu}_{rb}$ in an affine way.

Now that we have presented the heat equation that will be used, we move on to presenting the two algorithms.

2.2 POD

2.2.1 Method explanation

In the POD method, we compute all solutions for a chosen discretized parameter space, then perform an analysis of these solutions to retain only the most relevant information. For a given reduced space dimension, this is the best possible basis in terms of space approximation. Furthermore, this method in some particular cases is just a simple Singular Value Decomposition (SVD), an already optimized method.

2.2.2 Some results

As an example, we computed a reduced basis with POD method for this problem. We used a mesh composed of 2601 nodes (51x51) to have the highest accuracy. As for the parameters, we use a training set of length 100. We present results, one with parameters among evaluated parameters, another with parameters outside $\mathbf{P}$. Our reduced basis only contains 2 vectors as we built it.

For $(\mu_1, \mu_2) = (6, 0.1)$, we have:

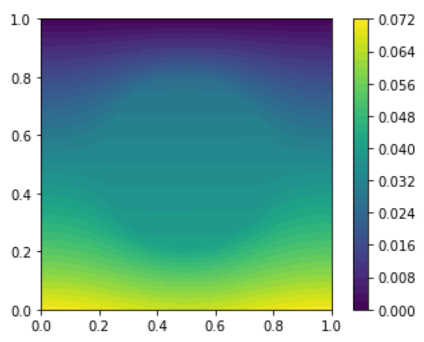

Figure 6: Reduced basis solution for $(\mu_1, \mu_2) = (6, 0.1)$

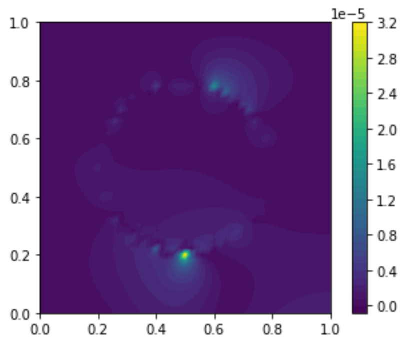

Figure 7: Absolute difference with truth solution for $(\mu_1, \mu_2) = (6, 0.01)$

In this example, displayed in Fig. 6 and 7, the truth solver took 1.2 s ± 163 ms to compute the solution, whereas the reduced basis solver took 35.3 μs ± 967 ns. The relative error, calculated as $\frac{U_\delta - U_{rb}}{U_\delta}$ is 4.017230065624981e-05. This results shows us how powerful the reduced basis method is. Nevertheless, this result is just an example, more data will be provided after.

For $(\mu_1, \mu_2) = (30, 3)$, we have:

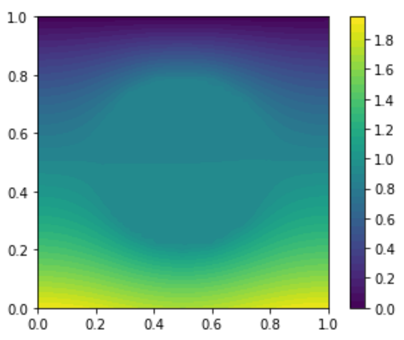

Figure 8: Reduced basis solution for $(\mu_1, \mu_2) = (30, 3)$

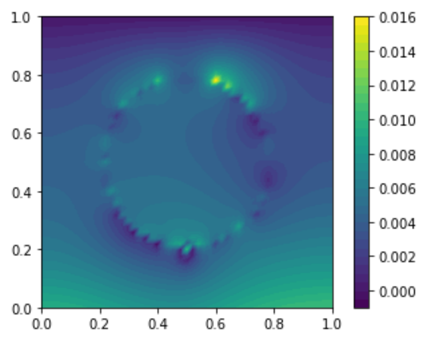

Figure 9: Absolute difference with truth solution for $(\mu_1, \mu_2) = (30, 3)$

In Fig. 8 and 9, the truth solver took 1.01 s ± 46.6 ms to compute the solution, whereas the reduced basis solver took 37.9 μs ± 1.62 μs. The relative error, calculated as $\frac{U_\delta - U_{rb}}{U_\delta}$ is approximately 0.0048, which is 100 times higher than previously. This is due to the training set used to build the reduced basis. Indeed, here we are evaluating parameters which were not evaluated for the basis creation. It remains very accurate nevertheless.

2.3 Greedy algorithm

Used in various areas, the Greedy algorithm is an iterative method based on local choices. In this section we present its functioning and the results we obtained with this algorithm.

2.3.1 Method explanation

The Greedy algorithm, contrary to the POD, does not search for the best basis to approximate the solution space, but only for a satisfactory one which will require less computation time. We present its implementation below.

Algorithm 1: Greedy algorithm

```
Data: A parameter space $\mathbb{P}$, $\mu$, tol
Result: A reduced basis $\mathbb{V}_\delta$
Compute $u_\delta(\mu)$ for all $\mu \in \mathbb{P}$ ;
Take $\mu_0$, a parameter taken at random in $\mathbb{P}$ ;
Initialize $\mathbb{V}_\delta = span(u_\delta(\mu_0))$ ;
Set the error $err = 10^{10}$ ;
while $err > tol$ do
    for $\mu \in \mathbb{P}$ do
        Compute $A^\mu_{rb}$ and $f^\mu_{rb}$ ;
        Solve $A^\mu_{rb} \times u^\mu_{rb} = f^\mu_{rb}$ ;
    end
    Find $\mu^* = argmax_{\mu \in \mathbb{P}} \eta(u^\mu_{rb})$ ;
    Add to $\mathbb{V}_\delta$ the normalized vector $u^{\mu^*}_{rb}$ ;
    Update $err = \max_{\mu \in \mathbb{P}} \eta(u^\mu_{rb})$ ;
end
```

2.3.2 Some results

As for the POD method, we computed a reduced basis with the Greedy algorithm for the heat equation shown in 2.1. We used a mesh composed of 2601 nodes (51x51) to have the best accuracy. As for the parameters, we used a training set of length 100. The results are shown below, one with parameters among evaluated parameters, another with parameters outside **P**. Our reduced basis only contains 2 vectors as we built it.

For $(\mu_1, \mu_2) = (6, 0.1)$, we have:

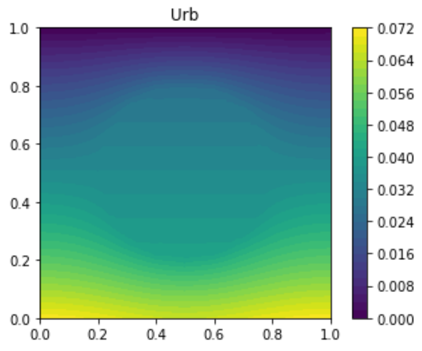

Figure 10: Reduced basis solution for $(\mu_1, \mu_2) = (6, 0.1)$

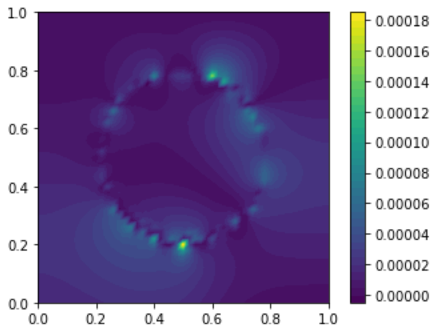

Figure 11: Absolute difference with truth solution for $(\mu_1, \mu_2) = (6, 0.01)$

In this example, displayed in Fig. 10 and 11, the truth solver took 1.26 s ± 50.8 ms to compute the solution, whereas the reduced basis solver only took 38.9 μs ± 709 ns. The relative error, calculated as $\frac{U_\delta - U_{rb}}{U_\delta}$ is approximately 0.00049.

For $(\mu_1, \mu_2) = (30, 3)$, we have:

In Fig. 12 and 13, the truth solver took 1.02 s ± 7.33 ms to compute the solution, whereas the reduced basis solver took 41.9 μs ± 737 ns. The relative error, calculated as $\frac{U_\delta - U_{rb}}{U_\delta}$ is approximately of 0.002. We observe that the Greedy algorithm is less subject to differences with the initial training set.

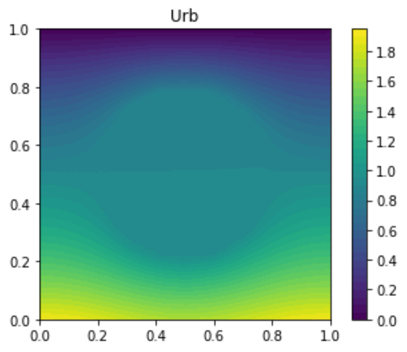

Figure 12: Reduced basis solution for $(\mu_1, \mu_2) = (30, 3)$

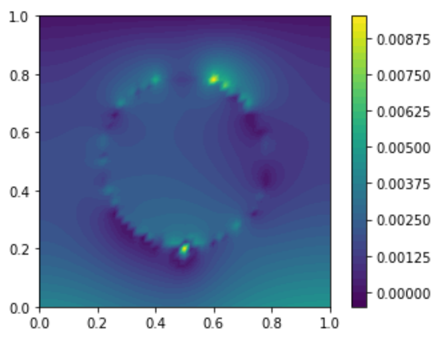

Figure 13: Absolute difference with truth solution for $(\mu_1, \mu_2) = (30, 3)$

2.4 Results and comparison

We saw how powerful these algorithms can be, now in this section, we will concentrate on comparing both, regarding convergence and computation time mainly, and see that training set size has an influence on performance.

2.4.1 Error calculation

To compute errors in the following sections, we use the following NumPy functions `numpy.mean` and `numpy.max` of 100 evaluations of $u_{rb}(\mu)$ for $\mu \in \mathbf{P}_h = [1, 10] \times [-1, 1]$. These functions are:

$$\max_{\mu \in \mathbf{P}_h} \|u_\delta(\mu) - u_{rb}(\mu)\| \quad \text{and} \quad \frac{1}{|\mathbf{P}_h|} \sum_{\mu \in P_h} \|u_\delta(\mu) - u_{rb}(\mu)\|$$

2.4.2 Convergence rate

Using a training set of size 100, we computed reduced basis of different sizes (from unique vector to 6), and then evaluated online error as described above.

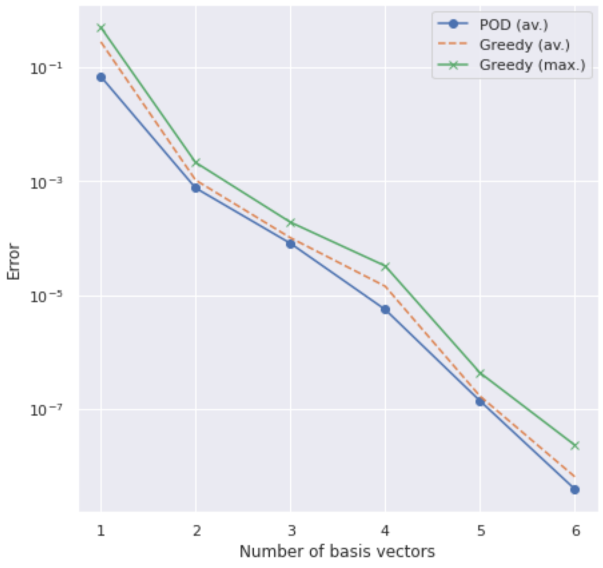

Figure 14: Errors comparison between Greedy algorithm and POD with respect to basis number of vectors

As expected, the POD provides the more accurate reduced basis, and both algorithms converge with respect to basis number of vectors.

On average, the mean online error is 2 times higher for the Greedy algorithm than for the POD method. Thus, we have to consider they are very small and remain satisfactory for our problem. We see from the output that for only one basis vector more, the Greedy algorithm can be more accurate than POD. In this way, we examine if using the Greedy algorithm with $n+1$ basis vectors is more time efficient than using POD with n basis vectors. Here, for our example, taking one more basis vector does not have an impact on computation time as we have a small number of vectors. However, this should be investigated with high dimensional reduced basis.

2.4.3 Computation time

One of the main advantages of the Greedy algorithm is a low-computation time for the reduced basis creation. Nevertheless, we cannot illustrate this because of differences in implementation. Indeed, the POD method was implemented using optimized SVD algorithms, from libraries, meanwhile our Greedy algorithm only uses basic **Numpy** functions. As the SVD algorithm is already optimized, we cannot efficiently compare our two algorithms.

Nevertheless, it is established that the Greedy algorithm can compute reduced basis faster than POD, because of its iterative form. As expected, the Greedy algorithm does not oversee all the space.

2.4.4 Training set size influence

As we can see in Fig. 15, the training set used for reduced basis creation has a huge influence. As before, we took the maximum error in 100 fast online evaluations. Tolerance was put at $1 \times 10{-}5$. We can see that, for some training sets, the tolerance is not sufficiently low to obtain more accuracy.

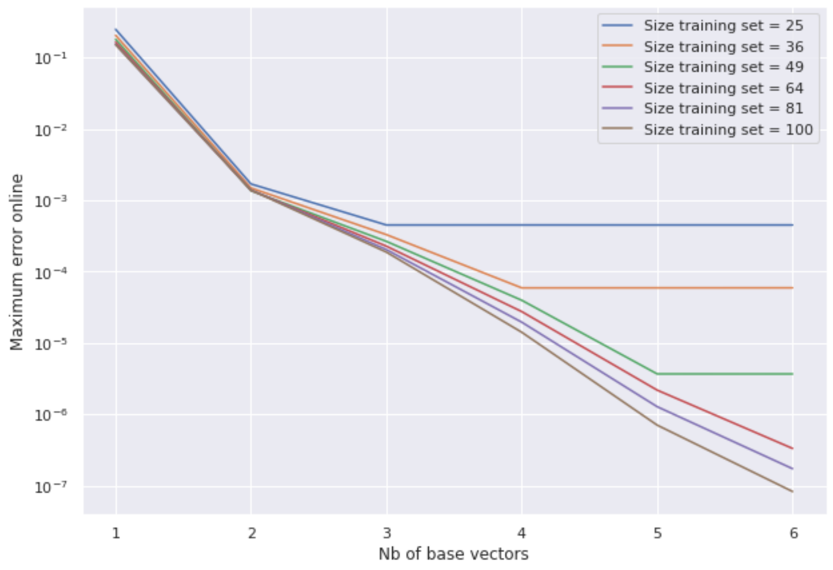

Figure 15: Errors comparison for POD method with respect to basis number of vectors

Globally, this figure illustrates that the higher the number of parameters evaluated, the more precise the reduced basis is. However, computation time also increases accordingly.

3 Implementation of the RBM for Burgers equation

The implementation of the RBM is significant for Burgers equation for a simple reason: it is a nonlinear PDE. In [3], they implemented the method directly, however, we will use the Cole-Hopf transformation to linearize the equation. The transformation can be a good robustness test of the error: as the reduced solution must be transformed back to obtain Burgers solution, the error transformation can make the solution inconsistent. In this part we will focus on the resolution of viscous Burgers equation parametrized by initial and boundary values. This section is organised as follows: we first introduce the model, the Cole-Hopf transform and a standard numerical solving, then we develop our reduced basis method to compute solutions faster, and end by discussing our results through different numerical resolutions and efficiency tests.

3.1 Model

In this part we describe the model we are interested in and presents its numerical solving.

3.1.1 Burgers equation

We search for u function of time $t \in [0;T]$ and space $x \in [0;1]$ satisfying *viscous Burgers equation*:

$$\partial_t u(t,x) + u(t,x)\partial_x u(t,x) - \mu \partial_{xx} u(t,x) = 0 \ , \ \forall t \in]0;T], x \in [0;1] \tag{1}$$

To ensure completion of the problem, we add the *Dirichlet boundary conditions*:

$$\begin{cases} u(t,0) = b_0(t) \ , \ \forall t \in [0;T] \ ; \\ u(t,1) = b_1(t) \ , \ \forall t \in [0;T] \end{cases} \tag{2}$$

And *initial value condition*:

$$u(0,x) = u_0(x) \ , \ \forall x \in [0;1] \tag{3}$$

With b_0, b_1, u_0 parametrized functions detailed in section 3.2.1.

3.1.2 Cole-Hopf transform

To linearize (1), one can use the *Cole-Hopf transform*:

$$u(t,x) = -2\mu \frac{\partial_x v(t,x)}{v(t,x)} \ , \ \forall t \in [0;T], x \in [0;1] \tag{4}$$

We know the following property holds: let u be Burgers solution, and v satisfies (4), then v is solution of heat equation.
Numerically, division by v could be a problem for $v << \partial_x v$ or being close to 0. We will come back to this problem in section 3.3.

3.1.3 Heat equation

Let v solves the standard *heat equation*:

$$\partial_t v(t,x) = \mu \partial_{xx} v(t,x) \ , \ \forall t \in [0;T], x \in [0;1] \tag{5}$$

Thus the Cole-Hopf transform allows the solving of Burgers equation through computation of the heat equation solution. We now must pay attention to the completion of the heat equation: both initial value conditions and boundary conditions are dependant on the base problem (namely Burgers equation). We can find them by passing base conditions in the Cole-Hopf transform:

- For the initial value condition, we obtain the following ODE:

$$u_0(x) = -2\mu\frac{v_0'(x)}{v_0(x)} \ , \ \forall x \in [0;1]$$

Hence, the resolution of this ODE leads to the following *initial value condition for heat equation*:

$$v(0,x) = v_0(x) = Ce^{\frac{-1}{2\mu}\int_0^x u_0(s)ds} \ , \ C \in \mathbb{R}^* \tag{6}$$

- For boundary conditions, we obtain the following:

$$\begin{cases} b_0(t) = -2\mu\frac{\partial_x v(t,0)}{v(t,0)} \ , \ \forall t \in [0;T] \ ; \\ b_1(t) = -2\mu\frac{\partial_x v(t,1)}{v(t,1)} \ , \ \forall t \in [0;T] \end{cases} \tag{7}$$

With (5), (6) and (7) the problem is complete and we can start numerical resolution.

3.1.4 Space discretization

As we are considering a 1D heat-equation, we choose the finite-difference methods. For a given integer N the space discretization is $x_i = i\delta_x \ , \ \forall i \in [\![0,N]\!]$ with $\delta_x = \frac{1}{N}$. We can construct the vector of discrete solution :

$$V_i(t) = v(t,x_i) \ , \ \forall i \in [\![0,N]\!]$$

We approximate $\partial_{xx}v(t,x)$ by its centered second order approximation and from (5) obtain the following:

$$\partial_t V_i(t) = \mu\frac{V_{i+1}(t) - 2V_i(t) + V_{i-1}(t)}{2\delta_x^2} \ , \ \forall i \in [\![0,N]\!] \tag{8}$$

In the same way, the initial value condition (6) becomes:

$$V_i(0) = v_0(x_i) \ , \ \forall i \in [\![0,N]\!] \tag{9}$$

And boundary conditions (7) become:

$$V_X(t)b_X(t) = -2\mu\partial_x V_X(t) \ , \ \forall t \in [0;T]$$

Hence, choosing the first order approximation $\partial_x v(t,0) = \frac{V_0(t)-V_{-1}(t)}{\delta_x}$ and $\partial_x v(t,1) = \frac{V_{N+1}(t)-V_N(t)}{\delta_x}$ we get the following Neumann-like interactions at boundaries:

$$\begin{cases} \frac{-\delta_x}{2\mu}V_0(t)b_0(t) = & V_0(t) - V_{-1}(t) \ , \ \forall t \in [0;T] \\ \frac{-\delta_x}{2\mu}V_N(t)b_1(t) = & V_{N+1}(t) - V_N(t) \ , \ \forall t \in [0;T] \end{cases}$$

$$\iff \begin{cases} V_{-1}(t) = & V_0(t)(1 + \frac{\delta_x}{2\mu}b_0(t)) \ , \ \forall t \in [0;T] \\ V_{N+1}(t) = & V_N(t)(1 - \frac{\delta_x}{2\mu}b_1(t)) \ , \ \forall t \in [0;T] \end{cases}$$

3.1.5 Time discretization

To eliminate stability constraints we chose implicit Euler's method. For a given integer T_{max} we can construct the time increment $\delta_t = \frac{1}{T_{max}}$, a time discretization space $t_j = j\delta_t \ , \ \forall j \in [\![0,T_{max}]\!]$, and the two-dimensional solution vector

$$V_i^j = v(t_j,x_i) \ , \ \forall (i,j) \in [\![0;N]\!] \times [\![0;T_{max}]\!]$$

We approximate the time derivative to obtain from (8) the following fully discrete equation:

$$V_i^j - V_i^{j-1} = \frac{\mu\delta_t}{2\delta_x^2}(V_{i+1}^j - 2V_i^j + V_{i-1}^j) \ , \ \forall(i,j) \in [\![1;N-1]\!] \times [\![0;T_{max}]\!]$$

and at boundaries:

$$\begin{cases} V_0^j - V_0^{j-1} = \frac{\mu\delta_t}{2\delta_x^2}(V_1^j + V_0^j(-1 + \frac{\delta_x}{2\mu}b_0(t_j))) \ , \ \forall(i,j) \in [\![1;N-1]\!] \times [\![0;T_{max}]\!] \ ; \\ V_N^j - V_N^{j-1} = \frac{\mu\delta_t}{2\delta_x^2}(V_N^j(-1 - \frac{\delta_x}{2\mu}b_1(t_j)) + V_{N-1}^j) \ , \ \forall(i,j) \in [\![1;N-1]\!] \times [\![0;T_{max}]\!] \ ; \end{cases}$$

We can rewrite this equation to obtain the linear system:

$$A^j V^j = V^{j-1} \ , \ \forall j \in [\![1;T_{max}]\!] \tag{10}$$

With A^j a tridiagonal matrix defined as follows:

$$A^j = \begin{pmatrix} 1+\alpha-\beta_0(t_j) & -\alpha & 0 & \cdots & \cdots & 0 \\ -\alpha & 1+2\alpha & & \ddots & & \vdots \\ & & & & \ddots & \vdots \\ 0 & & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & & & & \\ \vdots & & \ddots & & 1+2\alpha & -\alpha \\ 0 & \cdots & \cdots & 0 & -\alpha & 1+\alpha+\beta_1(t_j) \end{pmatrix}$$

With $\alpha = \frac{\mu\delta_t}{2\delta_x^2}$, $\beta_0(t_j) = \frac{\delta_t}{4\delta_x}b_0(t_j)$ and $\beta_1(t_j) = \frac{\delta_t}{4\delta_x}b_1(t_j)$
Thus we solve a tridiagonal linear system at each time iteration. We used the sparse solver *scipy.sparse.linalg.solvebanded*.

3.1.6 Getting back to Burgers solution

Now we can compute the heat solution, we use (4) to return to Burgers solution, thus, by approximating space derivative by the centered finite difference we obtain the following:

$$U_i^j = -\mu\frac{V_{i+1}^j - V_{i-1}^j}{\delta_x V_i^j} \ , \ \forall(i,j) \in [\![1;N-1]\!] \times [\![0;T_{max}]\!] \tag{11}$$

Therefore, the Burgers solver method can be summarized by these steps:

- For given functions b_0, b_1 and u_0;
- Choose N and T_{max} for heat solver;
- Compute $V_i^0 = e^{\frac{-1}{2\mu}\int_0^{x_i} u_0(s)ds}$;
- While $t_j < T$ solve the linear system at current time $A^jV^j = V^{j-1}$;
- Back to Burgers solution: $U_i^j = -\mu\frac{V_{i+1}^j - V_{i-1}^j}{\delta_x V_i^j}$;

3.2 The reduced basis method for Burgers equation

In this part, we develop the parametrization of initial value and boundary conditions together with showing the numerical offline/online procedure for problem reduction.

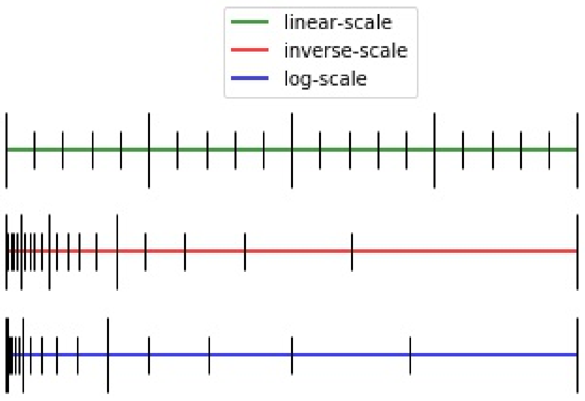

Figure 16: representations of different discretization scales of parameters space

3.2.1 Parameters

We choose as parameters the viscosity μ, and parametrized the initial Gaussian shape with σ and ν:

$$u_0(x) = e^{-\frac{(x-1/2)^2}{\sigma^2}} + \nu$$

We also set $b_0(t) = b_1(t) = u_0(0) = u_0(1)$ (as our initial value is symmetric). Therefore, our boundary conditions are constants over time.
ν is also a regularization term; maintaining Burgers solution above a minimal value prevents its numerical resolution from dropping under 0, changing movement direction and creating singularities.
We chose our 3 parameters to belong to the Cartesian product: $(\mu, \sigma, \nu) \in [\frac{1}{100}; 10] \times [\frac{1}{50}; \frac{1}{4}] \times [\frac{1}{10}; 1]$.

3.2.2 POD Reduction procedure

We implemented the POD procedure for creating a reduced basis of heat solutions. The key to the method is that we did not learn Burgers solutions, but heat solutions. In fact, the heat equation is a linear PDE, and we solve it easily with a tridiagonal system, so it is easier to implement the CRBM for the heat problem reduction. It is only after taking the Cole-Hopf transform of our reduced solution that we can talk about Burgers reduced solutions.
As described in section 1.2.1, we developed the POD as an offline procedure to select a reduced space on which we solve the reduced heat problem. What we discuss in this part is the choice of parameters-space discretization.
As we want the parameters space not to be too heavy to loop in, and because our parameters are not of the same impact and not in the same range, we need to choose wisely the discretized parameter-space. For example, μ belongs to $[\frac{1}{100}, 10]$, thus discretizing its space evenly gives as much importance to $\mu \in [9; 10]$ (high viscosity problems) as to all $\mu \in [\frac{1}{100}; 1]$ (a wide range of low to mid viscosity problems). And the same argument applies to σ. We want the discretization not to be even on parameters, but to represent as best as possible the range of solutions the POD needs to learn. Therefore, we choose our parameters discretized-space $\mathbb{P}_\delta$ to be in logarithmic scale for μ, inverse scale for σ and linear scale for ν. The three scales we use are represented in Figure 16 above.

3.2.3 Reduced solver implementation

After the offline POD procedure, we must implement the problem-dependant algorithms; the reduced solver and the pre-computer for affine-parameters parts of the solver.
First, the pre-computer: As explained in [2], we look for an affine decomposition of A^j and implement a pre-computer of all quantities independent of parameters.
We have chosen time independent bounds, so $A = A^j$ is also a constant matrix, independent of time. Therefore A allows the affine decomposition:

$$A = A_0 + \mu A_1 - \beta_0 A_2 + \beta_1 A_3 \tag{12}$$

with $A_0 = Id$, $A_1 = \begin{pmatrix} \alpha & -\alpha & & 0 & \cdots & \cdots & 0 \\ -\alpha & 2\alpha & & & \ddots & & \vdots \\ & & & & & \ddots & \vdots \\ 0 & & \ddots & \ddots & \ddots & & 0 \\ \vdots & \ddots & & & & & \\ \vdots & & \ddots & & & 2\alpha & -\alpha \\ 0 & \cdots & \cdots & 0 & & -\alpha & \alpha \end{pmatrix}$ (note that $\alpha = \frac{\delta_t}{2\delta_x^2}$ here),

$A_2 = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & & 0 & \\ 0 & & & \end{pmatrix}$, and $A_3 = \begin{pmatrix} & & & 0 \\ & 0 & & \vdots \\ & & & 0 \\ 0 & \cdots & 0 & 1 \end{pmatrix}$

Hence, the pre-computer defines A_q matrices for $q \in [\![0;4]\!]$ and applies the basis shift

$$A_q^{rb} = B^T A_q B$$

with B the reduced basis found earlier. Thus, during the online stage the reduced solver can assemble the reduced operator

$$A^{rb} = A_0^{rb} + \mu A_1^{rb} - \beta_0 A_2^{rb} + \beta_1 A_3^{rb} \tag{13}$$

The reduced solver steps are the following (with R the reduced solution vector):

- load A_q^{rb} and assemble A^{rb} as in (13);
- load $V_i^0 = v_0(x_i)$ and set $R^0 = B^T V^0$ (orthogonal projection of initial value condition into the reduced space);
- While $t_j < T$, solve the linear system at current time $A^{rb} R^j = R^{j-1}$ and increment time by δ_t

3.3 Results

In this section we show some basis reduction method solutions and compare them to "true solutions" (obtained with the method described in section 3.1.6) by using indicators and plotting graphs. We show the limitation of our work and of the method.

3.3.1 Indicators

To measure the closeness of a solution to the reference solution, we need error indicators. Based on the usual $\|\cdot\|_1$ and $\|\cdot\|_\infty$, we used:

$$mean\ error(Approx, Ref) = \frac{1}{N \times T_{max}} \sum_{i=0}^{N} \sum_{j=0}^{Tmax} |Approx_i^j - Ref_i^j|$$

$$ratio\ error\ max(Approx, Ref) = \frac{\max_{i=0}^{N} \max_{j=0}^{Tmax} |Approx_i^j - Ref_i^j|}{\max_{i=0}^{N} \max_{j=0}^{Tmax} |Ref_i^j|}$$

3.3.2 POD

We computed a 20 vectors reduced basis using POD. We set the discretizations as follows:

- 1025 space nodes,
- 61 time nodes,
- 20 nodes for $\mu \in [10^{-2}; 1]$ with a log scale.
- 20 nodes for $\sigma \in [\frac{1}{40}; \frac{1}{10}]$ with an inverse scale.
- 4 nodes for $\nu \in [\frac{1}{10}; \frac{1}{2}]$ with a regular scale.

Thus we computed $20 \times 20 \times 4 = 1600$ solutions and analysed $1600 \times 61 = 97600$ vector space solutions. The SVD error was 7.329×10^{-12}, and the time each step took was respectively:

- 17.4 seconds to compute all solutions,
- 1204.2 seconds to perform the SVD,
- 24.3 seconds to keep only the best vectors and return the reduced basis.

The SVD slows the algorithm, this is why the Greedy algorithm (which does not need to perform an SVD) can be relevant for faster reduced basis generation.

3.3.3 First result

As a first result, we show in figure 17 a solution along with its errors to the true solution: $mean\ error = 5.302 \times 10^{-5}$ $ratio\ error\ max = 1.175 \times 10^{-3}$.
It took 3.83 ms to compute this reduced solution, while the true solver took 9.85 ms. The time reduction factor is $\approx$ 2 to 3 for the solutions we computed. This result is significant knowing we are comparing with an optimized solver for tridiagonal linear systems. Furthermore, we can expect computation time to be proportional to the number of space nodes. Thus the method could lead to divide time cost by up to 9 for a two-dimensional problem and 27 for a three-dimensional problem.

The errors are very low, indeed it is a relevant approximation and the algorithm is satisfactory in this case.
We found for many reduced solutions that the initial errors were the biggest. This problem can be solved using the Greedy algorithm: we can force the reduced basis to be more accurate for initial repartitions. However, the initial error made get smaller with time, thus this lack of precision is not a significant problem. Furthermore, it emphasizes a property of our method: an error made at a given time will not be carried through all subsequent solutions. In fact the Cole-Hopf transformation is more sensitive to errors than the resolution itself. This is also why for reduced heat solutions with even very small errors (for heat counterpart of Figure 17, $mean\ error = 2.307 \times 10^{-6}$, and $ratio\ error\ max = 2.809 \times 10^{-5}$) reduced Burgers solutions lose precision.

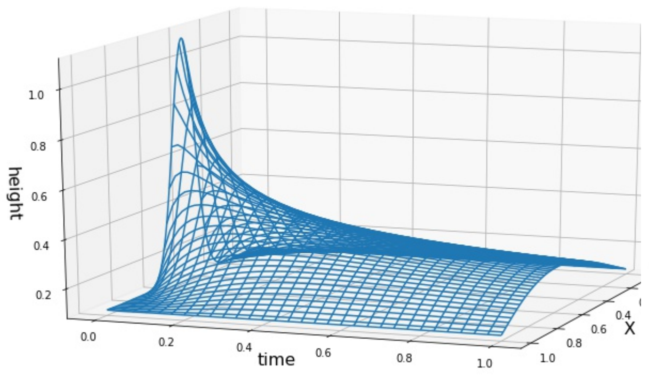

Figure 17: Reduced Solution for $\mu = 0.15,\ \sigma = 0.1,\ \nu = 0.1$

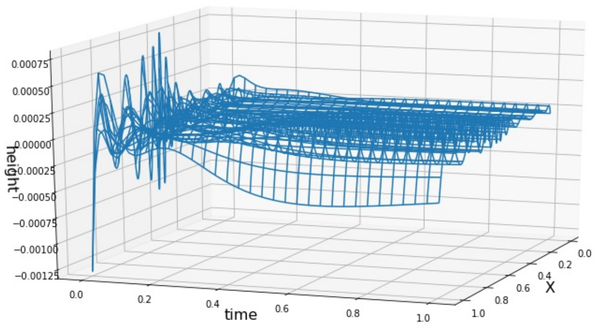

Figure 18: Difference with true solution

3.3.4 Problem at boundaries

For a lot of parameters, we find relatively safe approximations, but increasing ν leads sometimes to high inaccuracies always at boundary x=1, as we can see comparing Figure 19, and Figure 21. The only difference between these figures is parameter ν.

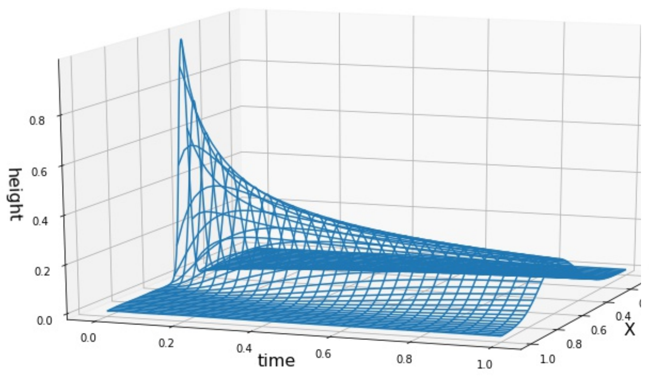

Figure 19: Reduced Solution for $\mu = 0.02,\ \sigma = 0.04,\ \nu = 0.$

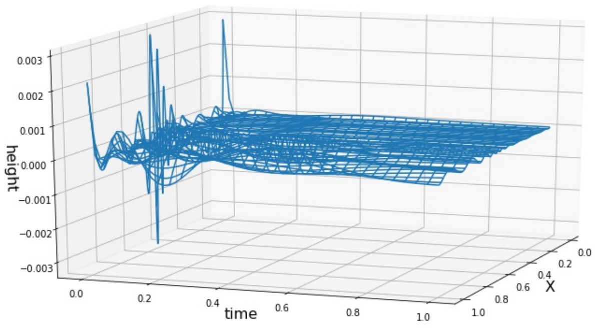

Figure 20: Difference with true solution

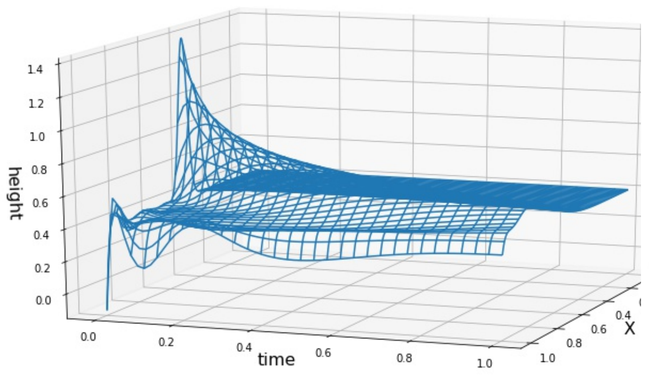

Figure 21: Reduced Solution for $\mu = 0.02,\ \sigma = 0.04,\ \nu = 0.4$

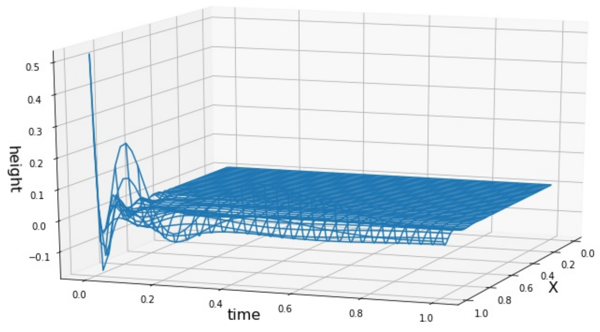

Figure 22: Difference with true solution

We have the following errors:

- For reduced heat solution with $\nu = 0$: *mean error* $= 9.946 \times 10^{-6}$, and *ratio error max* $= 2.888 \times 10^{-4}$;
- For reduced Burgers solution with $\nu = 0$: *mean error* $= 7.233 \times 10^{-5}$, and *ratio error max* $= 3.324 \times 10^{-3}$;
- For reduced heat solution with $\nu = 0.4$: *mean error* $= 2.109 \times 10^{-7}$, and *ratio error max* $= 3.469 \times 10^{-6}$;
- For reduced Burgers solution with $\nu = 0.4$: *mean error* $= 2.728 \times 10^{-3}$, and *ratio error max* $= 3.723 \times 10^{-1}$.

We see clearly an error all along axis $x = 1$. As we measured fidelity to the true solution for reduced Burgers solution and also for the reduced heat solution, we can state this is not due to an inaccurate reduced heat solution. It may be a problem in the dirichlet bounds at $x = 1$.

Conclusion

In this paper, we explored the reduced basis method, compared basis generation algorithms: the POD and the Greedy algorithm, and studied the method implementation for a nonlinear PDE: Burgers equation.

We saw theoretically why the POD provides a more accurate approximation than the Greedy algorithm and illustrated it numerically. Although the Greedy algorithm should be faster than the POD, we observed the contrary. We think this is because the POD is essentially a SVD (method taken from a library), whereas we implemented the Greedy algorithm by our own, which could be optimized.

The results of the method for Burgers equation are indecise, Cole-Hopf transform allows us to solve Burgers equation as a linear PDE, leading to an important gain of time, but it creates inaccuracies and boundary conditions lead sometimes to tremendous errors. However, we think time reduction should be even better for two or three dimensional problems. Future work could be done to extend the method for Neumann boundary conditions.

Furthermore, in order to enhance the computation of the reduced basis, we should try to implement an algorithm that includes both POD and Greedy algorithm to combine the advantages of each algorithm. This POD-Greedy algorithm is described in [2]. We also noticed that the error decreased as we refined the parameter space more and more. Some strategies could be to increase refinement of the mesh used for the space parameter, to chose randomly some points in that space instead of uniformly, and we could increase the number of points taken around some specific zones in the space that could carry more information. Also, we noticed that a Python package called **RBniCS** had implemented all the algorithms from our main study paper [2]. All the information about this package can be found at `https://gitlab.com/RBniCS/RBniCS/`.

Some studies could focus on enlarging these methods to high-dimensional problems, and test accuracy and time efficiency on them, as our problems were simple and low-dimensional problems.

References

[1] M. S. Alnæs, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. E. Rognes, and G. N. Wells, *The fenics project version 1.5*, Archive of Numerical Software, 3 (2015).

[2] J. Hesthaven, G. Rozza, and B. Stamm, *Certified Reduced Basis Methods for Parametrized Partial Differential Equations*, Springer, Jan. 2016.

[3] A. Janon, *Sensitivity analysis and model reduction : application to oceanography*, no. 2012GRENM066, Nov. 2012.

[4] H. P. Langtangen and A. Logg, *Solving PDEs in Python*, Springer, 2017.