

Improvement of code behavior in a design of experiments by metamodeling

François Bachoc, Karim Ammar, Jean-Marc Martinez

▶ **To cite this version:**

François Bachoc, Karim Ammar, Jean-Marc Martinez. Improvement of code behavior in a design of experiments by metamodeling. Nuclear Science and Engineering, 2016, 183 (3), pp.387-406. 10.13182/NSE15-108 . cea-02382795

HAL Id: cea-02382795

https://cea.hal.science/cea-02382795

Submitted on 27 Nov 2019

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.

Improvement of code behavior in a design of experiments by metamodeling

François Bachoc*†

Institut de Mathématiques de Toulouse

Karim Ammar

CEA-Saclay, DEN, DM2S, SERMA, LPEC, F-91191 Gif-Sur-Yvette, France

Jean-Marc Martinez

CEA-Saclay, DEN, DM2S, STMF, LGLS, F-91191 Gif-Sur-Yvette, France

Total number of pages: 47

Total number of tables: 4

Total number of figures: 9

*Corresponding author. E-mail: francois.bachoc@math.univ-toulouse.fr Address: Institut de Mathématiques de Toulouse, Université Paul Sabatier, 118 route de Narbonne, 31062 TOULOUSE Cedex 9, France. Phone: 0033 5 61 55 69 16

†The author conducted a part of the research related to this manuscript when he was affiliated first to CEA-Saclay, DEN, DM2S, STMF, LGLS, F-91191 Gif-Sur-Yvette, France and then to the University of Vienna.

Abstract

It is now common practice in nuclear engineering to base extensive studies on numerical computer models. These studies require to run computer codes in potentially thousands of numerical configurations and without expert individual controls on the computational and physical aspects of each simulations. In this paper, we compare different statistical metamodeling techniques and show how metamodels can help to improve the global behaviour of codes in these extensive studies. We consider the metamodeling of the Germinal thermalmechanical code by Kriging, kernel regression and neural networks. Kriging provides the most accurate predictions while neural networks yield the fastest metamodel functions. All three metamodels can conveniently detect strong computation failures. It is however significantly more challenging to detect code instabilities, that is groups of computations that are all valid, but numerically inconsistent with one another. For code instability detection, we find that Kriging provides the most useful tools.

1 Introduction

Physical models and corresponding computer codes enable to evaluate nuclear reactor performances within a computation time of a few hours (see e.g. [1, 2]). However, to be relevant, an optimization or a propagation of uncertainty study requires the numerical evaluations of a significant number of reactor configurations (for instance several millions in optimization [3]). Because of the current limitation of computing resources, these procedures can not be directly applied to computer codes. Hence, metamodels, that provide a computationally cheap approximation of the output of computer codes, are commonly used. Metamodels are constructed from a learning base of code inputs and outputs, the generation of which is called a design of experiments. In this paper, we give a detailed analysis of the metamodeling process for the Germinal V1 thermomechanical code [4]. [Note that the metamodels of the Germinal code are typically intended to be used in an optimization process applied to a sodium cooled fast reactor [5].]

Different metamodeling methods (neural networks, Kriging and kernel methods) are analyzed and benchmarked in this paper. Furthermore, as detailed below, metamodels can not only predict computer code results but also contribute to improve the behavior of these codes during the design of experiments.

To understand this last point it is important to highlight that it is challenging to automatically carry out several thousands of code simulations, as is typically the case in a design of experiments. Indeed current codes in nuclear engineering are complex:

- The code inputs and outputs are not simple scalars. For instance, it may be necessary to generate a 3D geometry and its associated meshing[1].

- It requires significant expertise to assess or anticipate the numerical validity of a calculation. Indeed, many different convergence criteria need to be taken into account. A code output file may include several indications such as "error" or "warning", which impact is difficult to assess.

- There is a large number of possible calculation options that can be mixed.

Germinal V1 is one of multiple codes designed in the 1990s, developed to be launched manually and on a case-by-case basis. That is, for each run, an expert generates the input files and checks the consistency of the code results. However, in a design of experiments, many code runs need to be carried out, each of which can not be managed manually. It is hence necessary to develop a "code manager", as schemed in Figure 1. In order to explain the code manager, consider a parametric study, where each simulation is characterized by a finite number of scalar parameters (see Section 2 for those considered in this paper). Then, the code manager consists first in a preprocessor which generates the code input files from the parametric variables (for example, it may automatically construct an axial mesh from a global variable like a height). After the code execution, the code manager also incorporates a postprocessor which checks the occurrence of computational failures and then condense the

[1]See for instance `http://www.salome-platform.org`

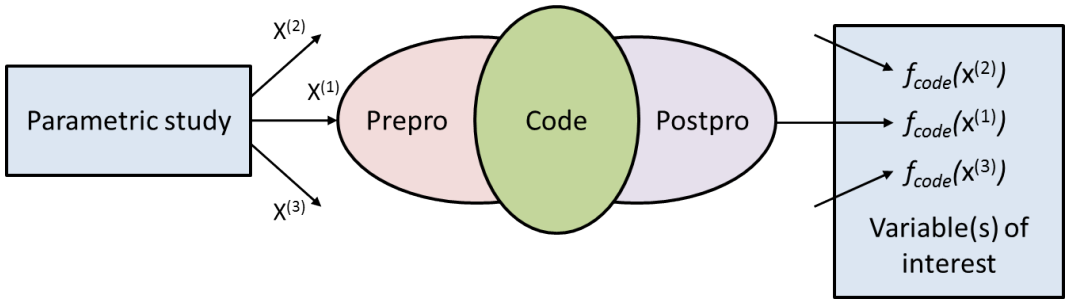


Figure 1: Illustration of the code manager. In a parametric study, all code simulations are characterized by a finite-dimensional vector $\boldsymbol{x}$ of scalar parameters. The code manager first consists in a preprocessor, which generates code inputs from these parameters. After the simulation, the code manager incorporates a postprocessor, which checks the occurrence of computational failure and then condense the code output in some variables of interest.

code output file in some variables of interest. The construction of an efficient postprocessor, able to correctly interpret all the output messages produced by the code, is a real challenge. Indeed, when many simulations are carried out for many different inputs, a very large number of failure scenarios can occur, not all of which can be anticipated.

In this paper, we consider a parametric study in which the Germinal code is used to evaluate the thermalmechanical response of a nuclear fuel pin to irradiation. Each simulation is characterized by 11 parameters (given in Section 2), and we consider a single scalar variable of interest for the simulation results. Hence, the code manager is represented by a function f from $\mathbb{R}^{11}$ to $\mathbb{R}$ that the metamodels under consideration aim at approximating. In the sequel, we refer to this function as the code function.

We compare the metamodels obtained from neural networks, Kriging and kernel methods. We find that neural networks require the shortest computation time for metamodel evaluation, while Kriging provides the most accurate

predictions. Kernel methods, and most of all Kriging, provide valuable accuracy indicators for their predictions.

We also analyze several issues related to the construction of the code manager. We show that the postprocessor can fail to detect computation failures, and that the preprocessor can generate code input files, as a function of the simulation parameters, in an inconsistent way. This preprocessor issue yields code instabilities, that is groups of computations with similar input conditions but overly different output values.

The issue with the postprocessor is, as we show, well solved by the three metamodels. Indeed, their prediction errors for the simulations in the learning base can be investigated, and a few outlier computations can be flagged. It is then possible to manually check these computations and to confirm their numerical failures.

On the other hand, code instabilities are significantly more difficult to handle and we find that Kriging provides the best tools to tackle them. Indeed, the estimated nugget effect in the Kriging metamodel (to be defined in Section 3.1) is a direct quantifiers of small scale variations of the Germinal code function. This nugget effect turns out to be large in the first learning base we have considered. As a consequence, we have investigated the preprocessor behaviour, and we have found and solved an important input file generation issue. This improvement of the preprocessor results in an updated version of the code manager, from which we have generated an updated Germinal simulation base. We find that the three metamodels are more accurate for predicting the updated Germinal computations. Furthermore, the estimated nugget effect for Kriging decreases between the original and updated

computations, confirming the improvement of the code manager. In light of this discussion, we believe that the estimated nugget effect of Kriging is a reliable quantifier of the global order of magnitude of the code instabilities.

The rest of the paper is organized as follows. In Section 2, we present in details the parametric study for the Germinal code. In Section 3, we introduce the Kriging, kernel and neural network metamodels. In Section 4, we discuss the prediction results of the metamodels for the original Germinal computations. In Section 5, we show how the metamodels, and in particular Kriging, help in detecting outlier computations and code instabilities. In Section 6, we present the resulting prediction improvement of the metamodels for the updated computations.

Finally, it should be noted that, in this paper, we do not discuss the important problem of code validation. That is, we aim at predicting the output of the Germinal code, without assessing if the picture displayed by this output is representative of the underlying physical reality. We refer to [6, 7] for references on code validation. Remark nevertheless that constructing an accurate metamodel of a code is also useful for its validation (see e.g. [8] for the Kriging metamodel).

2 Presentation of the parametric study for the Germinal code

2.1 Fuel pin thermomechanical simulation with the Germinal code

It is well known that material properties evolve when they are submitted to high neutron flux. In particular, in fast breading reactors, irradiation can have a strong impact on fuel pin thermomechanical properties. The Germinal code V1 [4] can be used to simulate the temporal evolution of these thermomechanical properties, resulting from irradiation. This code implements a simplified fuel description model based on mono-group neutron flux, power and irradiation damage distribution as well as sodium inlet temperature and mass flow per pin. In this paper, the variable of interest we focus on is the fusion margin, which is the difference between the fuel melting temperature (around 2700°) and the maximal fuel temperature obtained throughout the Germinal simulation of the fuel life.

We consider this variable of interest since its prediction by metamodels is particularly challenging. Indeed, the computation of the fuel temperature depends on the fuel conductivity and on the heat transfer coefficients between the fuel pellets and the cladding. These coefficients depend on the irradiation in a strong non-linear manner. Hence, in the parametric study described below, the fusion margin is definitely a non-linear function of the simulation parameters. Note also that, in a context of multi-physical optimization (neutron physics, thermal-hydraulics and thermomechanics) for fast reactor core

(using the TRIAD platform [5]), we have built neural network metamodels for a large number of variables of interests of the Germinal code. We have found that the fusion margin variable was the most difficult to predict by the neural networks.

Finally, the fusion margin is interesting in that it characterizes two very different physical regimes. When it is large and positive, the fuel pin mechanical properties do not change throughout the simulation. When, it is small, or negative, they do, which results in much more involved physical phenomena, that are challenging to model numerically. As a result, the fusion margin is generally more difficult to predict by metamodels when it is small or negative.

2.2 The parametric study

We are interested in a parametric study where a Germinal simulation is characterized by 11 scalar parameters $x_1, ..., x_{11}$ defined as follows. [These parameters are used by the preprocessor to generate input files for the Germinal code, see Figure 1.]

- The parameter x_1 is the cycle length in the fuel pin simulation.
- The parameters $x_2, ..., x_7$ characterize the nature of the fuel pin. The parameter x_2 is the plutonium concentration, x_3 is the diameter of the fuel hole, x_4 is the external diameter of the clad, x_5 is the thickness of the gap between the fuel and the clad, x_6 is the thickness of the clad and x_7 is the height of the fuel pin. Figure 2 provides a visualization of x_3 to x_6.

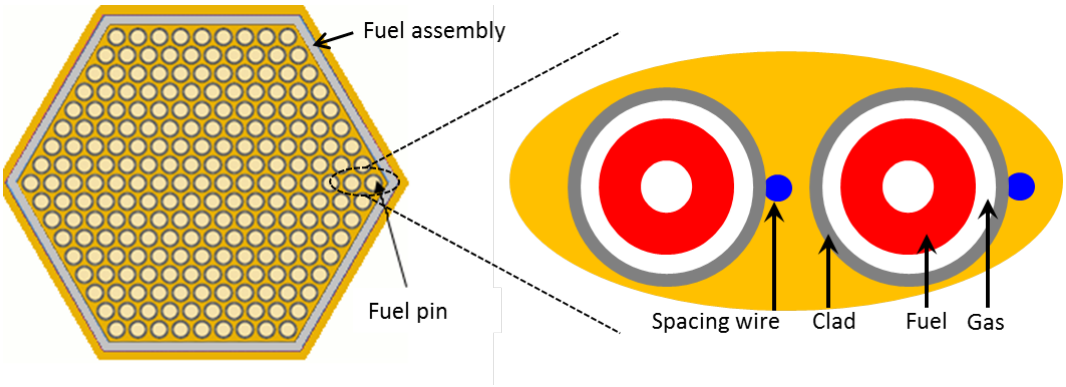


Figure 2: A schematic representation of a fuel pin and a fuel assembly in nuclear fast-neutron reactors.

- The parameters x_8, x_9, x_{10} characterize the power map in the fuel pin, with x_8 the average power, x_9 the axial form factor and x_{10} the power shift due to the fuel depletion. [Note that, in the case of multi-physics coupling [5, 9], x_9, x_{10} and x_{11} would be obtained from a neutron-physic simulation.]

- The parameter x_{11} is the volume of expansion for fission gas.

When building the learning base for the metamodels, we consider an hypercubic domain for the 11 input variables, characterized by 11 minimal and maximal values, summarized in Table I. The learning base is then obtained by first generating a LHS-Maximin [10] set of input parameters $\boldsymbol{x} = (x_1, ..., x_{11})$ on this hypercubic domain, and then removing some of them that can be shown to result in infeasible fuel pins prior to carrying out the corresponding Germinal simulation. The resulting learning base under consideration in this paper includes 3807 input vectors $\boldsymbol{x}$. We write $y_1 = f(\boldsymbol{x}^{(1)}), ..., y_n = f(\boldsymbol{x}^{(n)})$ for the $n = 3807$ computation inputs and

parameter	x_1 : cycle length (EFPD)	x_2 : plutonium content (% atomic)	x_3 : hole diameter (mm)	x_4 : external clad diameter (mm)	x_5 : fuel gap (mm)	x_6 : clad thickness (mm)
min	360	10	0.125	6.2	0.1	0.5
max	440	30	3	12.8	0.2	0.6

parameter	x_7 : pin height (mm)	x_8 : average pin power (W/cm)	x_9 : axial form factor	x_{10} : power shift	x_{11} : volume of expansion (cm^3)
min	60	150	1	0.8	32
max	160	440	1.6	1.2	94

Table I: Minima and maxima of the intervals of variations for the 11 simulation parameters in the parametric study. [EFPD stands for Equivalent Full Power Day.]

outputs.

3 Presentation of the metamodels

3.1 Kriging

Kriging is widely used in Nuclear Engineering, for instance for metamodeling of computer codes [11] or for improving their predictions using experimental

results [12]. In this paper, we use a standard implementation of Kriging [10, 13, 14], at the exception of the numerical optimization of the likelihood, see below.

Gaussian process model. The Kriging metamodel is based on modeling a deterministic computer code function $f_{code} : \mathcal{D} \subset \mathbb{R}^d \to \mathbb{R}$ (where in Section 2, $d = 11$) as a realization of a Gaussian process Y on $\mathcal{D}$. That is, we assume, for $\boldsymbol{x} \in \mathcal{D}$,

$$f_{code}(\boldsymbol{x}) = Y(\omega, \boldsymbol{x}),$$

where ω is a fixed element in a probability space Ω. Hence, the paradigm is that, although the computer code function is fixed, the value $f_{code}(\boldsymbol{x})$ remains unknown to the user as long as a computation is not carried out for the conditions $\boldsymbol{x}$. The Kriging metamodel thus follows a Bayesian approach, by considering the unknown $f_{code}(\boldsymbol{x})$ as the realization of a Gaussian variable $Y(\boldsymbol{x})$ (see also the presentation in [12]).

In this paper, we assume that the Gaussian process Y has a zero mean function and is hence characterized by its covariance function $C : \mathcal{D} \times \mathcal{D} \to \mathbb{R}$. In general, this covariance function is assumed to be continuous, so that the Gaussian process Y yields continuous realizations, which is consistent with the fact that the code function f_{code} is continuous, or at least that a small variation in the condition $\boldsymbol{x}$ causes only a small change in the computed value $f_{code}(\boldsymbol{x})$. Nevertheless, the Germinal code function studied in this paper is subject to small scale variations, meaning that a small variation in $\boldsymbol{x}$ can cause a significant change in $f_{code}(\boldsymbol{x})$, because of the code instabilities.

Hence, we assume that

$$C(\boldsymbol{x}^{(a)}, \boldsymbol{x}^{(b)}) = \sigma_0^2 \bar{C}(\boldsymbol{x}^{(a)}, \boldsymbol{x}^{(b)}) + \delta_0^2 \mathbf{1}\{\boldsymbol{x}^{(a)} = \boldsymbol{x}^{(b)}\}, \tag{1}$$

where $\bar{C}$ is a continuous correlation function and with $\sigma_0^2 > 0$ and $\delta_0^2 \geq 0$. Thus, Y can be written

$$Y(\boldsymbol{x}) = Y_c(\boldsymbol{x}) + Y_d(\boldsymbol{x}), \tag{2}$$

where $Y_c(\boldsymbol{x})$ is a continuous Gaussian process with covariance function $\sigma_0^2 \bar{C}(\boldsymbol{x}^{(a)}, \boldsymbol{x}^{(b)})$ and $Y_d(\boldsymbol{x})$ is a discontinuous Gaussian process with covariance function $\delta_0^2 \mathbf{1}\{\boldsymbol{x}^{(a)} = \boldsymbol{x}^{(b)}\}$. The term $\delta_0^2 \mathbf{1}\{\boldsymbol{x}^{(a)} = \boldsymbol{x}^{(b)}\}$ is referred to as a nugget effect [15].

Covariance parameter estimation. Most classically in Kriging, the covariance function C is estimated, from the computations $y_1 = f_{code}(\boldsymbol{x}^{(1)}), ..., y_n = f_{code}(\boldsymbol{x}^{(n)})$ in the learning base. In this paper, we estimate the covariance function within the parametric set

$$\mathcal{C} = \left\{ \sigma^2 \left[\bar{C}_{\boldsymbol{\ell}}(\boldsymbol{x}^{(a)} - \boldsymbol{x}^{(b)}) + \alpha \mathbf{1}\{\boldsymbol{x}^{(a)} = \boldsymbol{x}^{(b)}\} \right], \sigma > 0, \boldsymbol{\ell} \in (0, +\infty)^d, \alpha \geq 0 \right\}, \tag{3}$$

where $\bar{C}_{\boldsymbol{\ell}}(\boldsymbol{h})$ is the Matérn 3/2 correlation function,

$$\bar{C}_{\boldsymbol{\ell}}(\boldsymbol{h}) = (1 + \sqrt{6}|\boldsymbol{h}|_{\boldsymbol{\ell}}) \exp(-\sqrt{6}|\boldsymbol{h}|_{\boldsymbol{\ell}}),$$

with $|\boldsymbol{h}|_{\boldsymbol{\ell}} = \sqrt{\sum_{i=1}^d \frac{h_i^2}{\ell_i^2}}$.

The Matérn 3/2 correlation function $\bar{C}_{\boldsymbol{\ell}}(\boldsymbol{h})$ is one of the most commonly

used covariance functions. It is stationary, that is $\bar{C}_{\boldsymbol{\ell}}(\boldsymbol{x}^{(a)} - \boldsymbol{x}^{(b)})$ depends on $\boldsymbol{x}^{(a)}$ and $\boldsymbol{x}^{(b)}$ only through their difference. Furthermore, for every $\boldsymbol{\ell}$, this correlation function yields Gaussian process realizations that are exactly one time continuously differentiable (see for instance [13]). The component ℓ_i can be seen as a correlation length in the i-th dimension. When ℓ_i is small, the condition x_i is particularly important for the Gaussian process $Y(\boldsymbol{x})$. Conversely, if ℓ_i is very large, then the realizations of $Y(\boldsymbol{x})$ are almost independent of x_i.

The covariance parameters $\sigma^2, \boldsymbol{\ell}, \alpha$ are estimated from the learning base. In this paper, we address Maximum Likelihood estimation (ML) which is the most standard method. [We note that other methods can also be employed, like Cross Validation [14, 16, 17].] Let, $\mathbf{R}_{\boldsymbol{\ell},\alpha}$ be the $n \times n$ matrix defined by $(R_{\boldsymbol{\ell},\alpha})_{i,j} = \bar{C}_{\boldsymbol{\ell}}(\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}) + \alpha \mathbf{1}\{\boldsymbol{x}^{(i)} = \boldsymbol{x}^{(j)}\}$. Let $\boldsymbol{y}$ be the $n \times 1$ vector $(y_1, ..., y_n)^t$. Then, the Maximum Likelihood estimator of $\sigma^2, \boldsymbol{\ell}, \alpha$ is defined by

$$(\hat{\boldsymbol{\ell}}, \hat{\alpha}) \in \underset{(\boldsymbol{\ell},\alpha)}{\arg\min} \log\left(\frac{1}{n}\boldsymbol{y}^t \mathbf{R}_{\boldsymbol{\ell},\alpha}^{-1} \boldsymbol{y}\right) + \frac{1}{n}\log(|\mathbf{R}_{\boldsymbol{\ell},\alpha}|), \tag{4}$$

where $|.|$ is the determinant, and by

$$\hat{\sigma}^2 = \frac{1}{n}\boldsymbol{y}^t \mathbf{R}_{\hat{\boldsymbol{\ell}},\hat{\alpha}}^{-1} \boldsymbol{y}. \tag{5}$$

The nugget variance δ_0^2 in (1) is estimated by $\hat{\delta}^2 = \hat{\sigma}^2\hat{\alpha}$. Note that the advantage of the parameterization with $\sigma^2, \boldsymbol{\ell}, \alpha$ in (3), compared to a parameterization with $\sigma^2, \boldsymbol{\ell}, \delta^2$ as in (1), is that it provides an explicit expression for $\hat{\sigma}^2$.

Numerical optimization of the likelihood. The optimization problem (4) is relatively challenging in our case, since the optimization space has dimension $d+1=12$ and since n is around 3800 which makes it computationally costly to evaluate the determinant and to solve the linear system in (4).

Hence, we evaluate $\hat{\boldsymbol{\ell}}$ and $\hat{\alpha}$ in two steps. First, we select a random subsample of the learning base, of size 1000 and minimize the equivalent of the function (4), when the learning base is equal to this random subsample. We let $\hat{\boldsymbol{\ell}}$ and $\tilde{\alpha}$ be the outcome of this first step.

For the second, step, recall that $\mathbf{R}_{\hat{\boldsymbol{\ell}},\tilde{\alpha}}$ is the $n \times n$ matrix defined by $(R_{\hat{\boldsymbol{\ell}},\tilde{\alpha}})_{i,j} = \bar{C}_{\hat{\boldsymbol{\ell}}}(\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}) + \tilde{\alpha}\mathbf{1}\{\boldsymbol{x}^{(i)} = \boldsymbol{x}^{(j)}\}$. Consider a SVD decomposition of this matrix, $\mathbf{R}_{\hat{\boldsymbol{\ell}},\tilde{\alpha}} = \mathbf{USU}^t$, with $\mathbf{U}$ of size $n \times n$ so that $\mathbf{UU}^t = \mathbf{I}_n$ and $\mathbf{S}$ a diagonal matrix with diagonal elements $s_1 \geq ... \geq s_n > 0$. Then, let L_α be the function in (4) evaluated at $\hat{\boldsymbol{\ell}}, \alpha$. This function can be written, with $v_i = (\mathbf{U}^t \boldsymbol{y})_i$,

$$L_\alpha = \log\left(\frac{1}{n}\sum_{i=1}^{n}\frac{v_i^2}{s_i + \alpha - \tilde{\alpha}}\right) + \frac{1}{n}\sum_{i=1}^{n}\log\left(s_i + \alpha - \tilde{\alpha}\right), \tag{6}$$

which is computed with negligible computational cost. Hence, we can plot the graph of L_α and compute $\hat{\alpha}$ as its minimizer. Finally, $\hat{\sigma}^2$ is computed by (5).

Hence, in this two-step optimization procedure, only the first step entails an important computational cost. The second step is carried out in negligible time and provides an estimation of the nugget component α that is more accurate since all the elements of the learning base are used.

Prediction. Once the estimators $\hat{\sigma}^2, \hat{\boldsymbol{\ell}}, \hat{\alpha}$ are computed, the standard "plug-in" approach [13] is to assume that the covariance function is known and equal to that obtained from the estimators, that is

$$C(\boldsymbol{x}^{(a)}, \boldsymbol{x}^{(b)}) = \hat{\sigma}^2 \left[\bar{C}_{\hat{\boldsymbol{\ell}}}(\boldsymbol{x}^{(a)} - \boldsymbol{x}^{(b)}) + \hat{\alpha}\mathbf{1}\{\boldsymbol{x}^{(a)} = \boldsymbol{x}^{(b)}\}\right].$$

We make this assumption, which enables to construct the Kriging metamodel of f_{code} as follows. Let $\mathbf{R}$ be a shorthand for $\mathbf{R}_{\hat{\boldsymbol{\ell}},\hat{\alpha}}$. Let, for $\boldsymbol{x} \in \mathcal{D}$, $\boldsymbol{r}(\boldsymbol{x})$ be the $n \times 1$ vector defined by $(r(\boldsymbol{x}))_i = \bar{C}_{\hat{\boldsymbol{\ell}}}(\boldsymbol{x} - \boldsymbol{x}^{(i)}) + \alpha\mathbf{1}\{\boldsymbol{x} = \boldsymbol{x}^{(i)}\}$. Then, conditionally to $\boldsymbol{y}$, $Y(\boldsymbol{x})$ follows a Gaussian distribution with mean

$$\hat{f}_{code}(\boldsymbol{x}) = \boldsymbol{r}(\boldsymbol{x})^t \mathbf{R}^{-1} \boldsymbol{y}, \tag{7}$$

and variance

$$\hat{\sigma}^2_{code}(\boldsymbol{x}) = \hat{\sigma}^2 \left(1 + \hat{\alpha} - \boldsymbol{r}(\boldsymbol{x})^t \mathbf{R}^{-1} \boldsymbol{r}(\boldsymbol{x})\right). \tag{8}$$

In the above display, $\hat{f}_{code}(\boldsymbol{x})$ is the metamodel function of f_{code}, that can be compared with those obtained from the artificial neural network and kernel regression methods. The quantity $\hat{\sigma}^2_{code}(\boldsymbol{x})$, that we call the predictive variance, is however specific to Kriging. It is one of the benefits of considering a Gaussian process model for f_{code}. The predictive variance can be used, for instance, to construct the confidence interval $[\hat{f}_{code}(\boldsymbol{x}) - 1.65\hat{\sigma}_{code}(\boldsymbol{x}), \hat{f}_{code}(\boldsymbol{x}) + 1.65\hat{\sigma}_{code}(\boldsymbol{x})]$ that contains $Y(\boldsymbol{x})$ with probability 0.9.

Note that, for any $\boldsymbol{x}$ which does not belong to $\{\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(n)}\}$, we have

with the notation of (2), and where $\mathbb{E}$ denotes the expected value,

$$\hat{\sigma}^2_{code}(\boldsymbol{x}) = \mathbb{E}\left((\hat{f}_{code}(\boldsymbol{x}) - Y_c(\boldsymbol{x}))^2\right) + \mathbb{E}(Y_d(\boldsymbol{x})^2) = \mathbb{E}\left((\hat{f}_{code}(\boldsymbol{x}) - Y_c(\boldsymbol{x}))^2\right) + \hat{\delta}^2. \tag{9}$$

This is interpreted as follows: The value of the discontinuous Gaussian process $Y_d(\boldsymbol{x})$ in (2) can not be inferred from the values of $Y_d(\boldsymbol{x}^{(1)}), ..., Y_d(\boldsymbol{x}^{(n)})$ (predicting $Y_d(\boldsymbol{x})$ by 0 is in fact the best possibility). Hence, the prediction mean square error $\hat{\sigma}^2_{code}(\boldsymbol{x})$ for $Y(\boldsymbol{x})$ is larger than $\hat{\delta}^2$ and the difference between $\hat{\sigma}^2_{code}(\boldsymbol{x})$ and $\hat{\delta}^2$ corresponds to the prediction error for $Y_c(\boldsymbol{x})$, which is the continuous component of $Y(\boldsymbol{x})$. Thus, in practice, the square prediction error for the code function $f_{code}(\boldsymbol{x})$ should be on average larger than $\hat{\delta}^2$.

We conclude this presentation of Kriging with the virtual cross validation formulas [18, 16]. Consider $\hat{\sigma}^2, \hat{\boldsymbol{\ell}}, \hat{\alpha}$ to be estimated from the learning base $y_1 = f_{code}(\boldsymbol{x}^{(1)}), ..., y_n = f_{code}(\boldsymbol{x}^{(n)})$ and fixed. Then, let $\hat{f}_{code,LOO}(\boldsymbol{x}^{(i)})$ and $\hat{\sigma}^2_{code,LOO}(\boldsymbol{x}^{(i)})$ be the Leave-One-Out (LOO) prediction and predictive variance for $f_{code}(\boldsymbol{x}^{(i)})$, that would be obtained from (7) and (8) if $\boldsymbol{x}^{(i)}$ and $f_{code}(\boldsymbol{x}^{(i)})$ were removed from the learning base. Then we have, for $1 \leq i \leq n$,

$$f_{code}(\boldsymbol{x}^{(i)}) - \hat{f}_{code,LOO}(\boldsymbol{x}^{(i)}) = \frac{1}{(\mathbf{R}^{-1})_{i,i}}(\mathbf{R}^{-1}\boldsymbol{y})_i \tag{10}$$

and

$$\hat{\sigma}^2_{code,LOO}(\boldsymbol{x}^{(i)}) = \frac{1}{(\mathbf{R}^{-1})_{i,i}}. \tag{11}$$

Hence, the n LOO errors and predictive variances can be computed by means of a single $n \times n$ matrix inversion, while a naive approach, consisting in evaluating n different versions of (7) and (8), would necessitate to solve n

linear systems of size $(n-1) \times (n-1)$.

3.2 Kernel methods

Kernel methods [19, 20] are frequently used for statistical learning and metamodeling. The kernel metamodel eventually yields prediction formula similar to Kriging (compare (7) and (13)), although the philosophy is different.

Kernel methods, for inputs in a domain $\mathcal{D} \subset \mathbb{R}^d$, are based on a symmetric nonnegative definite kernel function $k : (\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}^2 \to \mathbf{R}$, see [20]. This kernel function defines a Hilbert space $\mathcal{H}_k$ of functions from $\mathcal{D}$ to $\mathbb{R}$, that is called the Reproducing Kernel Hilbert Space (RKHS) corresponding to the kernel function (see [20] for details).

Consider now the learning base $(\boldsymbol{x}^{(1)}, y_1 = f_{code}(\boldsymbol{x}^{(1)})), ..., (\boldsymbol{x}^{(n)}, y_n = f_{code}(\boldsymbol{x}^{(n)}))$. Then, for each $\lambda \geq 0$, that we call the regularity parameter, we can consider the function $\hat{f}_\lambda \in \mathcal{H}_k$ which solves

$$\hat{f}_\lambda = \arg\min_{f \in \mathcal{H}_k} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(\boldsymbol{x}_i))^2 + \lambda ||f||^2_{\mathcal{H}_k}, \tag{12}$$

where $||f||_{\mathcal{H}_k}$ is a complexity measure for the function f, see [20]. Thus, the aim is that the function $\hat{f}_\lambda$ both reproduce well the observations y_i and be of small complexity, in order to prevent overfitting. Increasing the value of λ prevents overfitting all the more.

It turns out that the abstract optimization problem (12) has an explicit solution that is computable in practice. Let $\mathbf{R}_\lambda$ be the $n \times n$ matrix defined by $(\mathbf{R}_\lambda)_{i,j} = k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) + n\lambda \mathbf{1}\{i = j\}$, let $\boldsymbol{r}(\boldsymbol{x})$ be the $n \times 1$ vector defined

by $r(\boldsymbol{x})_i = k(\boldsymbol{x}^{(i)}, \boldsymbol{x})$ and let $\boldsymbol{y} = (y_1, ..., y_n)^t$. Then, we have

$$\hat{f}_\lambda(\boldsymbol{x}) = \boldsymbol{r}(\boldsymbol{x})^t \mathbf{R}_\lambda^{-1} \boldsymbol{y}. \tag{13}$$

Note that, when $\lambda = 0$ and $\mathbf{R}_0$ is of full rank, we obtain an exact interpolation: $\hat{f}_0(\boldsymbol{x}^{(i)}) = y_i$. Nevertheless, using a non-zero λ enables us to deal with the small scale variations of the Germinal code (similarly to the nugget effect of the Kriging metamodel). Note also that calculating $\mathbf{R}_\lambda^{-1}\boldsymbol{y}$ is more convenient numerically when λ is large.

We select the value of the regularity parameter λ by Generalized Cross Validation (GCV) [21]. The selected λ is given by

$$\lambda_{GCV} = \arg\min_\lambda \frac{||\mathbf{R}_\lambda^{-1}\boldsymbol{y}||}{\texttt{Trace}(\mathbf{R}_\lambda^{-1})}, \tag{14}$$

where $||.||$ is the Euclidean norm. Hence, the final kernel metamodel function is $\hat{f}_{code} = \hat{f}_{\lambda_{GCV}}$. Note that the minimization problem (14) entails a negligible computation cost, since a SVD decomposition can be used, similarly to (6) for Kriging.

Because the prediction formula (7) and (13) are identical, there exist virtual LOO formulas for kernel methods, that are similar to those of Kriging in (10). By letting $\mathbf{R}$ be $\mathbf{R}_{\lambda_{GCV}}$, we have

$$f_{code}(\boldsymbol{x}^{(i)}) - \hat{f}_{code,LOO}(\boldsymbol{x}^{(i)}) = \frac{1}{(\mathbf{R}^{-1})_{i,i}} (\mathbf{R}^{-1}\boldsymbol{y})_i, \tag{15}$$

where $\hat{f}_{code,LOO}(\boldsymbol{x}^{(i)})$ is defined as in (10) but for the kernel metamodel.

Note that, since kernel methods are not based on a probabilistic model,

there are no error indicators similar to $\hat{\sigma}^2_{code}(\boldsymbol{x})$ in (8) for Kriging.

In this paper, the kernel function k we consider is defined by $k(\boldsymbol{x}, \boldsymbol{y}) = \prod_{i=1}^{d} \bar{k}(x_i, y_i)$ with

$$\bar{k}(x, y) = \sum_{l=0}^{m} \frac{1}{(l!)^2} B_l(x) B_l(y) + \frac{(-1)^{m+1}}{(2m)!} B_{2m}(|x - y|), \tag{16}$$

where B_l is the l-th Bernoulli polynomial. The benefit of this kernel function is that the corresponding RKHS $\mathcal{H}_k$ consists in the Sobolev space of functions that are m times differentiable [19, 22]. Hence m can be chosen according to the smoothness we require from the metamodel function. We choose $m = 2$ in this paper, as it provides the minimal value for (14).

3.3 Artificial neural networks

Artificial Neural Networks (ANNs) are known as efficient modeling tools to approximate nonlinear functions with the fundamental property of parsimonious approximation [23]. We carried out all computations for the neural networks with the uncertainty quantification platform URANIE [24]. We consider the Multi Layer Perceptron (MLP) [25] with one hidden layer and one output. The MLP consists of simple connections between neurons and is characterized by the number of hidden neurons and the weights of their corresponding connections.

For a given number of hidden neurons, the weights are fitted by using the standard back-propagation procedure [26]. This procedure is repeated with different weight initializations, and the eventual values of the weights are selected by cross validation. Finally, the number of hidden neurons is

selected by a minimization of the RMSE (Root Mean Square Error) on the full learning data set.

3.4 Computation times

The Kriging and neural network metamodels are used in two steps. First, in what we call the construction phase, the neural network structure and the covariance parameters for Kriging are optimized (Sections 3.1 and 3.3). This first step yields the metamodel function $\hat{f}_{code}$. Note that, beneficially, the construction phase is not needed for kernel regression. Second, in what we call the evaluation phase, for many inputs $\boldsymbol{x}$, the metamodel predictions $\hat{f}_{code}(\boldsymbol{x})$ are calculated.

With the implementation we used, for the learning base under consideration, and on a personal computer, the computation time for the construction phase is around five to ten hours for both Kriging and neural networks. Since this typically takes place only once, this time is not critical. The evaluation time is, on average, 0.00015 seconds per input $\boldsymbol{x}$ for the neural networks and 0.004 seconds for Kriging and kernel methods. Hence, neural network evaluation is faster, which is explained because the evaluation cost of the neural network metamodel function is proportional to the number of hidden neurons, while those of the Kriging or kernel regression metamodel functions are proportional to n. In our case, n is much larger than the number of hidden neurons. For the three metamodels, the evaluation times are not prohibitive for using the metamodel functions in an optimization framework, like in [5], where a few millions of metamodel evaluations would be required.

$\hat{\sigma}(^\circ)$	$\hat{\ell}_1$	$\hat{\ell}_2$	$\hat{\ell}_3$	$\hat{\ell}_4$	$\hat{\ell}_5$	$\hat{\ell}_6$	$\hat{\ell}_7$	$\hat{\ell}_8$	$\hat{\ell}_9$	$\hat{\ell}_{10}$	$\hat{\ell}_{11}$	$\hat{\delta}(^\circ)$
1264	21	50	12	4.5	12	64	100	2.2	6.6	5.9	100	28.5

Table II: Estimated covariance parameters $(\hat{\sigma}, \hat{\boldsymbol{\ell}}, \hat{\delta})$ for the Kriging metamodel of the fusion margin output of the Germinal code.

4 Prediction and classification results for the original Germinal computations

4.1 Prediction results

Estimated covariance parameters for Kriging. The estimated covariance parameters for the Kriging metamodel are presented in Table II. Note that we have applied an affine standardization of $\{\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(n)}\}$ in $[0,1]^{11}$, to obtain, for $i = 1, ..., 11$, $\min_j \boldsymbol{x}_i^{(j)} = 0$ and $\max_j \boldsymbol{x}_i^{(j)} = 1$. Hence, for the correlation length vector $\boldsymbol{\ell}$ we present, all the components are at the same scale and should be compared to inputs in $[0,1]^{11}$.

The input variables x_i with smallest estimated correlation lengths $\hat{\ell}_i$ are considered the most influential for the code function in the Kriging model. In Table II, the smallest estimated correlation length is $\hat{\ell}_8$, corresponding to the average pin power input. This is natural, since the average pin power has a strong direct influence on the power map in the fuel pin, which is intrinsically related to the temperature in the fuel pin and thus to the fusion margin. Similarly, the inputs x_9 (axial form factor) and x_{10} (power shift) impact the power map and the inputs x_3 (hole diameter) and x_4 (external clad diameter) characterize the geometry of the fuel pin. These four inputs

thus have a strong impact on the fusion margin, so that their corresponding estimated correlation lengths are also relatively small.

On the contrary, in the Kriging model, the code output is considered unaffected by the values of the input variables x_i with very large correlation lengths. [Note that in Table II, the maximum correlation length value is 100, which is the upper bound in the likelihood optimization procedure, and is practically equivalent to an infinite correlation length.] The two input variables with correlation lengths 100 are x_7 (pin height) and x_{11} (volume of expansion). Indeed, the fuel power (and thus the temperature) is not related to the pin height. Furthermore, the volume of expansion has no physical link with the temperature.

The estimated nugget variance is $\hat{\delta}^2 = (28.5°)^2$, which is a signal that code instabilities might be present, as confirmed in Section 5.2, and which indicates that the RMSE should be at least around 30°, as is confirmed below. This interpretation of the covariance parameters of Kriging is hence beneficial and constitutes and asset, in comparison with neural networks and kernel methods.

Prediction criteria. We evaluate the accuracy of the metamodels by using a test base $(\boldsymbol{x}_t^{(1)}, f_{code}(\boldsymbol{x}_t^{(1)})), ..., (\boldsymbol{x}_t^{(n_t)}, f_{code}(\boldsymbol{x}_t^{(n_t)}))$, that is generated independently from and in the same way as the learning base, with $n_t = 1613$.

The first criterion we consider is the Root Mean Square Error on the test base (RMSE), with $\hat{f}_{code}(\boldsymbol{x})$ the prediction of $f_{code}(\boldsymbol{x})$, obtained from the

artificial neural network, Kriging or kernel methods,

$$RMSE^2 = \frac{1}{n_t}\sum_{i=1}^{n_t}\left(\hat{f}_{code}(\boldsymbol{x}_t^{(i)}) - f_{code}(\boldsymbol{x}_t^{(i)})\right)^2. \tag{17}$$

A second criterion is the Q^2 (considered for instance in [27]) defined by

$$Q^2 = 1 - \frac{RMSE^2}{sd_{code}^2}, \tag{18}$$

where sd_{code} is the standard deviation of the output on the test base (the standard deviation of $\{f_{code}(\boldsymbol{x}_t^{(1)}), ..., f_{code}(\boldsymbol{x}_t^{(n_t)})\}$). The Q^2 is thus a relative efficiency criterion, which value is always smaller than 1 and increases with the accuracy of the predictions.

The criteria $RMSE$ and Q^2 are not observable in practice, but can be estimated from the learning base. In order to do so, let $\tilde{f}_{code}(\boldsymbol{x}^{(i)})$ be the prediction $\hat{f}_{code}(\boldsymbol{x}^{(i)})$ of $f_{code}(\boldsymbol{x}^{(i)})$ obtained from the artificial neural network, or the LOO prediction $\hat{f}_{code,LOO}(\boldsymbol{x}^{(i)})$ of $f_{code}(\boldsymbol{x}^{(i)})$ with Kriging or kernel methods. Then, $RMSE$ and Q^2 can be estimated by $\widehat{RMSE}$ and $\widehat{Q}^2$, defined by

$$\widehat{RMSE}^2 = \frac{1}{n}\sum_{i=1}^{n}\left(\tilde{f}_{code}(\boldsymbol{x}^{(i)}) - f_{code}(\boldsymbol{x}^{(i)})\right)^2 \tag{19}$$

and

$$\widehat{Q}^2 = 1 - \frac{\widehat{RMSE}^2}{\widehat{sd}_{code}^2}, \tag{20}$$

where $\widehat{sd}_{code}$ is the standard deviation of the output on the learning base.

Then, for $\gamma \in (0,1)$, we define the criterion q_γ as the empirical quantile γ of the set of errors $\left|\hat{f}_{code}(\boldsymbol{x}_t^{(i)}) - f_{code}(\boldsymbol{x}_t^{(i)})\right|$, for $i = 1, ..., n_t$.

Finally, one specificity of Kriging is that it provides the predictive variance (8) which enables to build predictive confidence intervals for the code values $f_{code}(\boldsymbol{x})$. To assess the accuracy of the 90%-confidence intervals presented after (8), we consider the following Confidence Interval Ratio (CIR), defined as

$$CIR = \frac{1}{n_t} \sum_{i=1}^{n_t} \mathbf{1}\{|\hat{f}_{code}(\boldsymbol{x}_t^{(i)}) - f_{code}(\boldsymbol{x}_t^{(i)})| \leq 1.64\hat{\sigma}_{code}(\boldsymbol{x}_t^{(i)})\}. \qquad (21)$$

The CIR criterion is specific to Kriging and should be close to 0.9.

Prediction results. The prediction results are given in Table III. The standard deviation of the output on the test base is $sd_{code} = 342°$, and the RMSE for the neural network, Kriging and kernel methods are respectively 38.5°, 36.1° and 44.5°. The relative prediction errors are thus around 10%, which is a good performance considering the complexity of the fusion margin output. Similarly, the relative efficiency criteria Q^2 are around 99% for the three metamodels. Kriging provides slightly more accurate predictions than the neural networks, and these two metamodels perform better than kernel methods. The same hierarchy holds when we consider the quantiles $q_{0.9}$ and $q_{0.95}$ of the absolute prediction errors.

The Kriging estimate of the nugget variance is $\hat{\delta}^2 = (28.5°)^2$. As is seen in Section 3.1, under the Gaussian process assumption of Kriging, this value corresponds to the irreducible prediction error for $f_{code}(\boldsymbol{x})$, stemming from the small scale variations of f_{code} which are due to code instabilities. Hence, a large part of the prediction errors of the metamodels comes from these code instabilities.

For Kriging and kernel methods, $\widehat{RMSE}$ is a very reliable estimate of

	$\widehat{RMSE}$	RMSE	$\widehat{Q}^2$	Q^2	$q_{0.9}$	$q_{0.95}$
Neural network	34.5°	38.5°	0.990	0.987	61.6°	76.7°
Kriging	35.6°	36.1°	0.989	0.989	57.4°	72.7°
Kernel methods	44.3°	44.5°	0.983	0.983	68.5°	88.8°

Table III: Prediction results for the fusion margin output of the Germinal code (original computations). The standard deviation of the output on the test base is 342°. The quantity $RMSE$ and Q^2 are error and efficiency criteria for prediction on the test base. They are estimated by $\widehat{RMSE}$ and $\widehat{Q}^2$ that use the learning base, see (19) and (20). The estimates $\widehat{RMSE}$ and $\widehat{Q}^2$ are more accurate for Kriging and kernel methods than for the neural networks, thanks to the virtual LOO formulas.

RMSE, while $\widehat{RMSE}$ is moderately too optimistic for the neural networks, as it is smaller than RMSE. Indeed, the neural network functions are optimized according to their prediction errors on the learning base, so that these errors are eventually slightly smaller than the new errors on the test base. For Kriging and kernel methods, the LOO precisely avoids this phenomenon, by providing prediction errors for outputs $f_{code}(\boldsymbol{x}^{(i)})$ that are removed from the learning base. The 90% confidence intervals provided by Kriging are also appropriate, as they contain 89.8% of the output values in the test base (CIR = 89.8% in (21)).

In Figure 3, we plot the predictions as a function of the Germinal output values. For the three metamodels, the predictions are less accurate when the fusion margin is negative or close to negative. Indeed, this corresponds to complex physical processes, that are challenging to simulate numerically, as discussed in Section 2. Some of the prediction errors are particularly large

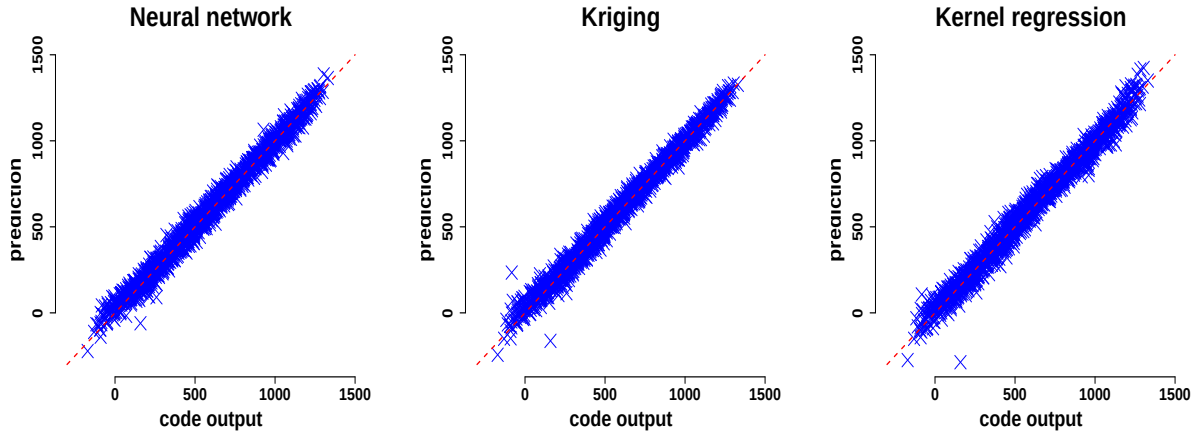


Figure 3: Plot of the metamodel predictions in the test base (y-axis), as a function of the Germinal output values (x-axis), for the neural networks (left), Kriging (middle) and kernel methods (right). Case of the original computations. The dashed lines is defined by $y = x$.

and stand out in Figure 3. We call outliers these Germinal computations that are poorly predicted, and give more comments on their detection in the learning base and their analysis in section 5.

4.2 Classification

Classification goal and classifiers. In practice, a simulated fuel pin (characterized by $\boldsymbol{x}$) is considered viable if the fusion margin output ($f_{code}(\boldsymbol{x})$) is larger than 300°. This value of 300° is a security margin, accounting for the possible discrepancies between a Germinal simulation and a real utilization of a fuel pin.

Hence, besides predicting the fusion margin output $f_{code}(\boldsymbol{x})$, it is desirable to classify the inputs $\boldsymbol{x}$ in two classes: Those which are viable ($f_{code}(\boldsymbol{x}) > 300°$) and those which are unsafe ($f_{code}(\boldsymbol{x}) \leq 300°$). Furthermore, the two

possible corresponding classification errors are not symmetric, so that it is very beneficial to have a tunable classifier, that can for example decrease the number of unsafe $\boldsymbol{x}$ that are classified as viable, at the cost of increasing the number of viable $\boldsymbol{x}$ that are classified as unsafe.

This tuning can be achieved naturally with metamodels. Let $\hat{f}_{code}(\boldsymbol{x})$ be the metamodel prediction at $\boldsymbol{x}$ and let $\hat{\sigma}^2_{code}(\boldsymbol{x})$ be as in (8) for Kriging and be $\widehat{RMSE}$ for neural networks and for kernel methods. Then, we consider the classifier, tuned by the parameter $\tau \in \mathbb{R}$, that classifies $\boldsymbol{x}$ as unsafe if

$$\hat{f}_{code}(\boldsymbol{x}) - \tau \hat{\sigma}_{code}(\boldsymbol{x}) \tag{22}$$

is smaller than 300°, and classifies $\boldsymbol{x}$ as viable otherwise. The user can give a large value to τ, if he considers that classifying as viable an unsafe $\boldsymbol{x}$ is more harmful than classifying as unsafe a viable $\boldsymbol{x}$, and a small value to τ otherwise.

Classification Results. We present the classification results of the three metamodels in the form of their Receiver Operating Characteristic (ROC) curves (see e.g. [28, Ch.11.16]). For fixed τ and for each classifier, we define the "true unsafe rate" as the ratio, on the test base, of the number of $\boldsymbol{x}$ that are unsafe and classified as unsafe, divided by the number of $\boldsymbol{x}$ that are unsafe (385). We also define the the "false unsafe rate" as the ratio, on the test base, of the number of $\boldsymbol{x}$ that are viable and classified as unsafe, divided by the number of $\boldsymbol{x}$ that are viable (1228). Thus, selecting an increasing sequence of values of τ yields an increasing sequence of "false unsafe rate"

values and an increasing sequence of "true unsafe rate" values. Plotting the second sequence as a function of the first constitutes a ROC curve. The higher this curve is, the better the classifier is, since the "true unsafe rate" is larger, for a given "false unsafe rate".

The ROC curves for the three metamodels are presented in Figure 4. The classification results are good, since one can, for example, achieves more than 95% "true unsafe rate" for less than 5% "false unsafe rate". The three ROC curves are difficult to compare visually, since depending on the value of the "false unsafe rate", any of them can be above the others. Nevertheless, the values of the area under the ROC curves [28, Ch.11.16], are 0.9977 for Kriging, 0.9974 for the neural networks and 0.9972 for kernel methods, which gives the same accuracy ranking as for the prediction errors.

5 Improvement of code behaviour

5.1 Outlier detection

As shown in Figure 3, for some intputs $\boldsymbol{x}_t^{(j)}$ in the test base, the corresponding Germinal output values $f_{code}(\boldsymbol{x}_t^{(j)})$ are predicted with particularly large errors. As we show below, similar couples $(\boldsymbol{x}^{(j)}, f_{code}(\boldsymbol{x}^{(j)}))$, that we call outliers, exist in the learning base.

To detect outliers in the learning base, we define the normalized prediction error at $\boldsymbol{x}^{(j)}$ as

$$\frac{\left|\tilde{f}_{code}(\boldsymbol{x}^{(j)}) - f_{code}(\boldsymbol{x}^{(j)})\right|}{\tilde{\sigma}_{code}(\boldsymbol{x}^{(j)})}, \tag{23}$$

where $\tilde{f}_{code}(\boldsymbol{x})$ is defined for the three metamodels as in (19), and where

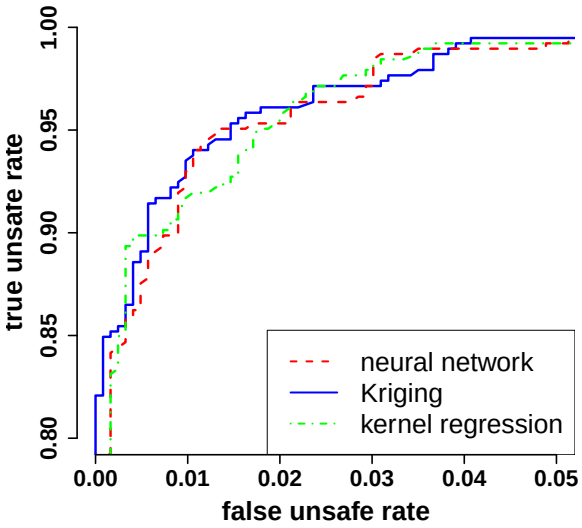


Figure 4: Plot of the "true unsafe rate" as a function of the "false unsafe rate", for varying values of the tuning parameter τ in (22). The total number of unsafe $\boldsymbol{x}$ is 385 and the total number of viable $\boldsymbol{x}$ is 1228. The areas under the ROC curves are 0.9977 for Kriging, 0.9974 for the neural networks and 0.9972 for kernel methods.

$\tilde{\sigma}_{code}(\boldsymbol{x}^{(j)})$ is defined as $\widehat{RMSE}$ for neural networks and kernel methods and as in (11) for Kriging. For neural networks and kernel methods, the normalization term $\tilde{\sigma}_{code}(\boldsymbol{x}^{(j)}) = \widehat{RMSE}$ is so that the average of the squares of (23), for $j = 1, ..., n$, is 1. Under the Gaussian process assumption, the Kriging normalized errors (23) follow the standard Gaussian distribution. [Note that these errors are however not independent in general.]

The normalized errors are presented in Figure 5. For the three metamodels, two particularly large normalized errors stand out. The other remaining errors are homogeneous and considerably smaller. In addition, for the three metamodels, these two largest errors correspond to the same computations $(\boldsymbol{x}^{(j)}, f_{code}(\boldsymbol{x}^{(j)}))$. Thus, these two computations are detected as outliers and should be thoroughly studied. The other computations with large normalized errors could possibly benefit from a more detailed investigation, but this is less of a priority.

One should explain why the normalized prediction errors for the two outliers are so large. We think that there are, in general, three possible causes for large prediction errors. The metamodels can be imperfectly specified, the input $\boldsymbol{x}^{(j)}$ for the outlier can be isolated in the learning base, and the output $f_{code}(\boldsymbol{x}^{(j)}))$ can stem from a computation of the Germinal code that have failed, so that the value $f_{code}(\boldsymbol{x}^{(j)}))$ does not make sense from a physical point of view and is very different from the values $f_{code}(\boldsymbol{x}^{(k)})$, for inputs $\boldsymbol{x}^{(k)}$ in the learning base that are close to $\boldsymbol{x}^{(j)}$.

These three causes can and should be addressed: Metamodels can be questioned and improved (for example the choice of the covariance function in Kriging or the structure of the neural network can be updated). New

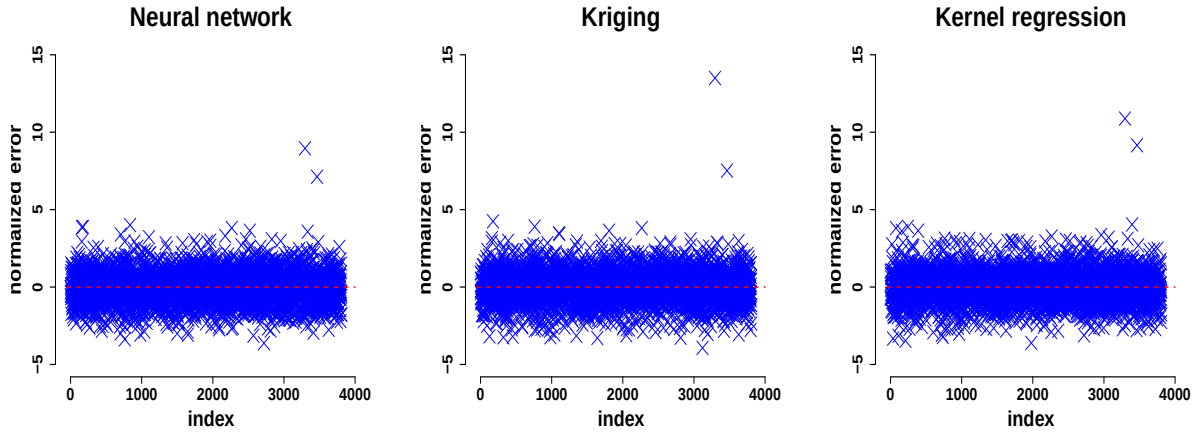


Figure 5: Plot of the normalized prediction errors (23) on the learning base (y-axis), as a function of the computation index (x-axis), for neural networks (left), Kriging (middle) and kernel methods (right). Case of the original computations. Two outliers stand out and correspond to the same computations for the three metamodels.

computations of the code can be made for inputs $\boldsymbol{x}$ in areas of the input space $\mathcal{D}$ that are insufficiently covered. Finally, suspicious computations can be checked manually. For the two aforementioned outlier computations, this third solution is appropriate.

Indeed, a detailed analysis of the two computation output files for the two aforementioned outliers shows that the same specific warning message was given for both. This message appears in only 14 of the remaining computations of the learning base. Furthermore, computations with input variables similar to those of the two outliers show drastically different output values Hence, we can conclude that this warning message implies much more serious consequences on the computation result than indicates its current description ("relatively" bad numerical behaviour).

In light of this analysis, we update the code postprocessor by giving

a more important weight to this warning message. Computations which output files contain it are now labeled as failures and are not incorporated in the learning base. In Section 5.2, we show how the postprocessor can also be updated, and we present the metamodel prediction results for the corresponding improved code manager in Section 6.

Note that, even though our analysis indicated that the computations have failed numerically for the two outliers, it would be very challenging and time consuming to point out the exact nature of the failure.

5.2 Reduction of the code instabilities

As discussed in Section 4.1, the relatively large nugget effect estimated by Kriging (28.5°) is a sign of code instabilities. [Note, in comparison, that the designed numerical approximations of Germinal, including for instance rounding of values, are around 1°.] To obtain graphical information on these instabilities, we run 97 additional computations, which input points are located along a line segment of the (normalized) input space $[0,1]^{11}$, and can hence be ordered. This provides us a one-dimensional visualization of the Germinal code function that we show in Figure 6.

In Figure 6, we observe oscillations of the code response, that can clearly not correspond to the modelled physical process (typically assumed to entail piecewise differentiable functions), and are hence code instabilities. [Note that we also observe two outlier computations, which output files actually contain the same warning as that described in Section 5.1.] For the sake of illustration, we also show the prediction and 95% confidence intervals ob-

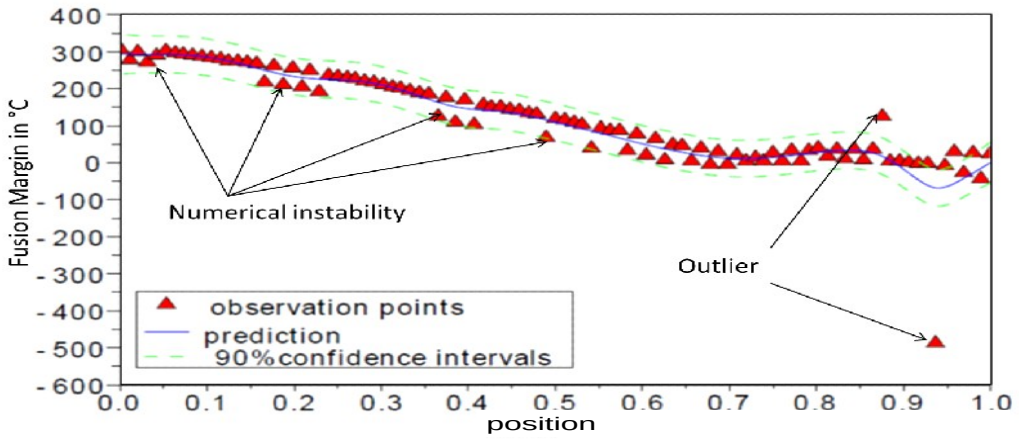


Figure 6: One-dimensional representation of the Germinal code function. We run 97 computations, which input points are located along a line segment, between two points $\boldsymbol{a}$ and $\boldsymbol{b}$ of the (normalized) input space $[0,1]^{11}$. Input points are thus indexed by their position on the segment, where 0 corresponds to $\boldsymbol{a}$ and 1 corresponds to $\boldsymbol{b}$. We observe a code instability, causing oscillations of the code response, that have no physical meaning but are caused by a preprocessing issue. We also show the prediction and 95% confidence intervals obtained with the Kriging estimated covariance parameters of Table II, where the support points of (7) and (8) correspond to these 97 computations. The output files of the two outlier computations contain the same warning as that described in Section 5.1.

tained with the Kriging estimated covariance parameters of Table II, where the support points (used to construct $\boldsymbol{r}(\boldsymbol{x})$, $\mathbf{R}$ and $\boldsymbol{y}$) of (7) and (8) correspond to the aforementioned 97 additional computations. We see that the covariance parameters estimated by Kriging, and in particular the nugget effect, are appropriate and adapted to the code instabilities, and entail satisfactory prediction and confidence intervals.

We investigated closely consecutive computations in the code instability zones of Figure 6. We found out that the code preprocessor generates automatically an axial mesh from a global pin height. A small variation of the pin height changes the fusion margin only moderately but may change the location of the maximum-temperature space point (the physical hot point) much more significantly. As illustrated in Figure 7, with the current mesh method, a computation point (a mesh node) coinciding with the physical hot point can shift away from it with a small pin height variation, thus yielding a computation of the fusion margin that is numerically (and erroneously) overly different.

Consequently, we updated the preprocessor, as illustrated in Figure 7. Together with the previously discussed postprocessor update, this yields an updated version of the Germinal code manager, that we used to generate new output values for the inputs of Figure 6. Figure 8 shows that the code instabilities have been corrected in the new code version. Hence, eventually, the Kriging nugget effect helps detecting code instabilities that can then be investigated and corrected.

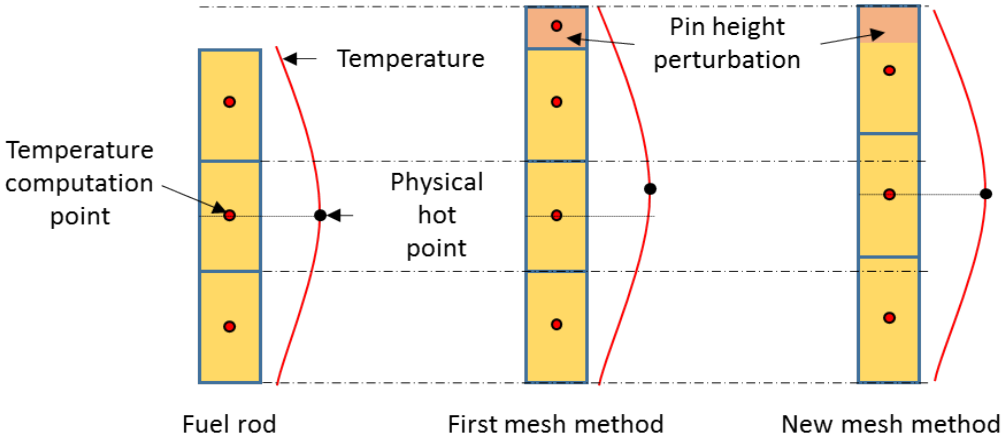


Figure 7: Simplified illustration of the impact of a perturbation of the pin height input on the mesh generation for a Germinal computation. In the original Germinal computations, mild changes of the pin height can cause significant modifications of the mesh, themselves causing the code instabilities of Figure 6. We consequently updated the preprocessor to solve this issue.

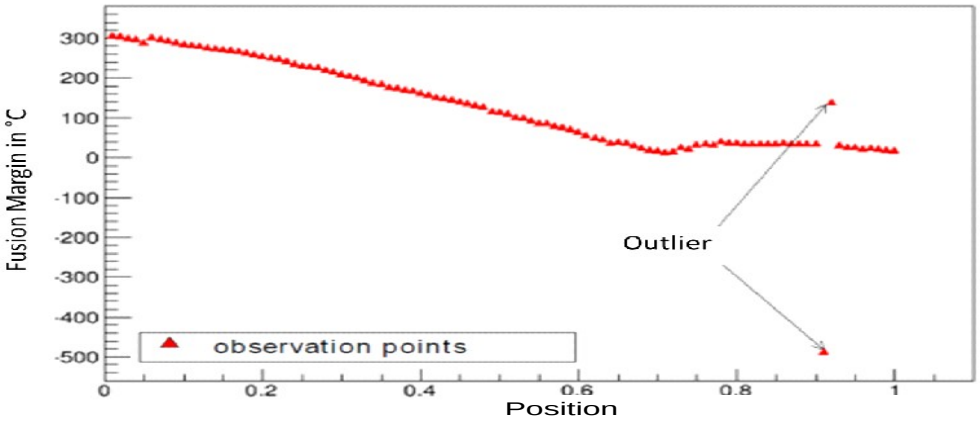


Figure 8: Same settings as in Figure 6, but for the updated Germinal computations. The code instabilities have been corrected.

6 Prediction and classification results for the updated Germinal computations

6.1 Prediction results

A discussed in Section 5, the Germinal code manager has been updated. We have repeated all the Germinal computations for the input points of the original learning and test bases of Section 4. These inputs, together with the new output values, correspond to updated learning and test bases that we use in this section. Because the postprocessor has been updated, additional computations are flagged as failure (those presenting the warning message discussed in Section 5.1), so that the updated learning and test bases have 3791 and 1606 points. The metamodels are then used exactly as for the original learning and test bases.

Estimated covariance parameters for Kriging. For the updated learning base, the estimated correlation lengths of the Kriging metamodel are similar to those for the original learning base. However, the estimate of the nugget variance $\hat{\delta}^2$ decreases significantly from the original to the updated learning base, going from $(28.5°)^2$ (Table II) to $(19.8°)^2$. This is a sign that the pre- and post-treatment procedures for the Germinal code have been improved, as is illustrated by Figure 8. Nevertheless, since the nugget variance remains significant, we believe that pre- and post-treatment issues might remain. The prediction results presented below are in agreement with this discussion.

	$\widehat{RMSE}$	RMSE	$\hat{Q}^2$	Q^2	$q_{0.9}$	$q_{0.95}$
Neural network	27.5°	31.3°	0.993	0.991	48.7°	63.4°
Kriging	27.2°	27.6°	0.993	0.993	43.2°	54.0°
Kernel methods	38.3°	38.5°	0.986	0.986	60.8°	75.3°

Table IV: Same context as for Table III. The standard deviation of the output on the test base is 326.2°. The estimates $\widehat{RMSE}$ and $\hat{Q}^2$ are more accurate for Kriging and kernel methods than for the neural networks, thanks to the virtual LOO formulas.

Prediction results. The prediction results, for the updated learning and test bases, are given in Table IV. The standard deviation of the output on the test base is 326°, and the RMSE for neural networks, Kriging and kernel methods are respectively 31.3°, 27.6° and 38.5°. Hence, the prediction errors of the metamodels are smaller than for the original computations in Section 4, but remain of comparable order of magnitude. This observation, together with the updated estimate of the nugget variance, indicates that the code instabilities have been reduced but not suppressed.

As for the original computations, Kriging gives the smallest RMSE, followed by neural networks and kernel methods. The quantity $\widehat{RMSE}$ is almost a perfect estimator of RMSE for Kriging and kernel methods and is again slightly too optimistic for neural networks. The 90% confidence intervals provided by Kriging are also appropriate, as they contain 91.2% of the output values in the test base (CIR = 91.2% in (21)).

6.2 Classification

The ROC curves for the three metamodels for the updated computations are presented in Figure 9, where we also re-plot the ROC curve of Figure 4 (original computations) for comparison. We observe that, in line with the prediction improvement in Table IV, the ROC curves are higher for the updated computations so that the classifiers perform better. Similarly, the area under the ROC curves are now 0.9984 for Kriging, 0.9980 for neural networks and 0.9978 for kernel methods, so that the three classifiers have improved performances compared to the original computations. For the updated computations, the ROC curve of Kriging is more clearly above the ROC curves of neural networks and kernel methods.

Note finally that, from Figure 9 and Table IV, Kriging performs better, in comparison to neural networks and kernel methods, for the updated computations than for the original ones. Indeed, first, the ratios of the RMSE of Kriging divided by the RMSE of neural networks and kernel methods are smaller in Table IV than in Table III. Second, the Kriging ROC curve becomes clearly higher than the two other ones for the updated computations. Similarly, the ratio of the RMSE of neural networks divided by the RMSE of kernel methods is smaller in Table IV than in Table III .

Hence, the relative differences between the three metamodel prediction errors are more accentuated for the new computations than for the old computations. We believe that this holds because of the decrease of the code instabilities in the new computations. Indeed, intuitively, the code instabilities cause systematic prediction errors, stemming from the fact that the

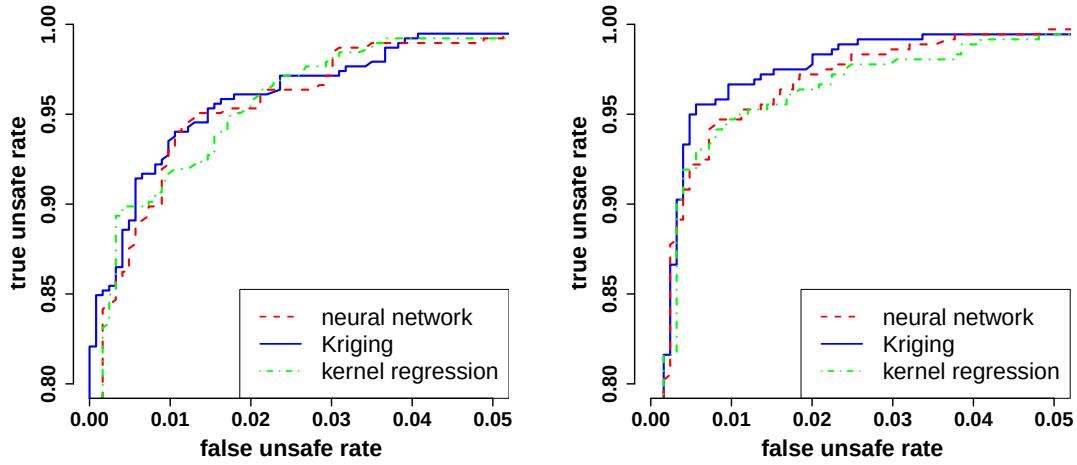


Figure 9: Plot of the "true unsafe rate" as a function of the "false unsafe rate", for varying values of the tuning parameter τ in (22), for the original computations (left) and the updated computations (right). The total number of unsafe $\boldsymbol{x}$ is 385 (left) and 359 (right), and the total number of viable $\boldsymbol{x}$ is 1228 (left) and 1247 (right). For the updated computations, Kriging becomes more accurate than neural networks and kernel methods.

brusque changes of $f_{code}(\boldsymbol{x})$ for very small changes of $\boldsymbol{x}$ are not predictable. [The square of these prediction errors have values $\hat{\delta}^2$ on average under the Gaussian process model, see (9).] These systematic errors are the same for the three metamodels, so that, when they become large, the ratios of prediction errors between different metamodels become closer to one.

7 Conclusion

Many studies in nuclear engineering, such as optimal conception, require an extensive use of computer codes, for many different input conditions. In order to limit the computation time, computer codes are replaced by metamodels, that provide approximations of the code output values, for a much cheaper computational cost.

In this paper, we present a detailed case study of the metamodeling of the fusion margin output of the Germinal code, in the case of the thermo-mechanical simulation of a fuel pin under irradiation. We compare the metamodels obtained from neural networks, Kriging and kernel methods. In our study, the computation time for metamodel evaluation is similar for Kriging and kernel methods and is the smallest for neural networks. The most accurate predictions are obtained from Kriging, followed by those obtained from neural networks, and finally by those obtained from kernel methods. Kriging and kernel methods provide the most reliable estimates of their prediction errors. This is thanks to the Leave-One-Out formula, which are not directly available for the neural networks. Kriging also arguably provides the most interpretability, with the underlying Gaussian process model and the covari-

ance parameters. The kernel methods are the simplest to implement, and the fit of the corresponding metamodel is the fastest.

Beyond this comparison, we demonstrate the pertinence of these three metamodeling techniques to improve the behaviour of the Germinal code in a design of experiments. Indeed, as many simulation codes, the Germinal code is conceived to be used for a limited number of specific situations, in which experts in physics or numerical simulation dedicate a consequent time to specify the simulation conditions and to interpret the results. In a design of experiments, where here thousands of simulations are carried out, an automatic code manager, consisting in pre- and post-processing scripts, has to replace this human intervention. Hence, specific problems and errors arise in the use of this code manager, that metamodeling techniques can detect, quantify and contribute to correct.

In the case study we address, we distinguish two types of issues related to the use of a code manager. First, some of the simulations in the design of experiments can be plagued by numerical flaws, that are not flagged by the code manager and which cause the simulation outputs to be meaningless. These meaningless simulation results are well-detected by the metamodels: In our study, the three metamodels detect the same two simulations as doubtful, and a human intervention indeed confirm that computational failures occurred. This property of the metamodels to rank the simulations according to statistical estimates of their reliability is very attractive. Indeed, it is not possible to check manually all the simulations that are carried out, but it is possible to do so for a few simulations that are automatically detected.

The second issue related to the use of a code manager is the instability of

the preprocessing step. In our study, we have analysed that changing input conditions very slightly can cause a non-negligible change in the preprocessing step (e.g. a significantly different mesh), itself causing a significant change in the simulation result. This code instability is problematic because it increases the prediction errors of the metamodels. We find that the estimate of the nugget variance provided by Kriging is particularly efficient for detecting and quantifying code instability. Especially, this nugget variance estimate decreases between the original and updated Germinal computations, which coincides with an improvement of the preprocessing step, itself enabling more accurate predictions for the three metamodels. Nevertheless, the nugget variance remains non-negligible for the updated Germinal computations, which is a signal that the code manager can still be improved.

Once the global presence of code instabilities is detected and quantified, we consider as a rather open problem the question of using metamodels to help code experts to solve them. In this study, we have proposed to carry out computations in a segment of the input space (see Figure 6), in order to have visual information on the code instabilities. This method enables us to detect pairs of very close input conditions yielding significantly different simulation results and to investigate them in details. It would be interesting to see if metamodels can provide more automatic tools to isolate such pairs of input conditions automatically.

Finally, we believe that the above-described issues, arising from the use of the Germinal code in a design of experiments, also occur in a large variety of situations in numerical simulation, in which codes are used automatically for a large number of different simulation conditions.

Acknowledgements

References

[1] H. Golfier, R. Lenain, C. Calvin, J.J. Lautard, A.M. Baudron, P.H. Fougeras, P.H. Magat, E. Martinolli, and Y. Dutheillet. APOLLO3: a common project of CEA, AREVA and EDF for the development of a new deterministic multi-purpose code for core physics analysis. In *Int Conf. on Math., Computational Meth., M&C2009, New York, USA*, 2009.

[2] G. Geffraye, O. Antoni, M. Farvacque, D. Kadri, G. Lavialle, B. Rameau, and A. Ruby. CATHARE 2 V2.5_2: A single version for various applications. *Nuclear Engineering and Design*, 241(11):4456–4463, 2011.

[3] E. Hourcade, X. Ingremeau, P. Dumaz, S. Dardour, D. Schmitt, and S. Massara. Innovative methodologies for fast reactor core design and optimization. In *ICAPP Nice*, 2011.

[4] L. Roche and M. Pelletier. Modelling of the thermomechanical and physical processes in FR fuel pins using the GERMINAL code. In *MOX Fuel Cycle Technologies for Medium and Long Term Deployment*, page 322, 2000.

[5] E. Hourcade, F. Jasserand, K. Ammar, and C. Patricot. SFR core design a system-driven multi-criteria core optimisation exercice with TRIAD. In *FR13 Paris*, 2013.

[6] D.G. Cacuci. *Sensitivity and uncertainty analysis. Theory.* Chapman & Hall/CRC, Boca Raton, FL, 2003.

[7] D.G. Cacuci and M. Ionescu-Bujor. Best-Estimate model calibration and prediction through experimental data assimilation-I: Mathematical framework. *Nuclear Science and Engineering*, 165:18–44, 2010.

[8] D. Higdon, M. Kennedy, J.C. Cavendish, J.A. Cafeo, and R.D. Ryne. Combining field data and computer simulations for calibration and prediction. *SIAM Journal on Scientific Computing*, 26:448–466, 2004.

[9] J.C. Le Pallec, C. Poinot-Salanon, N. Crouzet, and S. Zimmer. HEMERA V2: An evolutionary tool for PWR multi-physics analysis in salome platform. In *Proceedings of ICAPP 2011, France*, page 2851, 2011.

[10] T.J Santner, B.J Williams, and W.I Notz. *The Design and Analysis of Computer Experiments.* Springer, New York, 2003.

[11] B. A. Lockwood and M. Anitescu. Gradient-enhanced universal Kriging for uncertainty propagation. *Nuclear Science and Engineering*, 170:168–195, 2012.

[12] F. Bachoc, G. Bois, J. Garnier, and J.M. Martinez. Calibration and improved prediction of computer models by universal Kriging. *Nuclear Science and Engineering*, 176(1):81–97, 2014.

[13] M.L Stein. *Interpolation of Spatial Data: Some Theory for Kriging.* Springer, New York, 1999.

[14] C.E. Rasmussen and C.K.I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, Cambridge, 2006.

[15] I. Andrianakis and P. G. Challenor. The effect of the nugget on Gaussian process emulators of computer models. *Computational Statistics and Data Analysis*, 56:4215–4228, 2012.

[16] F. Bachoc. Cross validation and maximum likelihood estimations of hyper-parameters of Gaussian processes with model mispecification. *Computational Statistics and Data Analysis*, 66:55–69, 2013.

[17] F. Bachoc. Asymptotic analysis of the role of spatial sampling for covariance parameter estimation of Gaussian processes. *Journal of Multivariate Analysis*, 125:1–35, 2014.

[18] O. Dubrule. Cross validation of Kriging in a unique neighborhood. *Mathematical Geology*, 15:687–699, 1983.

[19] G. Wahba. *Spline Models for Observational Data*. Society for Industrial and Applied Mathematics, 1990.

[20] B. Schölkopf and A. J. Smola. *Learning with kernels: support vector machines, regularization, optimization and beyond*. MIT Press, 2002.

[21] G. Golub, M. Heath, and G. Wahba. Generalized cross-validation as a method for choosing a good ridge parameter. *Technometrics*, 21(2):215–223, 1979.

[22] R. X. Yue and F. J. Hickernell. Robust designs for fitting linear models with misspecification. *Statistica Sinica*, 9:1053–1069, 1999.

[23] G. Dreyfus. *Neural Networks, Methodology and Applications*. Springer, 2005.

[24] F. Gaudier. URANIE: The CEA DEN uncertainty and sensitivity platform. In *Procedia - Social and Behavioral Sciences*, volume 2, pages 7660–7661, 2010.

[25] C.M. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, 1995.

[26] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning internal representation by error backpropagation. In *Parallel Distributed Processing : Explorations in the Microstructures of Cognition*, pages 318–362. MIT Press, Cambridge, 1986.

[27] A. Marrel, B. Iooss, F. Van Dorpe, and E. Volkova. An efficient methodology for modeling complex computer codes with Gaussian processes. *Computational Statistics and Data Analysis*, 52:4731–4744, 2008.

[28] S. Tufféry. *Data Mining and Statistics for Decision Making Methods*. John Wiley and Sons, 2011.