Journal of Statistical Software

June 2015, Volume 65, Issue 11. *http://www.jstatsoft.org/*

DiceDesign and DiceEval: Two R Packages for Design and Analysis of Computer Experiments

Delphine Dupuy
École des Mines
de Saint-Étienne

Céline Helbert
Institut Camille Jordan

Jessica Franco
TOTAL S.A.

Abstract

This paper introduces two R packages available on the Comprehensive R Archive network. The main application concerns the study of computer code output. Package **DiceDesign** is dedicated to numerical design of experiments, from the construction to the study of the design properties. Package **DiceEval** deals with the fit, the validation and the comparison of metamodels.

After a brief presentation of the context, we focus on the architecture of these two packages. A two-dimensional test function will be a running example to illustrate the main functionalities of these packages and an industrial case study in five dimensions will also be detailed.

Keywords: computer experiments, design of experiments, metamodeling, R.

1. Introduction

The DICE consortium (Deep Inside Computer Experiments, `http://dice.emse.fr/`) was a partnership between academic and industrial laboratories from various fields (e.g., oil industry, automotive, nuclear, etc.). DICE aimed at developing and applying statistical methods to solve problems related to computer experiments, for example uncertainty propagation through a flow simulator, global optimization of a car crash simulator or evaluation of failure probabilities in a nuclear process.

In the context of computer experiments, a computer code is used to simulate a complex phenomenon. For example, a porous media flow simulation can be used to efficiently predict oil production. The computer model implements a mathematical model and will replace laboratory or field testing. Industrial computer codes are becoming more and more realistic thanks to the improvement of numerical methods but this accuracy comes at the cost of increased computation time. For example, the simulation of a car crash can take between 15 and 30 hours on a super computer, and the simulation of oil field production can take up to several weeks. Usually, the numerical model depends on a large number of parameters, also called the input variables, to which scientific experts can assign a range or a distribution. The problem is then to quantify the impact of the variability of the input variables on the variability of the variable of interest, also called output. This is done by running a limited number of simulations (the design of experiments) representing different combinations of the inputs and building a simplified model (metamodel) of the output as a function of the inputs. In this context, the choice of the design of experiments and the determination of the metamodel are of great importance.

The output of a simulation is the response. Given the value of the response at the points of the experimental design, we want to fit a simplified model. The type of the model depends upon the nature and the objective of the problem. In high dimension, one can for example fit an additive model to visualize the shape of the curve according to each factor. For stochastic data, a kriging model with homogeneous or heterogeneous noise can be used. The response surface will be used instead of the computer code to approximate the real value. A validation phase of the fitted model is then required. Firstly, the fitted model must be close to the training data set. One must quantify the learning error and also check the validity of the fitted model. For example, in a regression context, a normality test can be performed on the residuals. Secondly, this fitted model is the key point of the study as it will be used as a surrogate for the simulator code. It is then important to test the predictive performance of the model at points which do not belong to the training set.

In this article, we present a part of a numerical toolbox that provides a complete range of functions for each step of a case study. All methods used through the consortium applications are implemented using the R language (see R Core Team 2015) and integrated as R packages. The DICE toolbox consists of the following five packages:

- **DiceDesign** for numerical design of experiments (Franco, Dupuy, and Roustant 2014).
- **DiceEval** for evaluation and comparison of metamodels (Dupuy and Helbert 2014).
- **DiceKriging** for the estimation of a response surface via Gaussian processes (Roustant, Ginsbourger, and Deville 2015).
- **DiceOptim** for global optimization (Ginsbourger and Roustant 2015).
- **DiceView** for graphical methods for computer experiments design and models (Richet, Deville, and Chevalier 2013).

Each package addresses a specific task where classical and new methods are implemented. **DiceDesign** is dedicated to the construction of space filling designs and the computation of discrepancy and distance criteria to study a distribution of points. **DiceEval** provides tools for the evaluation and the comparison of classical metamodels. A validation procedure (containing numerical criteria, graphical plots and cross-validation methods) is also provided to validate a fitted model. **DiceKriging** implements a large panel of kriging models. The component **DiceOptim** performs different versions of the efficient global optimization algorithm proposed by Jones, Schonlau, and Welch (1998) to optimize the simulator using kriging. **DiceView** has been implemented to visualize a fitted kriging model for a better understanding of its behavior. Therefore, this package allows 2D/3D sections of kriging models obtained by **DiceKriging** to be viewed.

This article only deals with the first two packages, while a full description of **DiceKriging** and **DiceOptim** can be found in Roustant, Ginsbourger, and Deville (2012). This paper is organized as follows: Section 2 focuses on the **DiceDesign** package. First, the architecture of the package is described in detail. The generated space filling designs and the associated quality criteria are introduced and illustrated in dimension 2. Section 3 is dedicated to the **DiceEval** package. Metamodel quality criteria are defined and the functionalities of this package are illustrated on a test example in dimension 2. In Section 4, the package routines are illustrated on a five-dimensional industrial case provided by the consortium. A conclusion is given in Section 5.

2. DiceDesign package

In the computer experiments context, the number of simulations is often constrained by the high cost of one simulation. The exploratory phase consists of defining the position of the simulated points. In the absence of prior information about the output variable(s), it seems important to spread the points evenly throughout the experimental domain. Where deterministic simulators are concerned, replications are useless (in the sense that running the code many times with the same input values provides the same value for the output). Moreover, it is possible that the response depends only on some input factors or some combination of the inputs, hence alignments should be avoided.

The **DiceDesign** package provides routines to create some specific space filling designs (SFD) and to test their intrinsic quality. Table 1 gives the list of the available functions. As can be seen in Table 1, **DiceDesign** contains three groups of R routines: numerical criteria to assess the quality of the designs, a graphical tool and routines to generate designs.

Since version 1.2, the package has been supplemented with the routines described in Table 2. The new functions introduced in version 1.2 concern latin hypercube samples (LHS). Two algorithms to optimize LHS are proposed: the first one is based on a stochastic algorithm (ESE) and the second method uses simulated annealing (SA). These two methods are applied to the maximin criterion and to the discrepancy criterion. More details can be found in Damblin, Couplet, and Iooss (2013). Another space filling design generated with the WSP (Wooton, Sergent, Phan-Tan-Luu) algorithm is available (Santiago, Claeys-Bruno, and Sergent 2012).

Name	Category	Description
`coverage`	Criterion	Distance measure
`meshRatio`	Criterion	Distance measure
`mindist`	Criterion	Distance measure
`discrepancyCriteria`	Criterion	Discrepancy criteria
`rss2D/rss3D`	Statistical tool	Test and visualization of uniformity properties
`runif.faure`	Design	Low discrepancy sequence
`dmaxDesign`	Design	SFD based on a covariance matrix
`straussDesign`	Design	SFD based on the Strauss process

Table 1: Functions of the **DiceDesign** package.

Name	Category	Description
`mstCriteria`	Criterion	Mean and standard deviation of the MST
`discrepESE_LHS`	Design	Low discrepancy LHS (genetic algorithm)
`discrepSA_LHS`	Design	Low discrepancy LHS (simulated annealing algorithm)
`maximinESE_LHS`	Design	Maximin LHS (genetic algorithm)
`maximinSA_LHS`	Design	Maximin LHS (simulated annealing algorithm)
`wspDesign`	Design	Space filling design based on WSP algorithm

Table 2: Functions available in the **DiceDesign** package since version 1.2.

Lastly, Franco, Vasseur, Corre, and Sergent (2009) proposes classifying space filling designs using the mean and the standard deviation of the minimal spanning tree (MST). This criterion has also been implemented.

In this article, we focus on the functions of Table 1. We first present the quality criteria and the graphical tool. Then, the principles of the space filling designs implemented in the package are detailed and a two-dimensional case study will be used to compare them.

2.1. Quality criteria for computer experiment designs

During the exploratory phase, it seems important to ensure that the experimental domain is adequately covered by a design with a small number of points. The number of inputs being often large, a simple representation does not provide enough information to quantify the distribution of points in the whole domain. Therefore, practitioners use some criteria to study the distance between points or to evaluate to what extent the distribution is close to a uniform one. A part of **DiceDesign** is dedicated to the implementation of numerical criteria. Let $X = \{x^1, \ldots, x^n\}$ be a design of n experiments in $[0,1]^d$ where d represents the number of input parameters[1]. Among intrinsic criteria that can be used to characterize the distribution of the points throughout the experimental region, two groups can be identified: criteria directly computed using the distance between pairs of points, and discrepancy measures that quantify how a given distribution of points deviates from a perfectly uniform one. For a discussion of several criteria introduced below, one can see Pronzato and Müller (2012).

Distance criteria

Denote by γ_i the minimum distance between the point x^i and the other points of the design.

The `coverage` criterion which measures whether a design is close to a regular mesh is defined by:

$$\texttt{coverage} = \frac{1}{\bar{\gamma}} \left(\frac{1}{n} \sum_{i=1}^{n} (\gamma_i - \bar{\gamma})^2 \right)^{1/2},$$

where $\bar{\gamma}$ is the mean of the γ_i's. For a regular mesh, `coverage` $= 0$. Thus, a small value of the `coverage` measure means that the design is close to a regular grid.

[1] Most criteria are computed for a design in the unit cube $[0,1]^d$. If this condition is not fulfilled, the design is automatically rescaled.

	coverage	meshRatio	mindist
Random design	0.508	9.759	0.030
Grid 2^5	0	1	0.25

Table 3: Comparison of the distance criteria for a random design and a 5^2 factorial design.

The `meshRatio` is the ratio between the largest minimum distance and the smallest minimum distance:

$$\texttt{meshRatio} = \frac{\max_{i=1,\dots,n} \gamma_i}{\min_{i=1,\dots,n} \gamma_i}.$$

For a regular mesh, `meshRatio = 1`. Hence, it would be preferable to have a small value of `meshRatio`.

The last distance measure implemented in this package is called `mindist`. It returns the smallest distance between two points. The `mindist` criterion is defined by Chen, Tsui, Barton, and Allen (2003):

$$\texttt{mindist} = \min_{i=1,\dots,n} \gamma_i. \tag{1}$$

A small value of `mindist` means that there is a pair of points which are close, whereas a large `mindist` means that the points are well spread throughout the experimental domain. The maximization of the `mindist` criterion is called *maximin* criterion (see Johnson, Moore, and Ylvisaker 1990). This criterion is commonly used to optimize latin hypercube designs to ensure better space filling properties.

The following code illustrates the computation of the aforementioned criteria for a two dimensional random design (with twenty points):

```
R> n <- 20
R> dimension <- 2
R> X_random <- matrix(runif(n * dimension), ncol = dimension, nrow = n)
R> x <- seq(0, 1, length = 5)
R> grid <- expand.grid(x, x)
R> coverage(X_random)
R> meshRatio(X_random)
R> mindist(X_random)
```

In Table 3, a grid with five levels per dimension is taken as a reference although such a design displays alignments. The small value of the `mindist` criterion for the random design means that some points of the design are close.

Discrepancy criteria

Another measure of uniformity is the discrepancy. The L^p discrepancy compares a distribution of points to the uniform distribution. The discrepancy of a design is low if the number of points of the design falling into an arbitrary set is almost proportional to the measure of this set.

Different L^2 discrepancies are available in **DiceDesign**. For example, if we denote by $Vol(J)$ the volume of a subset $J \subset [0,1]^d$ and $A(X, J)$ the number of points of X that fall into J,

`type`	Specification
`L2`	Star L^2-discrepancy
`C2`	Centered L^2-discrepancy
`M2`	Modified L^2-discrepancy
`S2`	Symmetric L^2-discrepancy
`W2`	Wrap-around L^2-discrepancy
`all`	All types of discrepancies mentioned above (default)

Table 4: Possible specifications of argument `type` for `discrepancyCriteria`.

the L^2-star discrepancy is:

$$DL2^{\star}(X) = \left[\int_{[0,1]^{2d}} \left(\frac{A(X, J_{a,b})}{n} - Vol(J_{a,b})\right)^2 \mathrm{d}a\ \mathrm{d}b\right]^{1/2}$$

where $a = (a_1, \ldots, a_d)^\top$, $b = (b_1, \ldots, b_d)^\top$ and $J_{a,b} = \prod_{i=1}^d [a_i, b_i)$. The other L^2-discrepancies are defined according to the same principle with a different form from the subset J.

Among all the possibilities, `discrepancyCriteria` implements only the L^2 discrepancies because they can be expressed analytically even for high dimensions. This function takes two arguments: the first (`design`) is a matrix or a data frame corresponding to the design of experiments; the second (`type`) is a single value or a vector which describes the types of discrepancies to compute (see Table 4).

```
R> discrepancyCriteria(X_random, type = c("L2", "M2"))

$DisL2
[1] 0.0224633

$DisM2
[1]  0.09611067
```

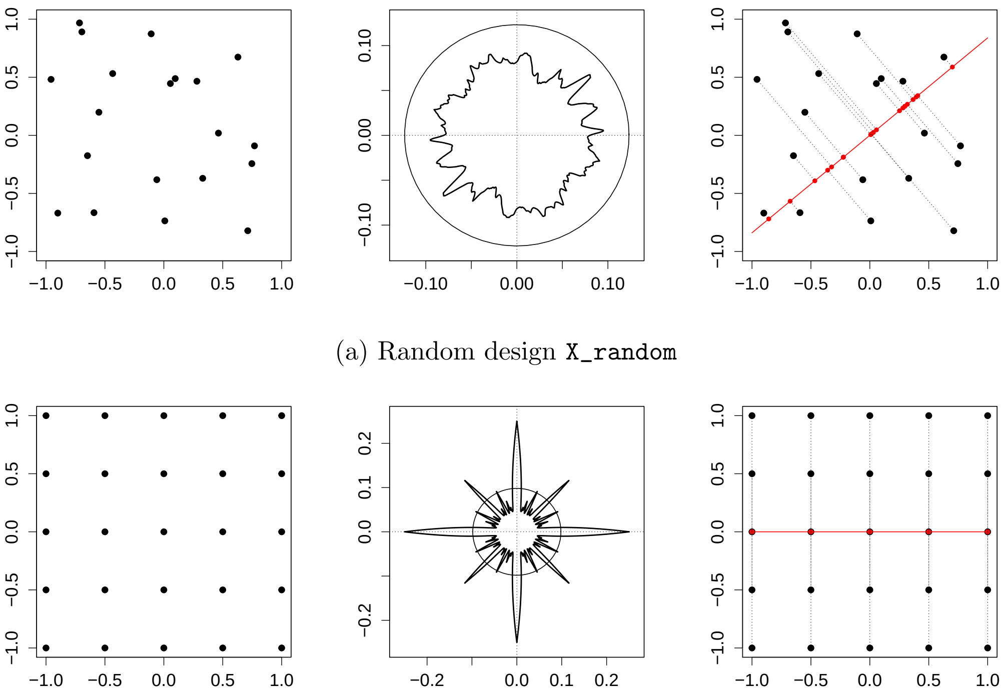
Note that in **DiceDesign**, centered L^2-discrepancy is computed using the analytical expression provided by Hickernell (1998). The interested reader can refer to Pleming and Manteufel (2005) for more details about the wrap-around discrepancy.

Radial scanning statistic

The function `rss2(3)d` refers to a statistical tool which tests the uniformity of the design.

For a two-dimensional design, the radial scanning statistic (RSS) angularly scans the domain. In each direction, it compares the distribution of projected points to their theoretical distribution under the assumption that all design points are drawn from a uniform distribution. For a d-dimensional design, `rss2d` detects the pair of dimensions that corresponds to the worst case (detection of alignments and clustering) and represents the projection on this two-dimensional factorial subspace.

`rss3d` is similar but for a three-dimensional subspace. For a d-dimensional design, all triplets of dimensions are tried to detect the worst defect according to the specified goodness-of-fit



(a) Random design `X_random`

(b) 5^2 factorial design

Figure 1: Graphical output of `rss2d` for a random design with 20 points in dimension 2 (top) and for a grid with five points per dimension (bottom). For the grid, the alignments are detected and the hypothesis of uniformity is rejected.

statistic. The output `worst.case` contains the triplet of dimensions that gives the worst value of the statistics and `worst.dir` corresponds to the direction that gives the worst value of the statistic in the three-dimensional subspace defined by the `worst.case` triplet.

These functions offer a viewing tool similar to a radar screen which corresponds to the statistic value in each direction. The uniformity radar can be used to detect the defects of a design before using it, especially when only a few variables are really significant.

The call of `rss2d` on the random design `X_random` introduced above is done as follows:

```
R> rss <- rss2d(design = X_random, lower = rep(0, dimension),
+    upper = rep(1, dimension))

2D Radial Scanning Statistic (RSS) with GREENWOOD statistic
Discretization step (in degree) : 0.5

Maximum of RS statistic values (global statistic) per pair of dimensions
(1,2) 0.1049559
```

The numerical output of the `rss2d` routine gives the value of the statistic in the worst direction. Here, for the `X_random` design, the output of `rss2d` is equal to `0.1049559`. This value has to be compared to the threshold value at 5% obtained under the uniform assumption (`rss$gof.test.stat = 0.12325`). The observed value being less than the reference, the design is not rejected, hence is considered as good.

This analysis can also be directly performed using the graphical output of the function. The three graphics at the top of Figure 1 show the results of `rss2d` obtained on the random design `X_random` whereas the results at the bottom concern a grid. The points of the design are plotted on the left, the radial statistic on the middle and the projected points on the right. For the random design, the observed statistic does not cross the threshold in any direction. The projected points in the worst direction can still be considered as sampled from a uniform distribution whereas for the grid, uniformity is definitely rejected. The threshold is crossed several times. For example, in the horizontal direction all 20 projected points collapse into 5 points.

More details about the radial scanning statistic can be found in Roustant, Franco, Carraro, and Jourdan (2010).

2.2. Space filling designs

In computer experiments, designs are commonly based on latin hypercubes (McKay, Beckman, and Conover 1979; Stein 1987). Another possibility is the use of low discrepancy sequences (e.g., Halton 1960; Hammersley 1960; Sobol' 1967; Faure 1982; Niederreiter 1987). As latin hypercube designs and low discrepancy sequences have already been implemented in R (see, for example, the packages **lhs**, Carnell 2012, for latin hypercube and **randomtoolbox**, Chalabi, Dutang, Savicky, and Wuertz 2014, or **fOptions**, Wuertz 2013, for low discrepancy sequences except Faure's sequence), we focus here on other space filling designs: maximum entropy designs (Shewry and Wynn 1987; Johnson *et al.* 1990) and Strauss designs (Franco, Bay, Dupuy, and Corre 2008) which can be directly generated from the **DiceDesign** package.

In this section the designs will be compared on the basis of the value of the `mindist` criterion defined by (1). This choice is justified by the fact that this criterion is relatively easy to interpret, is appropriate to the space filling context and is commonly used to optimize latin hypercube designs.

Dmax designs

The designs generated by the `dmaxDesign` routine are obtained by maximizing the determinant of the correlation matrix using a Federov-Mitchel exchange algorithm. The principle is to assume that a spatial Gaussian process will be observed at the points of the design. The spatial correlation between observations is defined by a variogram (Cressie 1993). For example, if we consider the *spherical* variogram γ defined by:

$$\gamma(h) = 1.5\frac{h}{\texttt{range}} - 0.5\left(\frac{h}{\texttt{range}}\right)^3 \quad \text{for } h \leq \texttt{range},$$

where h is the distance between two points and the `range` parameter exactly corresponds to the distance beyond which there is no longer any correlation between observations.

The spatial correlation matrix between coordinates, $C = (\rho)_{ij}$, is then defined by:

$$\rho_{ij} = \begin{cases} 1 - \gamma\left(h_{ij}\right) & \text{if } h_{ij} \leq \texttt{range}, \\ 0 & \text{if } h_{ij} > \texttt{range}, \end{cases}$$

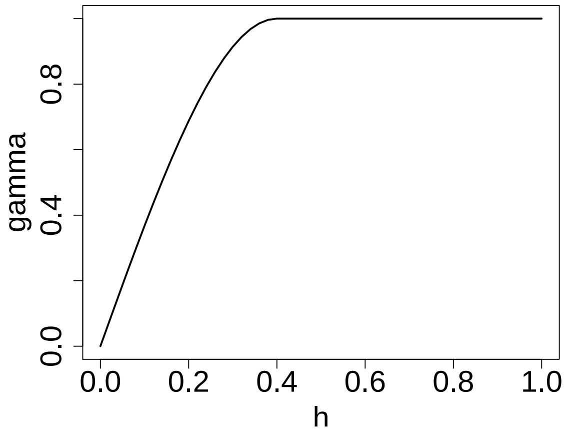


Figure 2: Spherical variogram γ for `range` = 0.4.

where h_{ij} is the distance between x^i and x^j and `range` is the range of the variogram.

The objective of the construction is to select the best set of points sufficiently distant from each other so that the observations at these points are not correlated. Thus, if the points are separated by more than the `range`, the determinant will be its maximum value.

Note that these designs are also called *maximum entropy designs* because maximizing the determinant of the correlation matrix is equivalent to maximizing the entropy of the distribution of points.

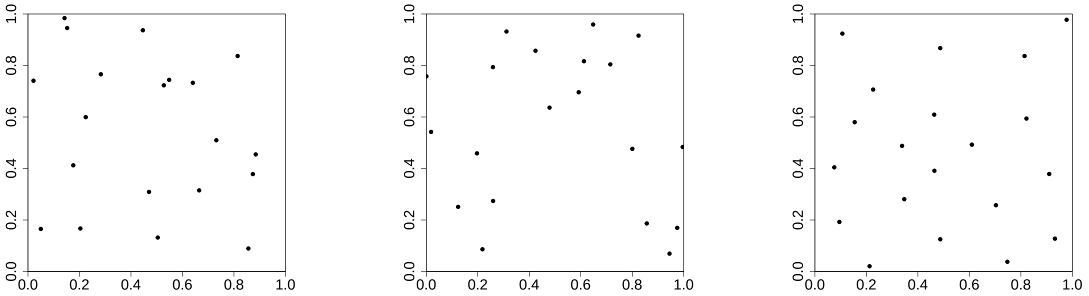
In `dmaxDesign`, the *spherical* variogram has been chosen. Such a choice can be justified by the fact that the impact of the `range` parameter is more important than the behavior at the origin (and hence, the choice of the covariance structure). Therefore `Dmax` designs depend only upon the `range` parameter.

```
R> X <- dmaxDesign(n = 20, dimension = 2, range = 0.2, niter_max = 1000,
+    seed = 1)
R> names(X)
```

```
[1] "n"           "dimension"   "range"       "niter_max"   "design_init"
[6] "design"      "det_init"    "det_end"     "seed"
```

Figure 3 presents Dmax designs obtained for two different values of the `range` parameter. For the first case, the `range` is small (0.1) so that the determinant becomes zero as soon as the points are separated by a distance greater than the `range`. The resulting design contains some empty areas and some points too close to each other. This problem is reduced with a larger `range` (0.2). In practice, it is therefore preferable to choose a large value for the `range` parameter.

The evolution of the `mindist` criterion as a function of the range parameter is given in Figure 4. It can be seen that the `mindist` criterion increases with the `range`. Note that the value of the determinant can be used to compare designs only if the `range` parameters that are used to build the designs are the same. Indeed, attention must be paid to the fact that whatever the quality of the design, the determinant is large when the `range` is small and small when the `range` is high.



(a) Initial random design (b) Dmax design – `range` = 0.1 (c) Dmax design – `range` = 0.2

Figure 3: Initial distribution and maximum entropy designs for different values of the `range` parameter (20 points in dimension 2 after 1000 iterations).

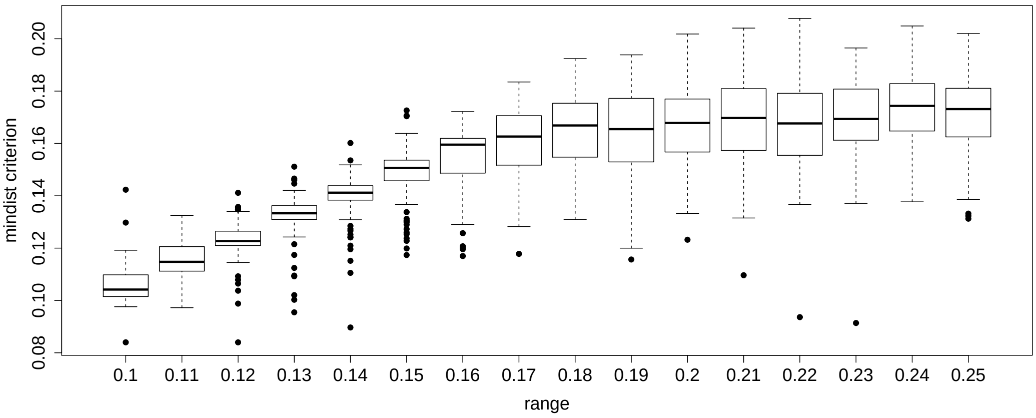


Figure 4: Evolution of the `mindist` criterion as a function of the `range` parameter (100 calls to `dmaxDesign` with `niter_max` = 2000).

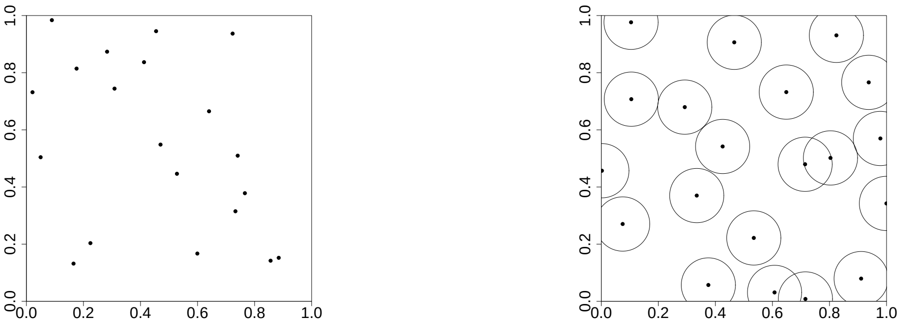
Strauss designs

Strauss designs (`straussDesign`) have been developed by the DICE consortium and are based on the Strauss-Gibbs process. This process has been historically introduced to represent repulsion between charged particles. A stochastic simulation is used to construct a Markov chain which converges to a spatial density of points described by the Strauss-Gibbs potential. In practice, the Metropolis-Hastings algorithm is implemented. The result of a simulation is represented in Figure 5. A sphere is drawn around each point of the design. It represents the area of influence of a point on the other points of the design. A point has an influence on another point if the two spheres intersect. Conversely, we say that there is an interaction between two points if their spheres intersect.

Note that the `C` random number generator (used by `straussDesign`) is installation dependent[2]. Hence, Strauss designs represented in the sequel will be difficult to reproduce.

Recall that we denote by $X = \left\{x^1, \ldots, x^n\right\}$ the design of n experiments in $[0,1]^d$. The

[2] Actually, the `C` random generator depends upon the version of the `libc` library.



(a) Initial random design

(b) Strauss design

Figure 5: Initial distribution and the resulting Strauss design (20 points in dimension 2). The design on the right is obtained after 1000 Markov chain Monte Carlo iterations with an interaction radius `RND` $= 0.19$, a potential $\alpha = 0.5$ and a repulsion parameter $\gamma = 0.001$. The spheres of radius `RND`$/2$ represented in Figure (b) quantify the area of influence of each point of the pattern.

probability density function π of the family of Gibbs point processes is:

$$\pi(x) \propto \exp(-U(x)) \tag{2}$$

where the potential U is represented by the function $U(x) = \beta \sum_{1\leq i<j\leq n} \varphi\left(\left\|x^i - x^j\right\|\right)$. In this formulation, $\beta = -\ln\gamma$ with $\gamma \in]0,1]$ a repulsion parameter and $\varphi : [0,+\infty[\longrightarrow \mathbb{R}$ is a decreasing continuous function such that $\varphi(0) = 1$ and $\lim_{x\to\infty} \varphi(x) = 0$. Note that as the local potential U takes into account only the interactions between pairs of points, the density π corresponds to a so-called pairwise interaction process.

`straussDesign` implements the family of functions $\varphi_{\alpha,\texttt{RND}}$ represented in Figure 6, defined by:

$$\varphi_{\alpha,\texttt{RND}}(h) = \begin{cases} \left(1 - \frac{h}{\texttt{RND}}\right)^{\alpha} & \text{for } 0 \leq h \leq \texttt{RND}, \\ 0 & \text{otherwise ,} \end{cases} \tag{3}$$

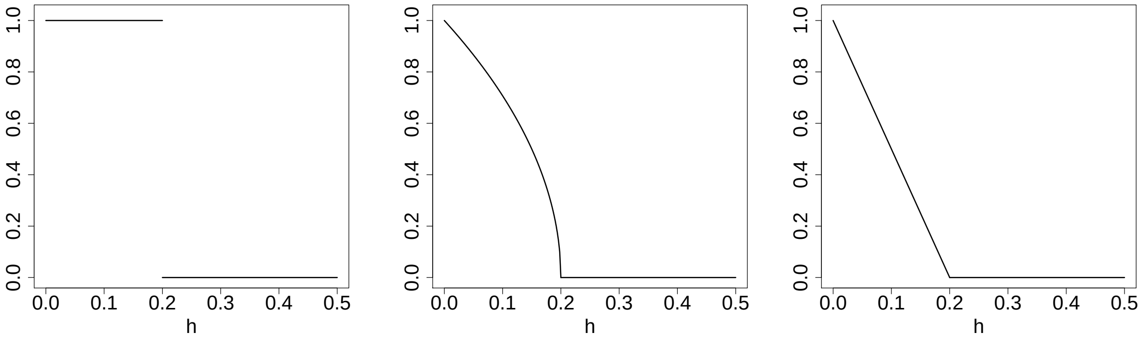
and depending on a potential parameter α and on the interaction radius `RND`.

Strauss processes correspond to the particular value $\alpha = 0$. In this case, the distribution π can easily be written as:

$$\pi(X) = k\gamma^{s(X)},$$

where k is the normalizing constant, $\gamma \in]0,1]$ is the repulsion coefficient (with β in the previous formulation equal to $-\ln\gamma$) and $s(X)$ is the number of distinct pairs of points $\left\{x^i, x^j\right\}$ of the design X that are separated by a distance no greater than the interaction radius `RND`.

Particular attention must be paid when setting the parameters. Each parameter (radius, potential and repulsion) has an impact on the final distribution. For the sake of readability, a brief study in dimension 2 will be conducted in order to illustrate the impact of each parameter on the distribution of points. Although the two-dimensional case is a little basic, it allows the results to be visualized. It is worth noting that the remarks on the influence of the parameters on the final distribution of points remain true in higher dimensions. Moreover, an advantage



(a) $\alpha = 0$ (b) $\alpha = 0.5$ (c) $\alpha = 1$

Figure 6: Potential power $\varphi_{\alpha,0.2}$ as a function of the distance between points h. The trivial case corresponds to the potential 0-1 ($\alpha = 0$) and intermediate case to the power $\alpha = 0.5$.

of these designs is their flexibility which allows the generation of a design coherent with the specific nature of the data and the objective of the study. For example, the fitting kriging model (Santner, Williams, and Notz 2003) requires the estimation of the variogram parameters (shape, nugget, sill and range); it is thus important to have information for different distances between points, even for small distances. When setting up a Strauss design, it is thus possible to adjust its parameters to cope with such configurations. This is often more complicated to achieve with other types of designs (e.g., minimax latin hypercube).

In dimension 2, one can easily represent the design and we will take advantage of this feature to illustrate the impact of the parameters on the quality of the point distribution, which will be evaluated by the `mindist` criterion. Note that the L^2-discrepancy criterion has also been computed for all the tested designs and its value is between 0.015 and 0.040; as a point of comparison, the value of Faure's low discrepancy sequence is 0.020.

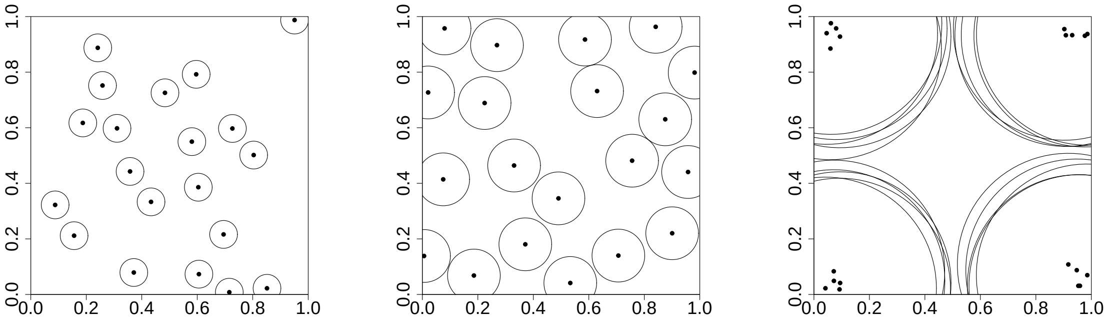
As shown previously, basic Strauss designs depend mainly on four parameters. The number of Monte Carlo iterations is fixed at its default value, namely `NMC` $= 1000$. The other parameters are the interaction radius `RND`, the repulsion parameter γ and the potential power α. We now study each of these parameters.

Interaction radius. The dependence on the radius parameter turns out to be important (see Figure 7). Excessively small radius values (Figure 7a) result in a distribution without interaction but with many gaps, while excessively large values lead to distributions with clusters (Figure 7c).

To complete this study, Figure 8 represents the change in the `mindist` criterion with respect to the value of the radius `RND`. Given the random nature of Strauss designs, 500 designs have been constructed for each value of `RND`. The optimal value of `RND` for the criterion `mindist` is slightly higher than 0.2 but it also corresponds to the area where the variability of this criterion is high. Then it may be wise to choose a radius slightly less than the optimal value.

Potential power α. The potential power α (see Figure 6) represents the strength of the interactions between points at a distance less than or equal to `RND`.

An interaction exists as soon as the spheres of radius `RND`/2 intersect. For the special case



(a) `RND` $= 0.1$ (b) `RND` $= 0.19$ (c) `RND` $= 0.8$

Figure 7: Impact of the `RND` parameter (20 points in dimension 2; $\alpha = 0$; $\gamma = 0.001$).

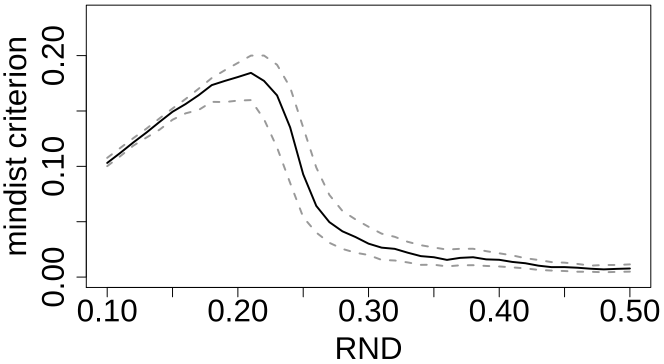


Figure 8: Impact of the interaction radius `RND` on the `mindist` criterion. The continuous line represents the median of the `mindist` criterion and the dashed lines the empirical quartiles (Q1 and Q3) for 500 designs. The designs were generated with $\alpha = 0.5$ and $\gamma = 10^{-4}$.

$\alpha = 0$, there is no difference between points that are close and those at a distance close to `RND`. The advantage of considering a potential different from zero is that the influence of a point on its neighborhood (closer than `RND`) actually depends on the distance between the points. Potential power can be used to give more importance to points that are close.

As shown previously, the choice of the interaction radius is not obvious. The flexibility offered by the potential parameter could be an alternative to correct an unfortunate choice of radius.

Repulsion parameter γ. Another comment on this kind of design concerns the `repulsion` parameter γ which is explicitly linked with the acceptance ratio. Indeed, the construction consists of choosing a point x^i of the current design and to propose a new point y_i. In the case of the Strauss process represented in Figure 10, the acceptance probability of the new point is $\min\left(1, \gamma^{s(y_i)-s\left(x^i\right)}\right)$ where $s\left(x^i\right)$ is the number of points of the design X in interaction with the point x^i.

It is important to notice that a small value will necessarily lead to a large value for the `mindist` criterion (which corresponds to the best way to fill a square space with balls). Depending on

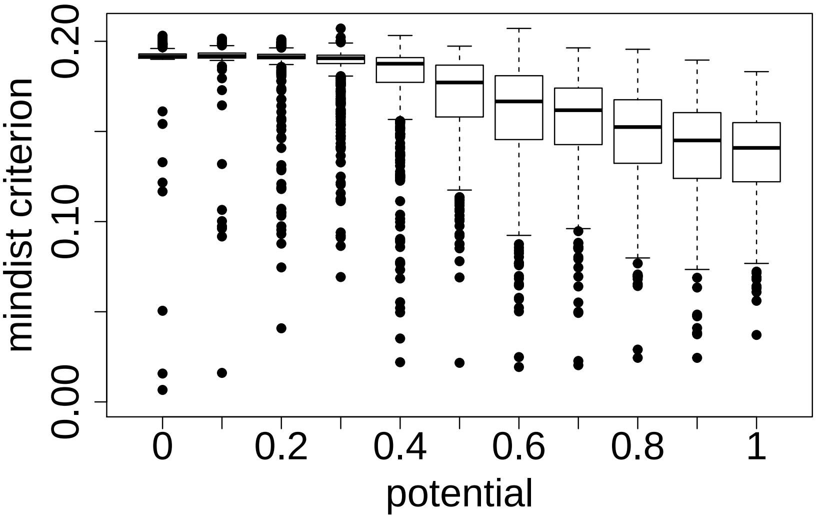
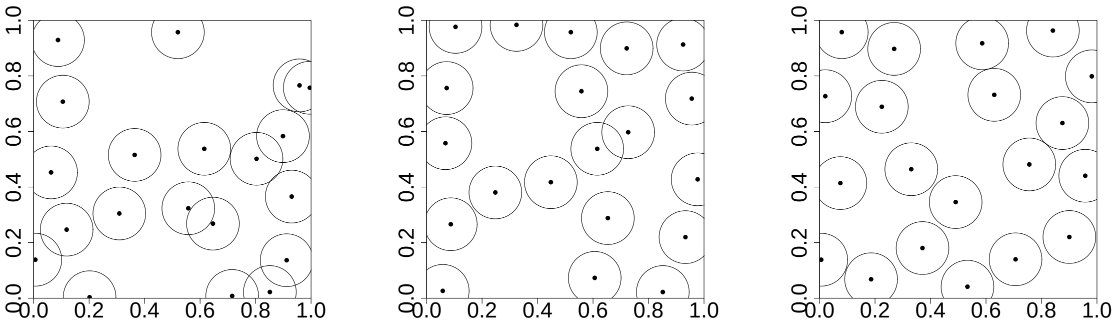


Figure 9: Impact of the α parameter on the `mindist` criterion (20 points in dimension 2). The `repulsion` parameter corresponds to $\gamma = 10^{-4}$ and the interaction radius `RND` $= 0.19$.



(a) $\gamma = 0.1$
`mindist` $= 0.055$

(b) $\gamma = 0.01$
`mindist` $= 0.150$

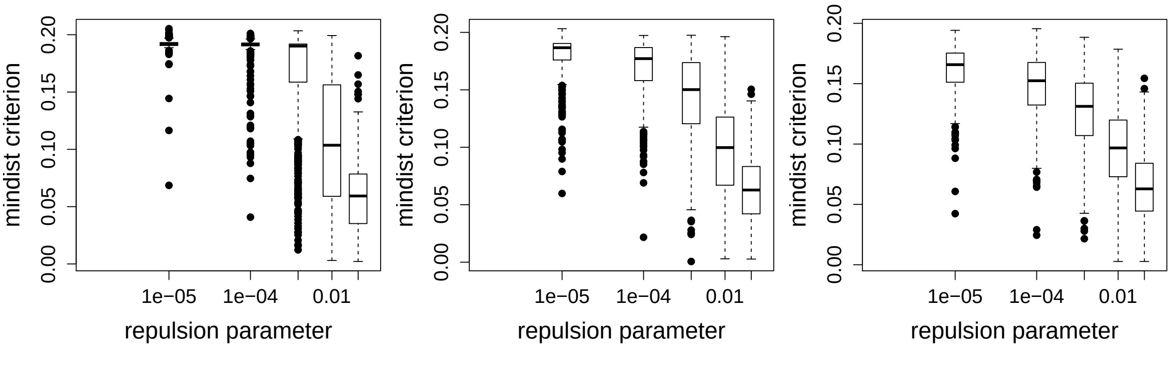
(c) $\gamma = 0.001$
`mindist` $= 0.191$

Figure 10: Impact of the `repulsion` parameter on the design (20 points in dimension 2). The interaction radius `RND` is fixed at 0.19 and the power α is fixed at 0.

the objective of the study, it might be important to generate points which are either closer or further apart. Hence, the value of the `repulsion` parameter should not be too small because otherwise points at a distance less than `RND` will almost never be accepted.

In Figure 11, the influence of the repulsion parameter on the `mindist` criterion is shown for three values of the power α. This representation confirms that a small value for the repulsion γ leads generally to a large value for the `mindist` criterion. The variability of the `mindist` is greater with $\alpha = 0.8$ than with lower values.

Constraints on the margins. The simulated law π can be adapted to some specific properties. Let us suppose that we wish to simulate a space filling design in dimension d and also in dimension 1 as latin hypercubes do. It is then possible to adapt the law π by taking into



(a) $\alpha = 0.2$ (b) $\alpha = 0.5$ (c) $\alpha = 0.8$

Figure 11: Impact of the `repulsion` parameter on the `mindist` criterion. The boxplots are drawn with the results obtained for 500 designs with 20 points in dimension 2, the value of the interaction radius is fixed at `0.19`.

account local potentials in the definition of the potential U:

$$U(x) = \beta \sum_{1 \le i < j \le n} \varphi\left(\left\|x^i - x^j\right\|\right) + \sum_{k=1}^{d} \beta_k \sum_{1 \le i < j \le n} \varphi_k\left(\left|x_k^i - x_k^j\right|\right)$$

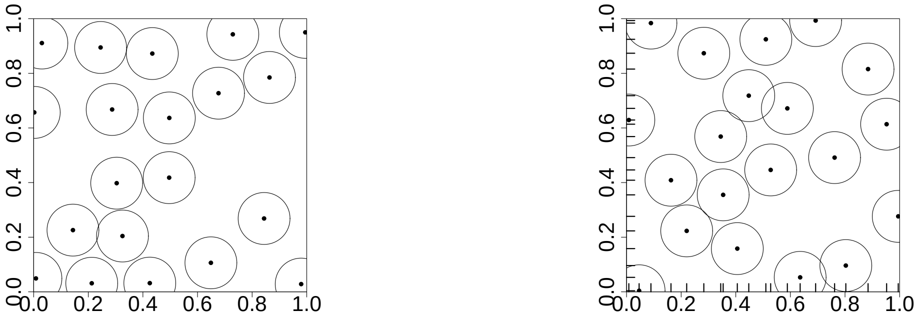
where φ_k are functions defined by (3) and $\beta_k = -\ln \gamma_k$ where γ_k are the repulsion parameters for each one-dimensional axis.

This variant is already implemented in the `straussDesign` function: the value of the argument `constraints1D` indicates whether constraints on the projections must be taken into account (`constraints1D` = 1) or not (`constraints1D` = 0 by default). All the functions φ_k ($1 \le k \le d$) correspond to $\alpha = 0$ and the interaction radius `R1D` (that corresponds to the interval of influence on the margins) is set to $0.75/n$ where n is the number of points of the design. The repulsion parameter γ_k that controls the repulsion between the projected points can be chosen by the user (argument `gamma1D`) but it will be the same in all directions.

Figure 12 gives the comparison of designs with and without constraints on the margins. The left plot is generated without constraints; it can be seen that the projections on the axis exhibit clusters. On the right side, constraints on the margins have been imposed and the projections are more uniform.

Due to the stochastic nature of Strauss designs, we compare (Figure 13) the `mindist` criterion of 500 designs of 20 experiments with or without one-dimensional constraints. The numerical values of the `mindist` in dimension 2 and in dimension 1 cannot be compared because the scale is not the same. Note that in dimension 1, the points of a regular grid are spaced at a distance of $1/(n-1) = 0.05$ for $n = 20$. This value corresponds to the maximum value of the `mindist` criterion in dimension 1.

We can see that the designs for which 1D-constraints are taken into account are leading to a greater value of the `mindist` criterion after projection (horizontal axis of Figure 13). The value of the `mindist` for the two-dimensional distribution of points is not really impacted by the constraints on the margins (vertical axis of Figure 13). However this remark ceases to hold true in higher dimensions since adding one-dimensional constraints reduces the value of the global `mindist` criterion.



(a) Strauss without constraint

(b) Strauss with margin constraints

Figure 12: Comparison of Strauss designs without (left) and with (right) constraints on the margins. The projections of the points on the factorial axis are represented as ticks on the axis. The designs have been generated for 20 points in dimension 2 with $\alpha = 0.5$, `RND` $= 0.19$, $\gamma = 0.001$ and $\gamma_{1D} = 0.0001$.

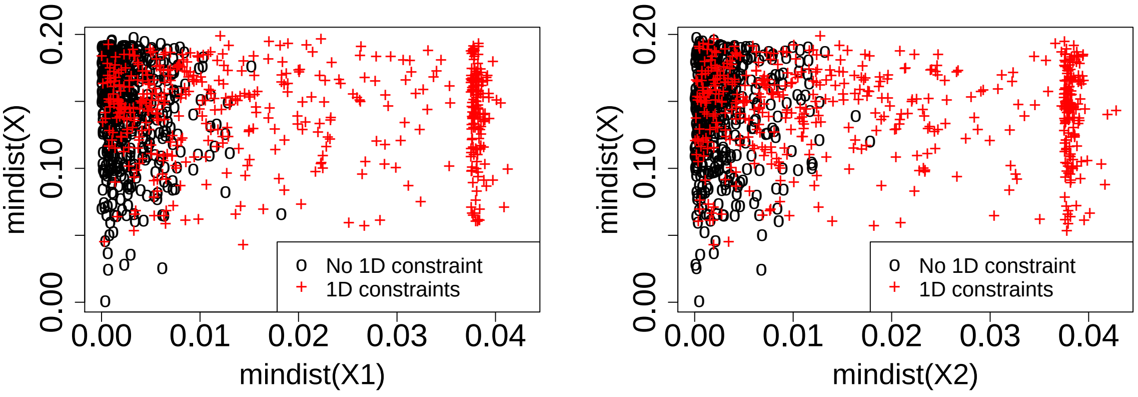


Figure 13: Comparison of the `mindist` criterion using the `constraints1D` option. The vertical axis represents the global `mindist` computed with the two-dimensional design X. The horizontal axis represents the `mindist` criterion computed on the one-dimensional axis: $X1$ on the left and $X2$ on the right. Symbols 'o' correspond to no constraint and symbols '+' to one-dimensional constraints. The results are obtained from 500 designs with 20 points in dimension 2 and for input parameters set at the values $\alpha = 0.5$, `RND`$=0.19$, $\gamma = 0.001$ and $\gamma_{1D} = 0.0001$.

Example of application. The following code generates a Strauss design. On the basis of the previous study, we choose a set of parameters as follows: a radius of 0.21, a potential power of 0.2, a small value for the repulsion parameter and no margin constraints. Better designs can be obtained with larger values of the radius (for example, approaching 0.23), the quality being however more variable (see Figure 8). The output of the function is a list containing the number of experiments (`n`), the dimension (`dimension`), the initial design (`design_init`), the value of the parameters of the Strauss (the interaction radius `RND` (`radius`), the potential power (`alpha`), the repulsion parameter (`gamma`), the number of Monte Carlo iterations (`NMC`), the value of the seed, the constraints on the margins and the Strauss design (`design`).

```
R> dim <- 2
R> n <- 20
R> X <- straussDesign(n, dim, RND = 0.21, alpha = 0.2, repulsion = 0.0001)
R> mindist(X$design)
```

```
[1] 0.210
```

```
R> X$seed
```

```
[1] 1434397079
```

The user can provide a value for the input parameter `seed`. Otherwise, the value of the seed is generated randomly using `Sys.time()` and the `seed` parameter is updated accordingly.

`straussDesign` implements exactly the simulation of a distribution defined by (2). In the above study (20 points in dimension 2), we find the values of the parameters `RND`, α and γ which maximize the `mindist` criterion. If either the number of points or the dimension changes, another study must be conducted. The parameter most affected is the interaction radius `RND`. The algorithm could be extended to the generation of heterogeneous designs or adapted to constrained domains but these generalizations are not yet available in this package.

The general algorithm is detailed in Franco *et al.* (2008).

3. DiceEval package

Suppose now that the simulations have been carried out at $x^1, \ldots x^n$. The numerical values of the output are then analyzed to provide a response surface for prediction, uncertainty propagation or global optimization. The **DiceEval** package deals with the construction of different types of response surfaces also called surrogates or metamodels. Several procedures (especially numerical and graphical tools) are provided to validate fitted models. Then, **DiceEval** gathers the most common methods used to model a computer code output. Most of them are already available in R.

The main contributions of this package are the standardization of the calls to fit the different metamodels and the implementation of standard criteria for quantifying the error between the model and the simulated phenomenon. The implementation of graphical tools to analyze and compare metamodels is also available. Moreover, for methods that depend upon a parameter, some routines have been developed to assist the user in choosing the best value for the parameter. The **DiceEval** routines are described in detail in Table 5.

In the sequel, a toy example in dimension 2 will serve as a running example. The main goal of this study is to illustrate to what extent the quality of the predictions is influenced by the choice of the experimental design and the choice of the type of metamodel. Note that the random design is the one generated in Section 2.1 and other designs are constructed according to the remarks in Section 2.2. The conclusion of the two-dimensional example is presented at the end of this section (see Section 3.6).

Name	Description
`dataIRSN5D`	Data set provided by IRSN
`testIRSN5D`	Test data for the `dataIRSN5D` case
`modelFit`	Fitting of metamodels
`modelPredict`	Prediction at new data for a fitted metamodel
`modelComparison`	Comparison of different types of metamodels
`R2`	Multiple R-squared
`MAE`	Mean absolute error
`RMA`	Relative maximal absolute error
`RMSE`	Root mean squared error
`residualsStudy`	Residuals plot
`stepEvolution`	Evolution of the stepwise model
`penaltyPolyMARS`	Choice of the penalty parameter for a PolyMARS model
`crossValidation`	K-folds cross validation
`testCrossValidation`	Test for the cross validation procedure

Table 5: Functions of the **DiceEval** package.

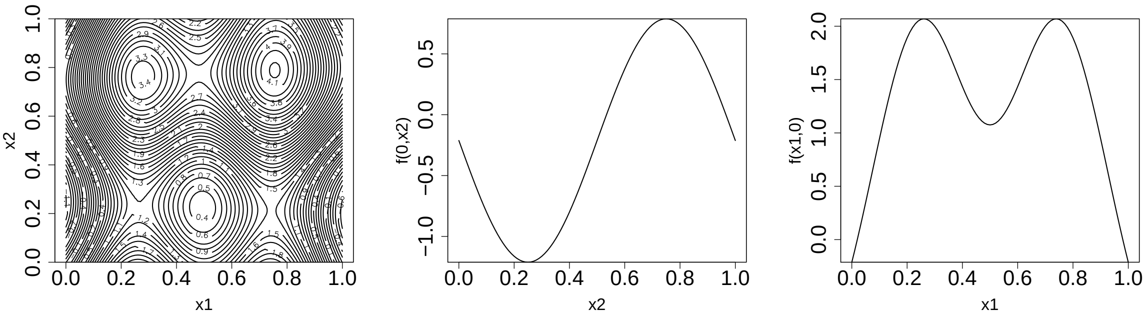


Figure 14: Representation of the function f. The last two figures represent the sections of the function for $x_1 = 0$ and $x_2 = 0$ respectively.

3.1. Illustration on a two-dimensional function

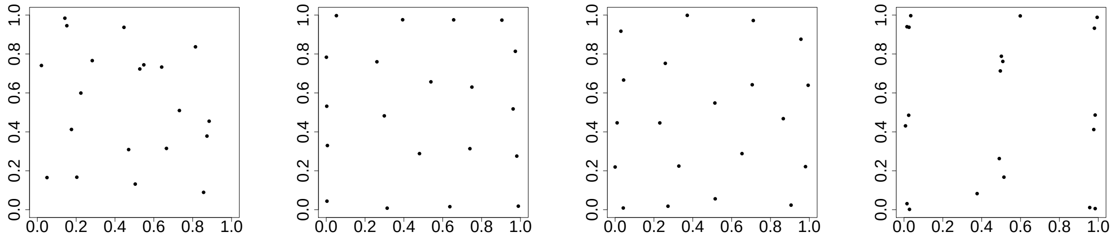
The main functions of the **DiceEval** package have been listed previously. Now, we will present its features in use. The package is illustrated using a 2-dimensional analytic function. Although simplistic, this example describes a study in detail from the beginning to the end. The dimension of the input domain allows a representation of the designs and the construction of the data set is simplified by knowledge of the output. The objective of this study is to highlight the influence of the choice of the design on the quality of the metamodel.

Let us consider the following analytical 2-dimensional test function f:

$$f\left(x_1, x_2\right) = 3.5\left[\exp\left(-\left(4x_1-1\right)^2\right) + \exp\left(-\left(4x_1-3\right)^2\right)\right] - \sin\left(2\pi x_2\right) + 2x_1x_2 - 1.5$$

for $0 \leq x_1 \leq 1$ and $0 \leq x_2 \leq 1$.

The contour plot and two sectional representations of the function f are represented in Figure 14.



(a) Random design (`mindist` = 0.030)

(b) Dmax (`mindist` = 0.175)

(c) Optimized Strauss (`mindist` = 0.210)

(d) Non-opt. Strauss (`mindist` = 0.007)

Figure 15: Designs of experiments in dimension 2.

Choice of a design of experiments in dimension 2

The first step of the study consists in exploring the experimental region $[0,1]^2$. Assume that we can only perform 20 simulations which equates to a design of experiments with ten points per dimension. Given the results of Section 2.2, four different designs represented in Figure 15 have been compared. The first design corresponds to uniformly distributed points. The second design is a Dmax design generated with a `range` parameter equal to 1 and to a large number of iterations (`niter_max` = 10000). Then, two Strauss designs have been generated: the first with the following parameters: `RND` = 0.21, $\alpha = 0.2$ and $\gamma = \exp(-10)$ (it will be called the optimized Strauss design) and the second (called the non-optimized Strauss design) has been generated with an excessively large `RND` that leads to groups of collocalized points. The `mindist` criteria indicated in Figure 15 show large differences between optimized designs and others. The optimized Strauss and Dmax designs have `mindist` criteria larger than 0.21.

3.2. Construction and prediction

The main function `modelFit` is dedicated to the construction of a model from a training data set; `modelPredict` aims at predicting the values of the model at new locations. Five metamodeling techniques are considered, i.e., the linear model and its stepwise version, additive models, MARS, PolyMARS and kriging models.

Note that most of the learning models depend on other packages, namely **gam** (Hastie 2014), **mda** (Hastie 2013), **polspline** (Kooperberg 2013) and **DiceKriging** (Roustant *et al.* 2015). In order to make package maintenance easier, the interface to external packages is restricted to the `modelFit` function. Except for kriging, the different models implemented in the **DiceDesign** package are illustrated in Hastie, Tibshirani, and Friedman (2009).

Additive models

The additive models assume that the output can be decomposed as a sum of one-dimensional functions of each of the inputs. For the inference, the idea is to estimate successively and nonlinearly the functions in each direction thanks to a backfitting procedure. Additive models are a particular case of generalized additive models developed in Hastie and Tibshirani (1991). For example, the function can be searched as a sum of d cubic splines in each direction.

MARS and PolyMARS models

The MARS model (multivariate adaptive regression splines; Friedman 1991) and its extension PolyMARS (polychotomous regression based on MARS; Kooperberg, Bose, and Stone 1997) can be viewed as the generalization of the stepwise linear regression with piecewise linear basis functions. These models do not belong to the family of linear models because each basis function is based on a knot that must also be characterized. These models are very flexible and adaptable. Therefore they are often neither sufficiently robust nor predictive. This is why their construction uses generalized cross validation in order to check for predictivity and penalize the complexity of the model. To go further in reducing the complexity of the inferred model, PolyMARS model is constrained during its construction because in each direction, linear functions must be added before piecewise linear ones. The aim is to provide more robustness, at the expense of flexibility.

Kriging models

The principle of kriging is to assume that the response is the realization of a Gaussian process characterized by a trend and a spatial covariance structure (see Santner *et al.* 2003). The set of parameters to be estimated are the trend coefficients, the ranges of the correlation kernel and the global variance of the process. Unlike previous models, the kriging predictor is interpolating (no prediction noise at the observation points) and the variance of prediction is directly linked to the position between the current point and other points of the design.

Fitting of a model with package ***DiceEval***

Let us denote by `X_unif` the random experimental design and `Y_unif` the corresponding response values. In the following, the first command builds an additive model that fits the data set (`X_unif`, `Y_unif`). The second command retrieves the values obtained on test points contained in the `matrix` (or `data.frame`) `X_test`. Here, the test set is a factorial grid of 10,201 points that corresponds to 101 levels in each direction.

```
R> modAm <- modelFit(X_random, Y_random, type = "Additive",
+    formula = formulaAm(X_random, Y_random))
R> Y_test_Am <- modelPredict(modAm, X_test)
```

Figure 16 gives a 2D plot of the additive response surface and two 1D cut sections with the true and the predictive functions in each direction. The additive model (with cubic spline basis in each direction) gives an approximation of the general behavior of the real function.

3.3. Validation

Numerical criteria are implemented in order to assess the quality of the fitted model. Two cases can be distinguished: the quality of the fitted model on the training set itself and its predictive qualities on a data set different from the training set. First, the training set is the information provided to construct a specified model from these data. Second, the fitted model will be used instead of the simulator; hence, it is important to quantify the generalization quality of the metamodel. In other words, the model must provide predicted values close to the real values even for points which are not in the training set.

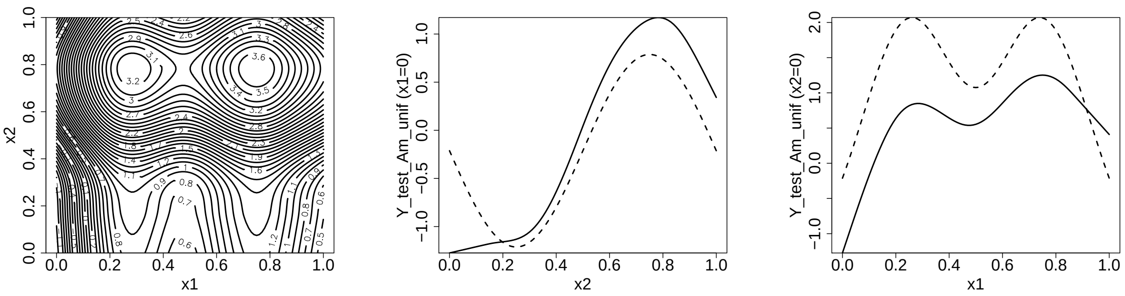


Figure 16: Approximation of the analytical 2-dimensional test function by the additive model `modAm`. Dashed curves represent the true function and the solid curves show the predicted function.

Let us denote by y_i the value of the response variable at the point x^i of the design and $\hat{y}_i$ the predicted value computed from the fitted model. In regression, the most common measure of predictability is the coefficient of determination:

$$\texttt{R2} = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2}, \tag{4}$$

where $\bar{y}$ is the mean of the observed values y_i. This criterion represents that part of the total variance that is explained by the predictors. Note that, as this criterion does not take into account the complexity of the model, it increases with the number of terms contained in the model. To avoid this problem it is recommended to consider the adjusted version of the determination criterion. The adjusted coefficient of determination is mainly defined for linear models. In the **DiceEval** package, criteria that can be evaluated and compared for all the metamodels are implemented. As it appears difficult to evaluate the complexity of MARS and PolyMARS models, the adjusted coefficient of determination is not available.

The root mean squared error (`RMSE`) is a L^2 measure which quantifies the accuracy of a predicted model:

$$\texttt{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2}. \tag{5}$$

The mean absolute error (`MAE`) is defined by

$$\texttt{MAE} = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|. \tag{6}$$

This L^1 criterion is similar to the `RMSE` with the advantage of being more robust because it is less affected by outliers. The last criterion, `RMA` (relative maximum absolute error) is a measure of type L^∞:

$$\texttt{RMA} = \frac{\max_{1 \le i \le n} |y_i - \hat{y}_i|}{\sigma_y} \tag{7}$$

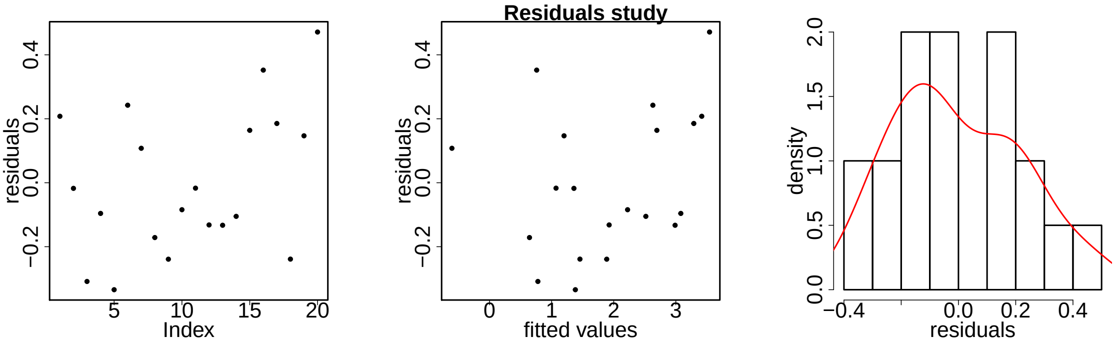


Figure 17: Study of the residuals for an additive model with one-dimensional splines.

with σ_y the standard deviation of $y_1, \ldots, y_n$. In practice, this criterion can be used to point out a region of the experimental domain where the fitted model is not appropriate.

In the **DiceEval** package, an R function is associated with each criterion. All these routines take two vectors as input arguments: the first corresponds to the real values of the response and the second to the predicted values at the same points. Some criteria (namely `R2` and `RMA`) are not symmetrical, hence the user must be careful with the order of the arguments while using these routines.

As can be observed below, the `R2` criterion when used as an evaluation measure of the additive model on the test set is high. This indicates an appropriate model.

```
R> R2(Y_test, Y_test_Am)
```

```
[1] 0.9087903
```

In addition to using the above criteria, the validation of a fitted model often leads to the study of the residuals distribution. The routine `residualsStudy` analyzes the residuals of a model by plotting the residuals (assumption of independence), the residuals against the fitted values (adequacy between model and reality) and the density. The graphical result for the additive model is provided in Figure 17.

Several models in the package depend on various parameters. Among the models, the `stepwise` and `PolyMARS` models depend on a penalty parameter. The stepwise procedure consists of selecting the most influential terms among a set of numerous candidate terms. For example, the forward procedure consists of successively adding the most influential term to the model. This selection is carried out by an automatic minimization that is often parametrized. The package proposes a study of the influence of the penalty parameter using graphical and numerical tools which can be used to choose the most appropriate value.

The criterion to minimize, denoted by C in the following, depends on the `penalty` argument and is defined by:

$$C(\texttt{penalty}) = -n \ln(\hat{\sigma}^2) + M \; \texttt{penalty},$$

where n is the size of the data set, $\hat{\sigma}$ is an estimation of the residual standard deviation and M is the number of parameters in the statistical model. Note that the first term is proportional to the likelihood function of the estimated model.

An illustration of the use of `modelFit` for the metamodels with a `penalty` parameter is detailed in the command lines below.

```
R>  modStep <- modelFit(X_unif, Y_unif, type = "StepLinear", penalty = 2)

Warning message:
In modelFit(X_random, Y_random, type = "StepLinear", penalty = 2) :
  [type=="StepLinear"] argument 'formula' not found, set at 'Y~.'

R> library("polspline")
R> modPolyMARS <- modelFit(X_random, Y_random, type = "PolyMARS", gcv = 2)
```

In the above example a warning is printed because the function is called without the argument `formula`. In such a case, the `formula` argument takes its default value, namely ’`Y ~ .`’.

The `penalty` parameter of the procedure plays an important role since it controls the resulting model complexity. A high `penalty` leads to a model with only a few terms and a possibly high σ^2 whereas a low penalty gives a model with numerous terms and thus low σ^2. In practice, the Akaike information criterion (`penalty` $= 2$) and the Bayesian information criterion (`penalty` $= \ln n$) are frequently used in stepwise procedures.

The routine `stepEvolution` of **DiceEval** studies the impact of the penalty parameter on the final stepwise model. As in the two-dimensional case this routine does not make much sense, it will be illustrated through a five-dimensional one.

The principle used to construct the PolyMARS model is the same as that for the stepwise model. A penalty argument (denoted by `gcv`) is used at the end of the procedure to choose the best model from a sequence of fitted models by minimizing the penalized residual sum of squares (see Kooperberg *et al.* 1997, for the exact definition of the `gcv`). A larger `gcv` value would tend to produce a smaller model. In practice, conducting a preliminary study is recommended to select an optimal value of the `gcv` parameter; hence the `penaltyPolymars` function tests the sensitivity of the quality criteria with respect to the `gcv` parameter. The construction of a model being quite efficient, PolyMARS models are fitted for different values of the `gcv` parameter, criteria that assess the quality of the fitted models (`R2`, `Q2` and if a test set is available `R2` in prediction) are then computed.

```
R> Crit <- penaltyPolyMARS(X_random, Y_random,
+    test = data.frame(X_test, Y_test))
```

A graphical representation of these results is proposed by putting `graphic = TRUE` in the input arguments.

3.4. Cross validation

Cross validation is a resampling method used to take into account the generalization error when no test set is available. This technique can also be used for model selection, to detect overfitting or instability of the model.

The principle is quite simple: the data set is randomly split into K subsets of (approximately) equal size: $A_1, \ldots, A_K$. For each $k = 1, \ldots, K$, a model $\hat{Y}^{-k}$ is fitted from the data of $\bigcup_{j \neq k} A_j$ and this model is validated on the subset A_k.

From an initial model given by `modelFit` and for a specified value of `K`, the `crossValidation` routine computes the predicted values obtained using K-folds cross validation and the `Q2` index defined as:

$$\texttt{Q2} = 1 - \frac{\sum_{i=1}^{n}\left(y_i - \hat{Y}^{k(i)}\left(x^i\right)\right)^2}{\sum_{i=1}^{n}\left(y_i - \bar{y}\right)^2}, \tag{8}$$

with $k(i)$ the number of the fold that contains the data $\left(x^i, y_i\right)$ and $\hat{Y}^{-k(i)}\left(x^i\right)$ the predicted value at the point x^i provided by a model fitted without the data of the set $A_{k(i)}$. We can notice that the `Q2` index has the same expression as `R2`, replacing the usual fitted value with the value obtained by cross validation.

The output of the function `crossValidation` contains moreover two numerical values `RMSE_CV` and `MAE_CV`. Let L be a criterion (here L could be the `RMSE` or the `MAE`). The "evaluation" of the model consists of computing (e.g. Hastie *et al.* 2009, p. 241):

$$L_k = \frac{1}{n/K}\sum_{i\in A_k} L\left(y_i, Y^{(-k)}\left(x_i\right)\right).$$

The cross validation criterion is the mean of the K criteria: $L_{CV} = \frac{1}{K}\sum_{k=1}^{K} L_k$.

Therefore, cross validation is used to estimate the quality of the model in prediction according to some criteria (e.g., `Q2` or `RMSE_CV`, ...). In such a context, the parameters of the initial model are fixed. These parameters are:

- The formula for linear and additive models.
- The formula and the penalty for stepwise models.
- The degree for MARS models.
- The penalty (`gcv`) for PolyMARS models.
- The formula of the trend and the choice of the correlation function for kriging models.

As cross validation allows to estimate the quality of a model with respect to a given criterion, it can be used to optimize the previously fixed parameters of the model. As an example, let us consider a PolyMARS model with `RMSE` as criterion. In such a case, the `gcv` parameter is fixed during the estimation of the `RMSE_CV`. Estimating the `RMSE_CV` criterion for different values of `gcv` allows to optimize the `gcv` value with respect to the `RMSE` criterion.

Leave-one-out (LOO) cross validation consists in using a single observation from the initial data set as the validation data, and the remaining observations as the training set. This is a classical variant of K-folds cross validation with K being equal to the number of observations n. LOO cross validation can be computationally expensive and the most commonly used value is $K = 10$. `testCrossValidation` gives the possibility of testing the robustness of the `Q2` criterion with respect to the number of subsets K.

3.5. Model comparison

The construction of a metamodel is often immediate in terms of computation time. In the context of computer experiments, it is important to choose the "best" model. The routine

	MARS	PolyMARS	Linear	Additive	Kriging
	(`degree` = 2)	(`gcv` = 2)	(quad. trend)		
Random design	0.635	0.714	0.623	0.909	0.973
Dmax	0.653	0.604	0.645	0.933	0.977
Optimized Strauss	0.561	0.654	0.677	0.935	0.988
Non-optimized Strauss	0.261	0.175	0.386	0.549	0.874

Table 6: Comparison of the predictive quality of different designs and different types of metamodels fitted to the two-dimensional toy example. The numerical values correspond to the `R2` criteria computed on the test set.

`ModelsComparison` allows a simultaneous comparison of several models. The output is the set of the quality criteria computed on learning and test sets for all the metamodels. A graphical tool representing these numerical values is provided. It is composed of two columns: the `R2`/`Q2` on the left-hand side and the `RMSE` on the right-hand side. When possible, each criterion is evaluated on the learning set, by cross validation and on the test set. Visually, the best model has the highest values of `R2`/`Q2` and the lowest of `RMSE`.

3.6. Conclusions of the study of the 2-dimensional toy function

Several metamodels have been estimated on the four designs represented in Figure 15: a linear model with a quadratic trend (i.e., the two predictor variables x_1 and x_2, the interaction $x_1 : x_2$ and the squared order terms x_1^2 and x_2^2), an additive model, a MARS model of degree two, a PolyMARS model with `gcv` equal to 2 and a kriging model. The kriging models presented in what follows have been obtained using the **DiceKriging** package (see Roustant *et al.* 2012). The prediction quality of the fitted models is tested on the same 101×101 grid as previously used. Table 6 contains the values of the `R2` criteria.

Table 6 shows that the quality of the fitted models differs widely from one to the other:

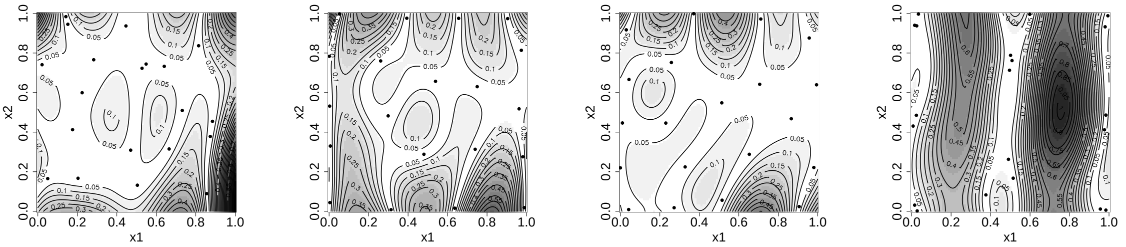
- The MARS models behave poorly. The `R2` criteria does not exceed 0.65. On this example, the obtained models hold too few cuts and thus cannot represent the target function correctly.

- The PolyMARS models are a slightly better but the dependency with respect to the design is very important. In the particular case of the non-optimized Strauss, the points being clustered, there is not enough information for a piecewise linear model to adequately represent the function. As the PolyMARS model relies on some constraints during its construction, this model generally outperforms the MARS model.

- We can see that the linear models cannot fully fit the function f: the `R2` criteria do not exceed 0.677. This model is not sufficiently flexible.

- The additive model (with cubic splines) is more flexible than the linear model. It can consequently represent more complex shapes. However, in the present case, the model cannot account for interactions. This is its principal limitation.

- For all the presented designs, kriging is able to outperform the additive model since it can capture interactions and non-linearities.



(a) Random design (b) Dmax (c) Optimized Strauss (d) Non-opt. Strauss

Figure 18: Representation of the error between kriging models and the real function. The contour lines correspond to the absolute values of the difference between the real function and the fitted one. Small errors are represented in white and dark regions correspond to large errors.

A focus on the kriging interpolations for the four designs is proposed in Figure 18. The contour plot of the error between the real function and the model is represented for each design. By construction of the metamodel, the error on the points of the experimental design (represented by points) equals zero.

4. An industrial case study in dimension 5

Among all the case studies proposed by the industrial partners of the DICE consortium, we present that proposed by IRSN (Institut de Radioprotection et de Sûreté Nucléaire) to illustrate the main functionalities of the **DiceEval** package. Data sets of this case study are provided with the package. Its dimension makes it a good example problem for the **DiceEval** functionalities. All the R commands can be found in the demonstration file attached to the **DiceEval** package:

```
R> library("DiceEval")
R> demo("IRSN5D", package = "DiceEval")
```

4.1. Context of the study and description of the data set

Nuclear criticality safety assessments are based on an optimization process to search for physical conditions adversely affecting safety in a given range of parameters of a system involving fissile materials. In the current example, the criticality coefficient (namely k-effective denoted *keff* in what follows) models the nuclear chain reaction trend:

- *keff* >1 means increasing neutron production leading to a potentially uncontrolled chain reaction that can have serious safety consequences.
- *keff* $=1$ means a stable neutron population as required in nuclear reactors.
- *keff* <1 is the safety state required for all unused fissile materials, for example for fuel storage.

Besides its fissile material geometry and composition, the criticality of a system is extremely sensitive to physical parameters like water density, geometrical perturbations or the properties of structural materials (such as concrete). A typical criticality safety assessment is therefore supposed to verify that the *keff* cannot reach the critical value of 1 (in practice the limit value used is 0.95) for a given hypothesis on these parameters.

The benchmark system is an assembly of four fuel rods contained in a reflecting hull. Regarding criticality safety hypothesis, the main parameters are the uranium enrichment of fuel (namely `e`, U235 enrichment, varying in [3%, 7%]), the rods assembly geometrical characteristics (namely `p`, the pitch between rods, varying in [1.0, 2.0] cm and `l`, the length of fuel rods, varying in [10, 60] cm), the water density inside the assembly (namely `b`, varying in [0.1, 0.9]) and the hull reflection characteristics (namely `r`, reflection coefficient, varying in [0.75, 0.95]). In this criticality assessment, the MORET Monte Carlo simulator (Fernex, Heulers, Jacquet, Miss, and Richet 2005) is used to estimate the criticality coefficient of the fuel storage system using these parameters (among others) as numerical inputs. The output *keff* is returned as a Gaussian density, the standard deviation of which appears to be negligible with respect to input parameters sensitivity.

4.2. Package DiceEval in use

Now, let us recall that **DiceEval** deals with the construction of different metamodels on a learning set and the comparison of their prediction performances on a test set if available and by cross validation otherwise.

Data of the IRSN case

In this case study, the learning set is a Strauss design with 50 points and the test set is a grid of 324 points. Unfortunately, we cannot provide the parameters that were used to generate the Strauss design. The data set is nevertheless used because of the availability of the data.

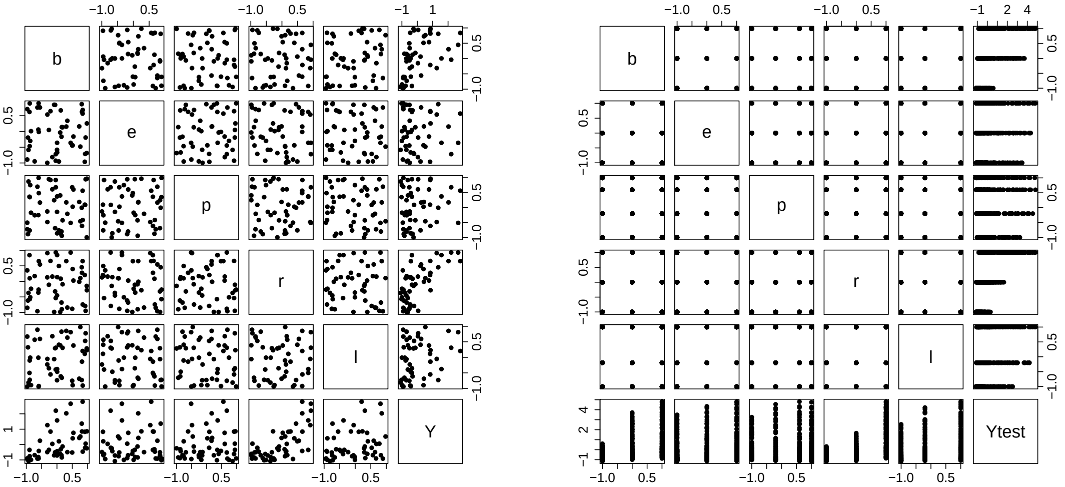
```
R> data("dataIRSN5D", package = "DiceEval")
R> X <- dataIRSN5D[, -6]
R> Y <- dataIRSN5D[, 6]
R> data("testIRSN5D", package = "DiceEval")
R> Xtest <- testIRSN5D[, -6]
R> Ytest <- testIRSN5D[, 6]
```

A simple transformation is applied to translate the experimental design on $[-1,1]^5$. For factorial designs with two levels, this transformation ensures orthogonality.

In Figure 19, we can observe that the projective points of the training set visually fill the two-dimensional space whereas the test set exhibits clusters. However, the study was conducted with this test data set because of its availability.

As can be seen in Figure 19, the link between the response and the entries is not obvious except in the `r` direction where an exponential relation emerges.

In the next four subsections, we show how to carry out model construction, check the quality, find the best parametrization of metamodels and compare metamodels. First, the main functionalities of **DiceEval** are illustrated on the linear model with linear trend, without interactions and quadratic terms. Then, we present some specific calls and functions for other types of models.



(a) IRSN training data set

(b) IRSN test set

Figure 19: Two-dimensional projections of the designs of experiments (training and test sets) for the 5D IRSN application.

Adjustment

An important contribution made by the package is the use of a unique function called `modelFit` to fit different types of metamodels. This function makes it possible to fit linear and additive models, stepwise versions of linear model, two models with piecewise linear functions (namely MARS and PolyMARS) and kriging models.

For all metamodels, the call depends on the `type` argument which specifies the model's family. Furthermore, some other arguments are sometimes needed: `formula` for linear and additive models, `formula` and `penalty` for stepwise linear models, `degree` for the MARS models and `gcv` for the PolyMARS models. In the case of the linear model, we only have to specify the input and output data, the type `"Linear"` and the `formula`.

```
R> modLm <- modelFit(X, Y, type = "Linear", formula = Y ~ .)
R> names(modLm)

[1] "data"    "type"    "formula" "model"
```

The output of `modelFit` is a list of arguments containing the data, the fitted model of the `type` specified in this argument and other parameters related to the model. The `summary` procedure can still be called to gain access to information on the fitted models.

```
R> summary(modLm$model)

Call:
lm(formula = fmla, data = data)

Residuals:
     Min       1Q   Median       3Q      Max 
-0.84585 -0.23768 -0.08712  0.20903  0.94854 
```

	Training	*Test*
R2	0.849	0.756
RMA	0.948	1.313
MAE	0.300	0.549
RMSE	0.384	0.703

Table 7: Quality criteria for the linear model.

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.12058    0.05889   2.048 0.046592 *
b            0.77530    0.09404   8.245 1.80e-10 ***
e            0.24118    0.09709   2.484 0.016865 *
p            0.27431    0.09469   2.897 0.005853 **
r            1.22144    0.09983  12.235 9.35e-16 ***
l            0.41519    0.09972   4.164 0.000144 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.4097 on 44 degrees of freedom
Multiple R-squared: 0.8493,     Adjusted R-squared: 0.8322
F-statistic: 49.59 on 5 and 44 DF,  p-value: < 2.2e-16
```

In this example, the linear model with a linear trend fits the training data well (the R2 is equal to 0.85). Once the metamodel is validated on the learning set, it is important to check its prediction quality.

Checking for prediction quality

As seen before, different criteria, all based on residuals, are available in the package: R2, RMA, MAE and RMSE (see Equations 5–7). These functions need two arguments: the first is the vector of the exact values and the second is the vector of the fitted values at the same points. The criteria can also be evaluated on learning and test sets.

The results obtained here are as expected. First of all, R2 is decreasing from learning to test sets (the variance explained by the model is higher on the training set). Moreover, RMA, MAE and RMSE are increasing from learning to test sets (errors are smaller in adjustment than in prediction). Recall that a perfect fit corresponds to 1 for R2 and 0 for the other criteria. Ideally, the criteria should be consistent between both training and test sets. In the latter case, the value of the RMSE then corresponds to the real model bias.

However, a test set is often not available. Thus, cross validation is commonly used in order to assess the prediction error variance. The crossValidation function measures the Q2 (defined by (8)) of a previously fitted model for a given number of folds. The results of crossValidation on a linear model and 10 folds are obtained by the following commands.

```
R> crossValidation(modLm, K = 10)$Q2
```

```
[1] 0.8092859
```

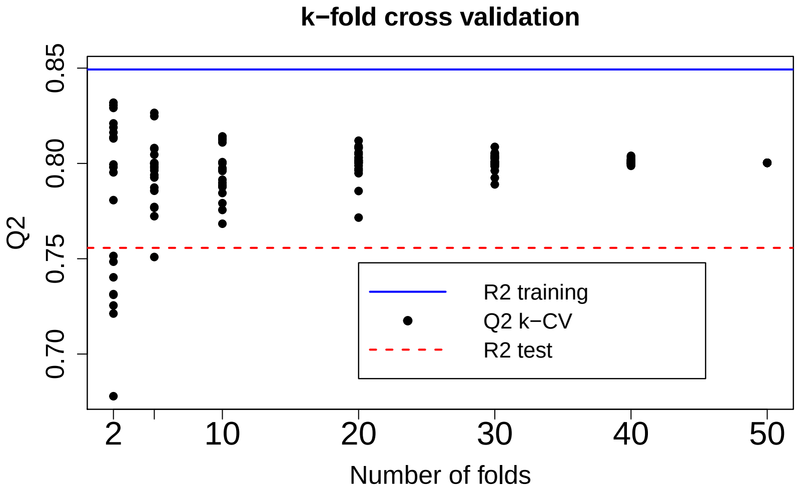


Figure 20: Estimated `Q2` with respect to the number of folds for a linear model.

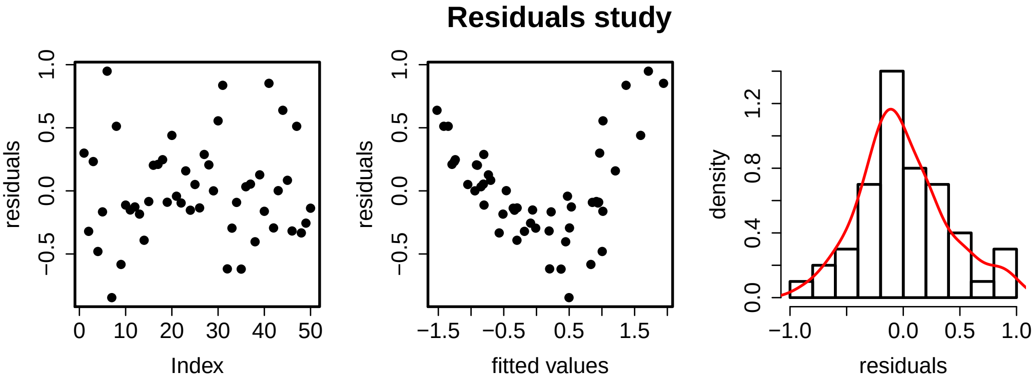


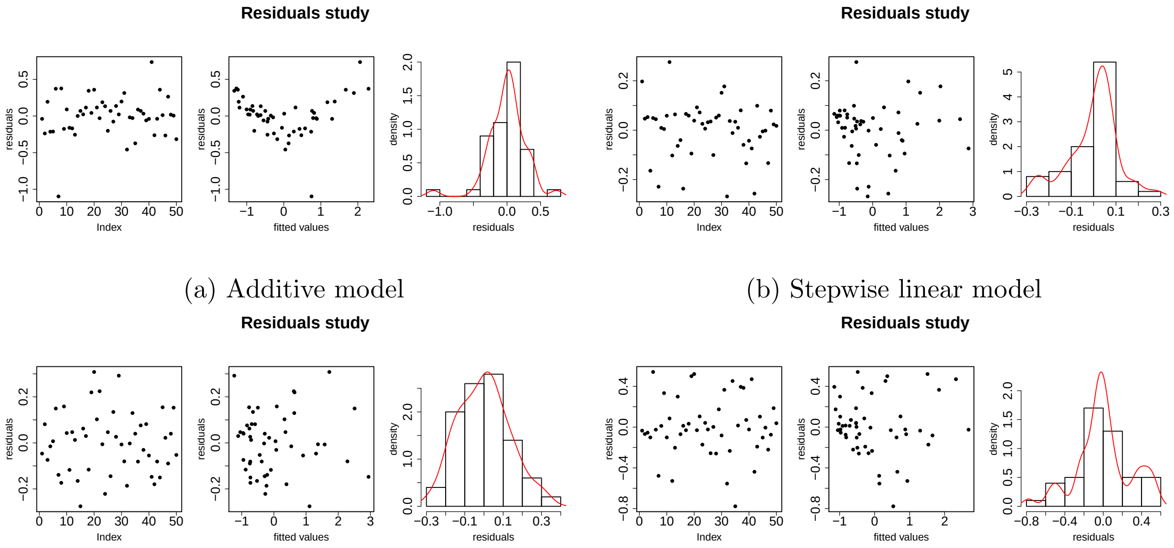
Figure 21: Residuals study for the linear model.

Moreover, the `testCrossValidation` function allows the assessment of the `Q2` variability (Figure 20). For each number of folds, 10 different partitions are computed to visualize `Q2` variability. We consider here 2, 5, 10, 20, 30, 40 and 50 folds.

As can be observed in Figure 20, the `R2` is effectively higher on the learning set (0.85) than on the test set (0.76). The cross validation method provides a better idea of what the `R2` will be in prediction. It is recalled that the tuning parameters of the model are set during cross validation (see Section 3.4). Overestimation of R2 is then expected. The function `residualsStudy` that provides the visualization of the residuals can also help check for predictive quality.

```
R> residualsStudy(modlm)
```

In the case of the linear model, the middle plot of Figure 21 shows that some important effects are not considered in the model. Indeed, dependence can be observed between residuals and predictors. A possible explanation is that non-linearities or interactions should be added to the model.



(a) Additive model

(b) Stepwise linear model

(c) MARS model

(d) PolyMARS model

Figure 22: Residuals study for different types of metamodels.

The study of the residuals of the additive model still shows the same behavior, i.e., dependence between residuals and inputs (see Figure 21). As the additive model allows the assessment of non-linearity in each direction, we conclude that the major missing effects are due to interactions.

```
R> library("gam")
R> modAm <- modelFit(X, Y, type = "Additive", formula = formulaAm(X, Y))
R> residualsStudy(modAm)
```

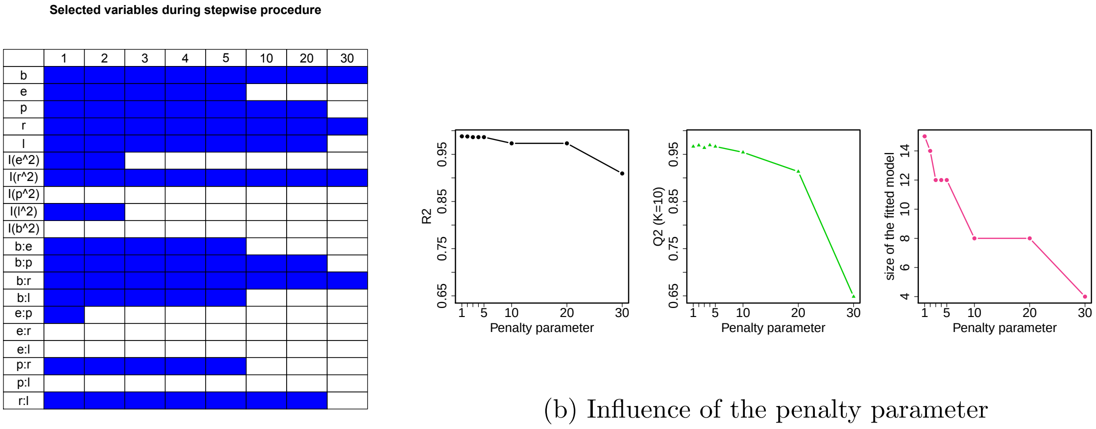
This dependence between the residuals and the response disappears when considering models that take interactions into account. This is, for example, the case of the stepwise model built from a trend that contains single effects, interactions and quadratic terms, the MARS model of degree 2 and the PolyMARS model.

```
R> modStep <- modelFit(X, Y, type = "StepLinear",
+    formula = Y ~ .^2 + I(e^2) + I(r^2) + I(p^2) + I(l^2) + I(b^2),
+    penalty = 2)
R> library("mda")
R> modMARS <- modelFit(X, Y, type = "MARS", degree = 2)
R> library("polspline")
R> modPolyMARS <- modelFit(X, Y, type = "PolyMARS", gcv = 4)
```

The study of the residuals for each of these three models is presented in Figure 22.

Determination of the best parameterization

We have already seen that several metamodels have to be parameterized. This is the case for the `degree` argument in the MARS model which indicates whether interactions are allowed or



Selected variables during stepwise procedure

	1	2	3	4	5	10	20	30
b	■	■	■	■	■	■	■	■
e	■	■	■	■	■			
p	■	■	■	■	■	■	■	
r	■	■	■	■	■	■	■	■
l	■	■	■	■	■	■	■	
I(e^2)	■	■						
I(r^2)	■	■	■	■	■	■	■	■
I(p^2)								
I(l^2)	■	■						
I(b^2)								
b:e	■	■	■	■	■			
b:p	■	■	■	■	■	■	■	
b:r	■	■	■	■	■	■	■	■
b:l	■	■	■	■	■			
e:p	■							
e:r								
e:l								
p:r	■	■	■	■	■			
p:l								
r:l	■	■	■	■	■	■	■	

(b) Influence of the penalty parameter

(a) Selected variables

Figure 23: The impact of the penalty parameter on the stepwise procedure. (a) variables included (rows = variables, columns = penalty) and (b) quality criteria.

not. PolyMARS similarly requires the value of the `gcv` penalization parameter. And finally, there is also a penalization parameter for the stepwise version of linear models.

In practice, these parameters are difficult to choose. Therefore, the package provides the user with functions that produce plots for decision support purposes. In the IRSN case study, the output of the routines is given in Figure 23 for the stepwise model and in Figure 24 for the PolyMARS model.

Let us focus now on the stepwise procedure for linear models.

```
R> out <- stepEvolution(X, Y,
+    Y ~ .^2 + I(e^2) + I(r^2) + I(p^2) + I(l^2) + I(b^2),
+    P = c(1, 2, 3, 4, 5, 10, 20, 30))
R> out$Q2
```

```
[1] 0.9665380 0.9690667 0.9637816 0.9694529 0.9665024 0.9541808
[7] 0.9133722 0.6483670
```

The visualization of the `stepEvolution` procedure indicates how sensitive the response is to inputs. Here, it can be observed that `b`, `r`, `r^2` and the interaction `b:r` are very influential on the response. Their effects are still present for a very high penalty (`penalty` = 30). Then, come `p`, `l`, the interactions `b:p` and `r:l`, and so on with a decreasing penalty. The graphic shows that even for low values of the penalty parameter, some interactions have no influence on the response, namely `b:e`, `b:l`, `e:r`, `e:l` and `p:l`.

The right part of Figure 23 shows the influence of the `penalty` parameter on the quality of the metamodel. In the first plot, it can be seen that `R2` is progressively decreasing with the `penalty` parameter whereas `Q2` ($K = 10$) reaches a maximum for a penalty parameter $P = 4$ (see the `Q2` numerical values from the `stepEvolution` output). This particular area corresponds to the best model containing enough variables to represent major phenomena but

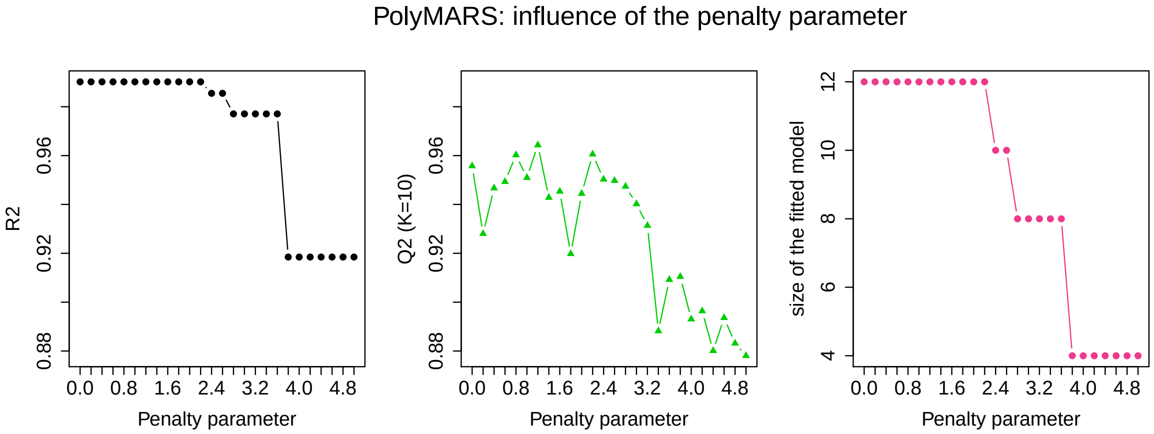


Figure 24: Study of the influence of the penalty parameter on the performance of a PolyMARS model fitted on the learning set. The plot represents the `R2` computed on the training set (left-hand side), the evolution of `Q2` estimated using 10-folds cross validation (middle) and the number of terms (right-hand side).

not too many in order to avoid overfitting. The last plot represents the change in the model size reduction according to the penalty parameter.

A similar function is implemented for the PolyMARS procedure. As mentioned by Kooperberg *et al.* (1997), the role of the penalty parameter `gcv` is crucial during the fitting procedure. For example, a small value of `gcv` induces no penalty on the acceptance criterion and thus the resulting model often contains many terms. `penaltyPolyMARS` is a procedure that helps the user choose the value of this parameter.

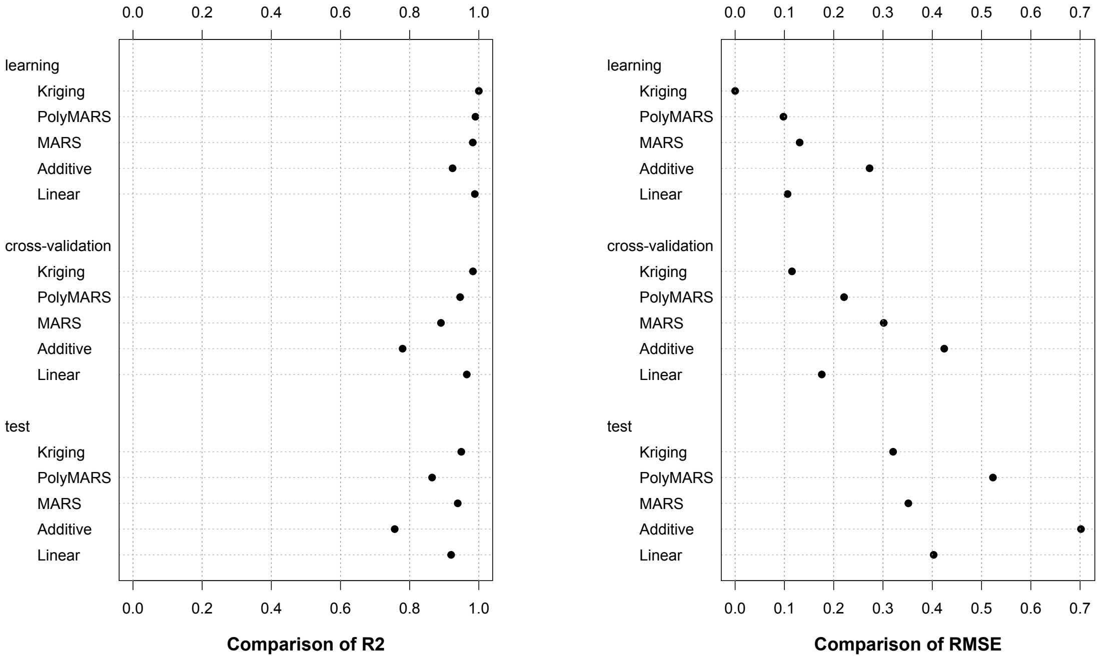
```
R> Crit <- penaltyPolyMARS(X, Y, graphic = TRUE)
```

Graphical output is represented in Figure 24. It can be seen in Figure 24 that `R2` decreases as `gcv` increases. Indeed, when `gcv` increases the model contains fewer terms and is then less flexible, therefore the fitted model is less close to the data. The change in `R2` on the test set is not the same as it first increases and then decreases with `gcv`. When `gcv` is low, the models contain mainly useless terms which leads to errors. When `gcv` increases, these useless terms disappear and the `R2` test increases consequently. If `gcv` keeps growing, some useful terms are dropped from the model, accordingly dragging the `R2` down.

The value of `gcv` is chosen by cross validation. It corresponds to the highest `Q2` and to the model with fewer terms. For this example, we choose a value of 2.2 for `gcv`, the value that represents a good trade-off for all the evaluation measures.

Model comparison

The different metamodels described above have advantages and drawbacks. A good overview of the previous results is given by the `modelComparison` routine. All the information needed to fit the metamodels that are to be compared is given as arguments.



(a) Comparison according to the `R2` criterion

(b) Comparison according to the `RMSE` criterion

Figure 25: Graphical comparison of the quality of metamodels for the IRSN case study. On the training set (top of the figure), there are only small differences between the studied models. In terms of prediction (at the bottom of the figure), note that `MARS` and `Linear` models are a little bit more precise.

```
R> crit  <- modelComparison(X, Y,
+    type = c("Linear", "Additive", "MARS", "PolyMARS", "Kriging"),
+    test = dataTest, degree = 2, gcv = 2.2,
+    formula = c(Y ~ .^2 + I(e^2) + I(r^2) + I(p^2) + I(l^2) + I(b^2),
+    formulaAm(X, Y), Y ~ .))
```

Let us detail the arguments of this function: four types of models will be compared. First, a linear model corresponding to the formula `~ .^2 + I(e^2) + I(r^2) + I(p^2) + I(l^2) + I(b^2)`. Second, an additive model with a standard formula with splines in all principal directions (directly obtained by using the hidden function `formulaAm`). Finally, a MARS model of degree 2 and a PolyMARS model with a penalty parameter `gcv` equal to 2.2. Results are presented in Figure 25.

As can be seen on the top of Figure 25, the `R2` criteria evaluated on the learning set are high for all models and the `RMSE` are low respectively. The prediction quality is lower when checked with the `Q2` criteria, and lower and more dispersed when observed on the test set. The graphical representation of the criteria emphasizes the good results of the kriging model. In the test set, the `Additive` model is the worst. This is explained by the fact that it does not take into account interactions whereas the others do. Moreover, in this example, we have already seen that interactions play a major role. This difference between the behaviors of the metamodels can also be partially observed on `Q2` which is lowest for the additive model.

5. Conclusions

DICE (Deep Inside Computer Experiments) was a research project that addressed many aspects of exploration and stochastic analysis of computer codes. Several issues have been addressed from designing experimental sets of points, metamodeling to optimization, sensitivity analysis and uncertainty quantification. In this paper, we focus on the two first issues (design and metamodeling) through the presentation of two R packages: **DiceDesign** and **DiceEval**.

DiceDesign provides new approaches to generating space filling designs. The first approach is based on maximization of the entropy, i.e., maximization of the determinant of the covariance matrix. The second approach relies on a stochastic generation of points based on Gibbs processes using a method based on Markov chain Monte Carlo. This simulation is more difficult to calibrate since a Gibbs distribution and Markov chain Monte Carlo parameters have to be chosen with respect to the desired properties of the design. However, these designs yield good results in practice and this approach can offer other possibilities such as generating non-stationary designs. Extensions are not yet implemented in the package. Furthermore, **DiceDesign** provides a set of criteria to assess the quality of the generated designs and an associated graphical tool.

The goal of the second package presented in this paper is to unify and to simplify metamodeling when different types of models are tested. Furthermore, **DiceEval** provides numerical and graphical tools to evaluate the quality of the estimation.

These two packages have been illustrated on a two-dimensional analytical case and on a five-dimensional real case study. The packages were built according to the needs of the project and routines were then validated on a wide range of industrial studies from oil to the automotive field.

Acknowledgments

This work was conducted within the framework of the DICE consortium between ARMINES, Renault, EDF, IRSN, ONERA and Total S.A. The authors are very grateful to Y. Richet (IRSN) who provided the data for the demonstration example. The authors wish to thank O. Roustant, L. Carraro, Y. Deville and D. Ginsbourger for their contributions.

We also wish to thank the anonymous referees for several helpful comments and suggestions that improved and clarified the presentation of this work.

References

Carnell R (2012). ***lhs**: Latin Hypercube Sample.* R package version 0.10, URL http://CRAN.R-project.org/package=lhs.

Chalabi Y, Dutang C, Savicky P, Wuertz D (2014). ***randtoolbox**: Toolbox for Pseudo and Quasi Random Number Generation and RNG Tests.* R package version 1.16, URL http://CRAN.R-project.org/package=randtoolbox.

Chen VCP, Tsui KL, Barton RR, Allen JK (2003). "A Review of Design and Modeling in Computer Experiments." In *Handbook of Statistics*, volume 22, pp. 231–261.

Cressie N (1993). *Statistics for Spatial Data.* John Wiley & Sons.

Damblin G, Couplet M, Iooss B (2013). "Numerical Studies of Space Filling Designs: Optimization Algorithms and Subprojection Properties." *Journal of Simulation*, **7**(4), 276–289.

Dupuy D, Helbert C (2014). ***DiceEval**: Construction and Evaluation of Metamodels.* R package version 1.3, URL http://CRAN.R-project.org/package=DiceEval.

Faure H (1982). "Discrépance de suites associées à un système de numération (en dimension *s*) [Discrepancy of Sequences Associated with a Number System (in Dimension *s*)]." *Acta Arithmetica*, **41**(4), 337–351.

Fernex F, Heulers L, Jacquet O, Miss J, Richet Y (2005). "The MORET 4B Monte Carlo Code – New Features to Treat Complex Criticality Systems." In *M&C International Conference on Mathematics and Computation Supercomputing, Reactor Physics and Nuclear and Biological Application.*

Franco J, Bay X, Dupuy D, Corre B (2008). "Planification d'expériences numériques à partir du processus ponctuel de Strauss." URL http://hal.archives-ouvertes.fr/hal-00260701/fr/.

Franco J, Dupuy D, Roustant O (2014). ***DiceDesign**: Designs of Computer Experiments.* R package version 1.6, URL http://CRAN.R-project.org/package=DiceDesign.

Franco J, Vasseur O, Corre B, Sergent M (2009). "Minimum Spanning Tree: A New Approach to Assess the Quality of the Design of Computer Experiments." *Chemometrics and Intelligent Laboratory Systems*, **97**(2), 164–169.

Friedman J (1991). "Multivariate Adaptative Regression Splines." *The Annals of Statistics*, **10**(1), 1–141.

Ginsbourger D, Roustant O (2015). ***DiceOptim**: Kriging-Based Optimization for Computer Experiments.* R package version 1.5, URL http://CRAN.R-project.org/package=DiceOptim.

Halton JH (1960). "On the Efficiency of Certain Quasi-Random Sequences of Points in Evaluating Multi-Dimensional Integrals." *Numerische Mathematik*, **2**(1), 84–90.

Hammersley JM (1960). "Monte-Carlo Methods for Solving Multivariate Problems." *The Annals of the New York Academy of Sciences*, **86**(3), 844–874.

Hastie T (2013). ***mda**: Mixture and Flexible Discriminant Analysis.* R package version 0.4-4, URL http://CRAN.R-project.org/package=mda.

Hastie T (2014). ***gam**: Generalized Additive Models.* R package version 1.09.1, URL http://CRAN.R-project.org/package=gam.

Hastie T, Tibshirani R (1991). *Generalized Additive Models.* Chapman and Hall.

Hastie T, Tibshirani R, Friedman J (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* 2nd edition. Springer-Verlag.

Hickernell F (1998). "A Generalized Discrepancy and Quadrature Error Bound." *Mathematics of Computation*, **67**(221), 299–322.

Johnson ME, Moore LM, Ylvisaker D (1990). "Minimax and Maximin Distance Designs." *Journal of Statistical Planning and Inference*, **26**(2), 131–148.

Jones D, Schonlau M, Welch W (1998). "Efficient Global Optimization of Expensive Black-Box Functions." *Journal of Global Optimization*, **13**(4), 455–492.

Kooperberg C (2013). ***polspline**: Polynomial Spline Routines.* R package version 1.1.9, URL http://CRAN.R-project.org/package=polspline.

Kooperberg C, Bose S, Stone CJ (1997). "Polychotomous Regression." *Journal of the American Statisitical Association*, **92**(437), 117–127.

McKay MD, Beckman RJ, Conover WJ (1979). "A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code." *Technometrics*, **21**(2), 239–245.

Niederreiter H (1987). "Low-Discrepancy and Low-Dispersion Sequences." *Journal of Number Theory*, **30**(1), 51–70.

Pleming JB, Manteufel RD (2005). "Replicated Latin Hypercube Sampling." In *46th Structures, Structural Dynamics & Materials Conference (16–21 April 2005) – AIAA 2005-1819.*

Pronzato L, Müller WG (2012). "Design of Computer Experiments: Space Filling and Beyond." *Statistics and Computing*, **22**(3), 681–701.

R Core Team (2015). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Richet Y, Deville Y, Chevalier C (2013). ***DiceView**: Plot Methods for Computer Experiments Design and Models.* R package version 1.3-1, URL http://CRAN.R-project.org/package=DiceView.

Roustant O, Franco J, Carraro L, Jourdan A (2010). "A Radial Scanning Statistic for Selecting Space-Filling Designs in Computer Experiments." In A Giovagnoli, AC Atkinson, B Thorsney, C May (eds.), *MODA-9 – Advances in Model-Oriented Design and Analysis*, pp. 189–196. Springer-Verlag.

Roustant O, Ginsbourger D, Deville Y (2012). "**DiceKriging**, **DiceOptim**: Two R Packages for the Analysis of Computer Experiments by Kriging-Based Metamodelling and Optimization." *Journal of Statistical Software*, **51**(1), 1–55. URL http://www.jstatsoft.org/v51/i01/.

Roustant O, Ginsbourger D, Deville Y (2015). ***DiceKriging**: Kriging Methods for Computer Experiments.* R package version 1.5.4, URL http://CRAN.R-project.org/package=DiceKriging.

Santiago J, Claeys-Bruno M, Sergent M (2012). "Construction of Space-Filling Designs using WSP Algorithm for High Dimensional Spaces." *Chemometrics and Intelligent Laboratory Systems*, **113**, 26–31.

Santner TJ, Williams BJ, Notz WI (2003). *The Design and Analysis of Computer Experiments*. Springer-Verlag.

Shewry MC, Wynn HP (1987). "Maximum Entropy Sampling." *Journal of Applied Statistics*, **14**(2), 165–170.

Sobol' IM (1967). "On the Distribution of Points in a Cube and the Approximate Evaluation of Integrals." *USSR Computational Mathematics and Mathematical Physics*, **7**(4), 56–117.

Stein M (1987). "Large Sample Properties of Simulations using Latin Hypercube Sampling." *Technometrics*, **29**(2), 143–151.

Wuertz D (2013). ***fOptions**: Basics of Option Valuation*. R package version 3010.83, URL http://CRAN.R-project.org/package=fOptions.

Affiliation:

Delphine Dupuy
UR LSTI – team CROCUS
École Nationale Supérieure des Mines de Saint-Étienne
158, cours Fauriel
42032 Saint-Étienne Cedex 2, France
E-mail: dupuy@emse.fr

Céline Helbert
Institut Camille Jordan – UMR5208
36 av. Guy de Collongue
69134 Ecully Cedex, France
E-mail: Celine.Helbert@ec-lyon.fr

Jessica Franco
Recovery Mechanisms
TOTAL DGEP/GSR/TG/COP/REC
L5 1005
Avenue Larribau
64018 Pau Cedex, France
E-mail: jessica.franco@total.com

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 65, Issue 11 *Submitted:* 2012-07-30
June 2015 *Accepted:* 2014-10-15