GPareto: An R Package for Gaussian-Process Based Multi-Objective Optimization and Analysis

Mickaël Binois
Mines Saint-Étienne

Victor Picheny
INRA

Abstract

The **GPareto** package for R provides multi-objective optimization algorithms for expensive black-box functions and uncertainty quantification methods. Popular methods such as EGO in the mono-objective case relies on Gaussian processes or kriging to build surrogate models. Driven by the prediction uncertainty given by these models, several infill criteria have also been proposed in a multi-objective setup to select new points sequentially and efficiently cope with severely limited evaluation budgets. They are implemented in the package, in addition with estimation of the whole Pareto front location and uncertainty quantification visualization in the design and objective spaces. Finally, it attempts to fill the gap between expert use of the corresponding methods and simple usage, where many efforts have been put on providing graphical visualization, standard tuning and interactivity.

Keywords: kriging, Pareto front, EGO, uncertainty quantification.

1. Introduction

Numerical modeling of complex systems is now an essential process in fields as diverse as natural sciences, engineering, quality or economics. Jointly with modeling efforts, methods have been developed for the exploration and analysis of corresponding simulators, in particular when runs are time consuming. A popular approach in this case is to rely on surrogate models to alleviate the computational expense. Many surrogate models are used in practice: polynomials, splines, support vector regression, radial basis functions, random forests or Gaussian processes (GP). They may be integrated in various optimization strategies, see e.g., Wang and Shan (2007), Santana-Quintero, Montano, and Coello (2010), Tabatabaei, Hakanen, Hartikainen, Miettinen, and Sindhya (2015) and references therein. We focus here on GP-based strategies, which have been recognized as very well-suited for sequential designs of experiments, and in particular in an optimization context (Jones, Schonlau, and Welch 1998; Jones 2001).

The **GPareto** package proposes Gaussian-Process based sequential strategies to solve multi-objective optimization (MOO) problems in a black-box, numerically expensive simulator context. More precisely, it considers the case of models with multiple outputs, $y^{(1)}(\mathbf{x}), \ldots, y^{(q)}(\mathbf{x})$ (where $y^{(i)} : \mathbb{X} \subset \mathbb{R}^d \to \mathbb{R}$), that are optimized simultaneously over a box-constrained domain $\mathbb{X}$. Typically, outputs (or *objectives*) are conflicting (e.g., quality versus quantity, etc.), so there exists no solution where all objectives are minimized at once. The goal is then to iden-

tify the set of optimal compromise solutions, called a Pareto set (Collette and Siarry 2003). Defining that a point $\mathbf{x}^*$ dominates another point $\mathbf{x}$ if *all* its objectives are better (which we denote by $\mathbf{x} \preceq \mathbf{x}^*$ in the following), the Pareto set $\mathbb{X}^*$ is the subset of the non-dominated points in $\mathbb{X}$:

$$\forall \mathbf{x}^* \in \mathbb{X}^*, \forall \mathbf{x} \in \mathbb{X}, \exists k \in \{1, \ldots, q\} \text{ such that } y^{(k)}(\mathbf{x}^*) \leq y^{(k)}(\mathbf{x}).$$

The image of the Pareto set in the objective space, $y^{(1)}(\mathbb{X}^*), \ldots, y^{(q)}(\mathbb{X}^*)$, is called the Pareto front, which is useful to practitioners to select solutions (see Figure 3 for an illustration). In practice, the Pareto set is usually not finite, and optimization strategies aim at providing a finite set that represents $\mathbb{X}^*$ well.

In general, numerical optimization has motivated a substantial activity in the R R Core Team (2019) community: see for instance the CRAN Task View on "Optimization and Mathematical Programming" (Theussl and Borchers 2015) or the recent special JSS issue (Varadhan 2014). However, most works are dedicated to mono-objective optimization with large budgets. For small budgets, the packages **DiceOptim** (Roustant, Ginsbourger, and Deville 2012; Ginsbourger, Picheny, and Roustant 2015) and **tgp** (Gramacy 2007; Gramacy and Taddy 2010) propose GP-based techniques, but for mono-objective problems only. There are a few packages on MOO in general: **nsga2R** (Tsou 2013), **emoa** (Mersmann 2012), **mopsocd** (Naval 2013), **goalprog** (Novomestky 2008) and **mco** (Mersmann 2014), which provide tools and algorithms such as NSGA-II algorithm implementations (Deb, Pratap, Agarwal, and Meyarivan 2002) or hypervolume computations (see Section 2.2). As for methods available for expensive black-box functions optimization, the package **SPOT** (Bartz-Beielstein and Zaefferer 2012) seems to be the only alternative to **GPareto**.

On the other hand, GP-based MOO has recently generated a substantial activity in the statistical and optimization communities, with focuses either on sampling strategies (Ponweiser, Wagner, Biermann, and Vincze 2008; Wagner, Emmerich, Deutz, and Ponweiser 2010; Svenson 2011; Emmerich, Deutz, and Klinkenberg 2011; Picheny 2015; Zuluaga, Sergent, Krause, and Püschel 2013) or on uncertainty quantification (Bhardwaj, Dasgupta, and Deb 2014; Calandra, Peters, and Deisenroth 2014; Binois, Ginsbourger, and Roustant 2015a). **GPareto** aims at filling this gap by making most of the recent approaches available in a unified implementation to both MOO experts and end-users. To this end, a substantial effort has been given to provide graphical visualization and standard tuning, and many entry-points ranging from high-level interfaces to specific method tuning have been made available.

GPareto is built upon the **DiceKriging** (Roustant *et al.* 2012) package dedicated to Gaussian process modeling. Several associated packages deal with various related problems, in particular **DiceOptim** (mono-objective optimization) and **KrigInv** (algorithms for inversion problems) (Chevalier, Picheny, and Ginsbourger 2014a,b). **GPareto** shares many aspects with those packages. This document is also available as a vignette in the package, which is available from the Comprehensive R Archive Network at `https://CRAN.R-project.org/package=GPareto` along with the full PDF documentation.

The remainder of this paper reviews briefly the methods available in the package, describes important implementation aspects and functionalities, and provides illustrations through a few examples.

2. Method

2.1. Principles of Gaussian-process based optimization

We recall here very briefly the scheme common to most GP-based (mono- or multi-objective) optimization, as in the famous EGO algorithm (efficient global optimization) proposed in the seminal article of Jones *et al.* (1998).

The mono-objective case

Let y be the output of the numerical model of interest and $\mathbf{x} \in \mathbb{R}^d$ the inputs to be optimized over. Considering for now that y is a scalar, it is assumed to be a realization of a Gaussian process $F(\mathbf{x})$ with mean $\mu(\mathbf{x})$ and covariance $c(\mathbf{x}, \mathbf{x}')$ known up to some parameters.

Step 1 Generate an initial set of n observations: $y_1 = y(\mathbf{x}_1), \ldots, y_n = y(\mathbf{x}_n)$. Typically, $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ are chosen using a *space-filling* design. A classical rule-of-thumb is to use $n = 10 \times d$.

Step 2 Fit the GP model to the data, by estimating the mean $\mu(\mathbf{x})$ and covariance $c(\mathbf{x}, \mathbf{x}')$. Typically, a parametric form is assumed for those functions, whose parameters are adjusted, e.g., by using maximum likelihood estimation. The GP model is the distribution of $Y(\mathbf{x})$ conditional on the observations $y_1, \ldots, y_n$, with plugged-in mean μ and covariance c.

Step 3 A new point $\mathbf{x}_{n+1}$ is chosen as the maximizer of a so-called *infill criterion* which is based on the GP model. This step requires running an inner optimization loop to find the best point over $\mathbb{R}^d$.

Step 4 A new observation $y_{n+1} = y(\mathbf{x}_{n+1})$ is obtained by running the simulator and the GP model is updated by conditioning on y_{n+1}. At this step, the estimates of μ and c might be updated.

Steps 3 and 4 are repeated until the simulation budget is exhausted or when a stopping criterion is met.

There are many R packages to perform Step 1, see for instance **planor** (Kobilinsky, Bouvier, and Monod 2015), **DiceDesign** (Dupuy, Helbert, and Franco 2015), or **lhs** (Carnell 2016). For Step 2, **GPareto** relies on the **DiceKriging** package, which offers a choice of mean and covariance functions. The model parameters estimation is based on maximum likelihood, see Roustant *et al.* (2012) for details.

Step 3 defines the sampling strategy, as the infill criterion determines the balance between exploration (search for new solutions) and exploitation (local improvement around existing observations). The EGO algorithm is based on the so-called *expected improvement* (EI) criterion. The improvement is defined as the difference between the current minimum of the observations and the new function value, such that for a GP model, EI is the conditional expectation of the improvement provided by a new observation $Y(\mathbf{x})$:

$$EI(\mathbf{x}) = \mathbb{E}\left[\max\left(0, \min_{1 \leq i \leq n}(y_i - Y(\mathbf{x}))\right) \middle| Y(\mathbf{x}_1) = y_1, \ldots, Y(\mathbf{x}_n) = y_n\right],$$

which has a closed form expression (see Jones *et al.* 1998, for calculations).

Noisy objectives

In many optimization problems, the objective cannot be evaluated exactly but through a "noisy" procedure, that is, one only has access to measurements of the form $f_i = y(\mathbf{x}_i) + \varepsilon_i$. A classical hypothesis, adopted here, is to assume independent Gaussian centered noise, that is: $\varepsilon_i \sim \mathcal{N}(0, \tau_i^2)$. GP modeling naturally adapts to this case (see for instance **?**), and the package **DiceKriging** offers options to take noise into account.

However, the EGO algorithm may not be used directly; **?** provides a review of the extensions that have been proposed to handle noisy objectives. Within those, the *reinterpolation* approach of **?** is attractive, since it amounts to building a secondary noiseless GP that can be directly used with EGO. As shown in **?**, this approach can be readily applied to the multi-objective case, and is implemented in **GPareto**.

The multi-objective case

When multiple objectives are considered (y has values in $\mathbb{R}^q$), Steps 2 and 3 need to be modified. Let us remark first that it is possible to go back to a scalar problem and apply standard methods, for instance by relying on objectives aggregation (Knowles 2006; Zhang, Liu, Tsang, and Virginas 2010) or modeling desirability functions (Henkenjohann and Kunert 2007). However, these have been found to be relatively poor solutions in practice (Henkenjohann and Kunert 2007; Svenson 2011).

GPareto focuses on approaches where GP models are fitted independently to each objective. Although it is possible to account for correlation between the different objectives, for instance using co-kriging models (e.g., Álvarez, Rosasco, and Lawrence 2011), experimental results in Svenson (2011); Kleijnen and Mehdad (2014) suggested that it provides little benefit compared to the additional complexity.

Choosing infill points from a set of GP models is a complex question (see Section 2.2). Within **GPareto**, we focus on approaches that compute a single infill criterion from the list of models. Hence, Step 3 is identical to the mono-objective case, provided that an adequate infill criterion is used.

2.2. Review of surrogate-based and Bayesian multi-objective optimization

In the mono-objective case, the expected improvement criterion evaluates the potential gain of an additional point in terms of the expected decrease over the best observation so far. In a similar fashion, a multi-objective improvement function can be defined by estimating the expected "progress" brought by a new observation (relatively to the current set of non-dominated observations $\mathcal{P}_n$).

This leaves room to put the focus either on a good coverage, on extremities, or on convergence toward the actual Pareto front, for which specific metrics, such as the *hypervolume* or *epsilon indicators*, have been proposed (see e.g., Svenson 2011; Emmerich *et al.* 2011). Specifically, the *hypervolume* improvement is the increment of the volume contained between the Pareto front and a reference point in the objective space, when a non-dominated point is added. The *epsilon* increment is the smallest scalar that must be added to components of a new point (in the objective space) such that it is dominated by the current Pareto front. An illustrative example is given in Figure 1.

These indicators, among others, have been used to define generalizations of the expected

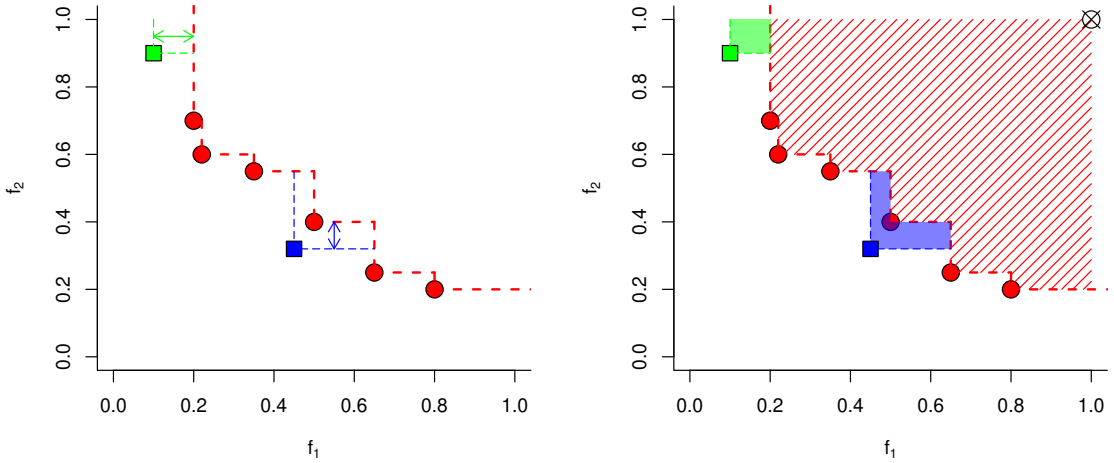

Figure 1: Comparison of additive-epsilon (left, arrows) and hypervolume (right, filled areas) improvements for two possible new observations (green and blue) to the current Pareto front (red points). The reference point for hypervolume computations is the black crossed circle. In terms of epsilon improvement, the green point is more interesting as it is farther away from the Pareto front, but the blue point is better in terms of volume increment.

improvement. Empirical comparisons showed the clear superiority of some approaches to others (Svenson 2011; Wagner *et al.* 2010), but no global consensus on a particular improvement function. In **GPareto**, two infill criteria derived from this point of view are available: the *expected hypervolume improvement* (EHI; Emmerich *et al.* 2011) and *expected maximin improvement* (EMI; Svenson and Santner 2016, related to the epsilon indicator). See the corresponding references for the technical details.

Two alternatives have been included in **GPareto** as well. First, in the *SMS-EGO* approach (S-metric selection EGO; Ponweiser *et al.* 2008; Wagner *et al.* 2010), the improvement is calculated as the hypervolume added to the current Pareto front by the lower confidence bound of the prediction at $\mathbf{x}$, hence it is closely related, but not equal to the EHI. To avoid large plateaus of zero improvement, an adaptive penalization is provided in regions where the lower confidence bound is dominated.

Finally, the *stepwise uncertainty reduction* (SUR) criterion of Picheny (2015) is in turn concerned with the probability of non-domination (also called probability of improvement), that is, the probability of a point not to be dominated by the current Pareto set: $\mathbb{P}\left[\mathbf{x} \npreceq \mathbf{X}_n\right]$. Intuitively, regions in the design space with non-null probabilities indicate a potential improvement for the Pareto front, and the improvement considered is the reduction of the average of this probability over the design space.

These sequential infill criteria share the common trait that they do not provide a continuous representation of the Pareto front but only consider the current set of non-dominated observations. This point is addressed in the following with a quantification of the uncertainty on both the Pareto set and front.

2.3. Uncertainty quantification

With limited evaluation budgets, the non-dominated solutions in the objective and variable spaces may not give a very precise or dense approximation of the Pareto front and set. However, the Gaussian process framework allows us to overcome this limitation by providing an uncertainty quantification of the optimization results.

Pareto front (objective space)

One straightforward idea is to use the surrogate models to give an estimate of the Pareto front, as is done e.g., in Calandra *et al.* (2014). While being fast, this approach is very dependent on the quality of the surrogates and there is no measure of uncertainty associated. In Binois *et al.* (2015a), an alternative relying on conditional simulations of Gaussian process models is detailed, which provides an estimate of the Pareto front and an associated measure of uncertainty.

In short, it exploits the capacity of the GP models to generate different possible realizations $S_1^{(1)}, \ldots, S_1^{(q)}, \ldots, S_N^{(1)}, \ldots, S_N^{(q)}$ for the outputs conditioned by the observations, i.e., conditional simulations, see Figure 2. For each path, a Pareto front is obtained (say, $\mathcal{P}^{(1)}, \ldots, \mathcal{P}^{(N)}$). Then, the set of fronts are used to define an average set $\bar{\mathcal{P}}$ estimating the true Pareto front while the deviation from this set is used as a measure of uncertainty. Note that handling sets of conditional Pareto fronts as performed in Binois *et al.* (2015a) requires the use of random closed sets theory (Molchanov 2005); in particular, the estimator and uncertainty measure used are the Vorob'ev expectation and deviation, respectively. Visually, representing the deviation for each random Pareto front directly illustrates which parts of the Pareto front are precisely known or not (see Figure 5).

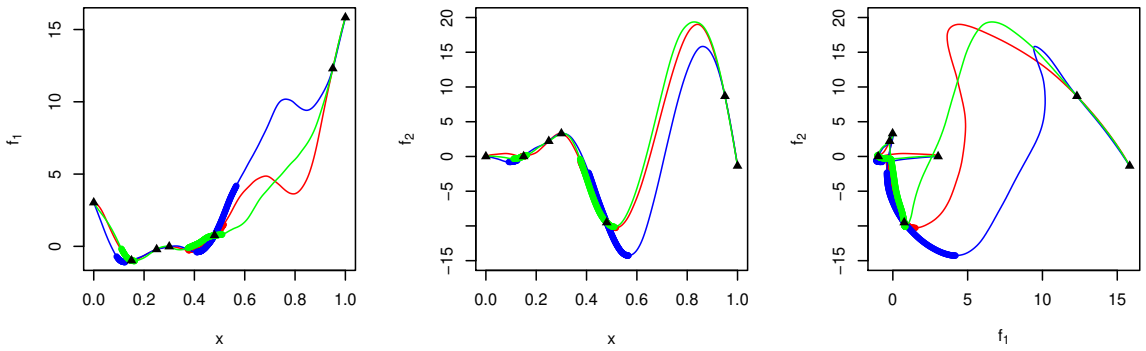

Figure 2: Left and center: three conditional (i.e., interpolating at observations) simulations of objectives f_1 and f_2, respectively, based on GP modeling. Right: corresponding images in the objective space. Pareto sets and fronts are shown in bold.

As described in Binois *et al.* (2015a), the current version of this approach requires conditional simulations on discrete sets of inputs (for instance, a grid or a space-filling sample, which is the solution adopted in **GPareto**, see Section 3.3). This set must be large to ensure that no important potential solution is missed, which makes this approach computationally intensive.

Pareto set (variable space)

In a similar fashion, returning a smooth estimate of the entire Pareto set $\mathbb{X}^*$ may be useful to practitioners. We propose here to rely on two complementary approaches.

First, conditional simulations can be used here: From each set of GP realizations, the Pareto set $\mathbb{X}_i^*$ can be obtained. Then, the sets $\mathbb{X}_1^*, \ldots, \mathbb{X}_N^*$ can be used to estimate a density, e.g., using kernel density estimation.

A complementary measure is the probability for a given point in the variable space to be non-dominated by the current set of observations, $\mathbb{P}\left[\mathbf{x} \preceq \mathbf{X}_n\right]$. This probability can be expressed in closed form (Keane 2006), so that it can be computed on a grid for instance to display the dominated and non-dominated regions in the variable space. The amount of intermediate probability values (not zero or one) quantifies the uncertainty on the Pareto set (Picheny 2015).

Note that both approaches require extensive sampling over the design space, which makes them computationally intensive.

3. Package overview and options

3.1. Architecture

The structure of the package reflects its main orientations: multi-objective optimization and associated quantification of uncertainty. In particular, readers familiar with the **DiceOptim** and **KrigInv** packages will find a very similar set of functions ranging from high-level interfaces to lower level criteria. Additional helper functions are also provided as well as test functions.

3.2. Functions related to the sequential design of experiments

As described in Section 2.1, Gaussian-process based optimization can be separated into four steps. Depending on the characteristics of the problem at hand, several levels of control are available. For the sake of clarity, we start by describing the highest-level functionalities before detailing routines that enable more control on the optimization process or may be integrated in other procedures.

User-friendly wrapper: `easyGParetoptim`

This is a simple interface to multi-objective optimization that perform all steps described in Section 2.1, which does not require much knowledge on the specificities of Gaussian process based optimization. If no additional control parameters are set, all Steps 1–4 are performed according to default values.

The minimal arguments for `easyGParetoptim` are the following, common with many optimization methods in R such as `optim`:

- `fn`, this is the multi-objective function that returns the values of the objectives at a given design;
- `budget`, the maximal number of evaluations of the expensive black-box function `fn`;

- `lower`, `upper`, vectors giving the limits of the domain for optimization.

A design of experiments may be passed using the argument `par` and corresponding values provided with `values`; otherwise a Maximin LHS design is constructed from **DiceDesign**.

Noisy objectives can be handled with the argument `noise.var`, which stands for the noise variance. We assume here that the user has prior knowledge of the variance. The two main options are to provide a vector of size q (constant noise) or a function (same arguments as `fn`) if the noise depends on `x`.

Additional tuning of the inner procedures are available using the `control` list, in particular the criterion (`method`) and the optimization routine of the acquisition function (`inneroptim`). By default, `easyGParetoptim` uses `"SMS"` as criterion, with `"pso"` as inner-optimization routine. Both choices have been made to favor speed while ensuring robustness. They are also the default choice for the following `GParetoptim` routine.

`GParetoptim`

This function handles Steps 3 and 4, hence assuming that users have performed a design of experiments and built surrogate models at their convenience, which they provide in the argument `model`. Besides `fn`, `lower`, `upper` and `noise.var` shared with `easyGParetoptim`, more parameters are directly exposed, such as `crit` for selecting the infill criteria or `cov.reestim` to decide whether or not hyperparameters are updated after adding new observations. More flexibility is given using control parameters, `optim_control` for the optimization of the infill criterion and `crit_control` for parameters of the latter, that are useful for the following `crit_optimizer` function.

`crit_optimizer`

Optimizing the criteria, also known as acquisition functions, is quite complicated due to their multi-modality: see Figure 5 for an illustration. Besides, in general, no derivative expressions are available and there are large plateaus. On top of that, the attraction basin of the global optimum of the infill criterion may have a very small volume in the variable space (see Roustant *et al.* 2012, for the illustration of this problem). Nonetheless, acquisition functions are typically much cheaper to evaluate than the objective functions and intensive optimization can be carried out.

Three solutions to perform this inner optimization are provided in **GPareto**:

1. the user can provide a set of candidate points with `optimcontrol` in `crit_optimizer` and `GParetoptim` (hence reducing the problem to a discrete search);
2. the default optimization routine is `genoud` (Mebane, Sekhon *et al.* 2011), a genetic algorithm;
3. the `psoptim` optimization method (Bendtsen 2012), a particle swarm algorithm is also provided;

and the corresponding tuning parameters may be passed to `optimcontrol`. Passing any other optimization method is also possible, given that it works as the standard `optim` method in R from package **stats**.

Criteria functions

Four criteria are available in **GPareto** 1.1.3:

- `crit_SMS` for the SMS-EGO criterion (Ponweiser *et al.* 2008; Wagner *et al.* 2010); (based on the MATLAB source code of the authors);
- `crit_EHI` for the expected hypervolume improvement criterion (Emmerich *et al.* 2011) (based on the MATLAB source code of the authors for the bi-objective case);
- `crit_EMI` for the expected maximin improvement criterion (Svenson and Santner 2016; Svenson 2011);
- `crit_SUR` for the expected excursion volume reduction criterion (Picheny 2015).

The `crit_SMS` criterion has an analytical expression for any number of objectives while the one for `crit_EHI` has this only for the bi-objective case. There is a semi-analytical[1] formula for `crit_EMI` for two objectives. Note that the formula for `crit_EHI` is coded using **Rcpp** (Eddelbuettel and François 2011; Eddelbuettel 2013), which offers considerable speed-up over an R implementation.

With $m > 2$, computations of `crit_EMI` and `crit_EHI` rely on sample average approximation (SAA) (Shapiro 2003), as proposed e.g., in Svenson (2011). The principle is to take samples from the posterior distribution of $\mathbf{Y}(\mathbf{x})$, i.e., $\mathbf{Y}(\mathbf{x})^{(1)}, \ldots, \mathbf{Y}(\mathbf{x})^{(p)}$, and take the average of the improvement function over these samples: $\mathbb{E}\left(I(\mathbf{Y}(\mathbf{x}))|\mathcal{A}_n\right) \approx \frac{1}{p}\sum_{j=1}^{p} I(\mathbf{Y}^{(j)}(\mathbf{x}))$. Note that a large sample size p is often needed to obtain a good approximation, which is at the cost of computational time. By default, the number of SAA samples `nb.samp` is set to 50.

`crit_SUR` requires integrating some quantities over the design space $\mathbb{X}$, which must be done numerically, making this criterion computationally intensive. Similarly to the **KrigInv** package (Chevalier *et al.* 2014a), several alternatives to select integration points are provided using the function `integration_design_optim`, including uniformly distributed random points, quasi Monte Carlo sequences, as well as importance sampling schemes (as described in Picheny 2015). For now `crit_SUR` is available for two and three objectives.

In terms of complexity, both `crit_EHI` with $m > 2$ and `crit_SMS` use hypervolume computations provided in the **emoa** package (much more frequently for the first one, which is thus slower). Those have an exponential complexity in the number of objectives and also depend on the number of points in the Pareto front. For `crit_EMI` the complexity mainly depends on the number of sample points for the SAA approximation and linearly in the number of objectives, it is more affordable than `crit_EHI` for more than two objectives. For `crit_SUR`, the complexity is essentially related to the integration over the input domain which can become cumbersome with many variables.

Importantly, except for `crit_SUR`, these criteria depend on the relative scaling of the objectives, i.e., multiplying one objective by a constant modifies the results. Scaling may be performed by the user, e.g., from the maximum and minimum values observed for each objective as in Parr (2012) or Svenson (2011). In addition, `crit_EHI` and `crit_SMS` need a reference point for bounding hypervolume computations. If no reference point is given by the user,

[1] numerical quadrature is needed for some 1-dimensional integrals, see Svenson (2011).

with `refPoint`, we set it to $R_i = \max_{\mathbf{y}_j \in \mathcal{P}_n} \left(y_j^{(i)}\right) + \max\left(1, 0.2 \times \left(\max_{\mathbf{y}_j \in \mathcal{P}_n} \left(y_j^{(i)}\right) - \min_{\mathbf{y}_j \in \mathcal{P}_n} \left(y_j^{(i)}\right)\right)\right)$, $1 \leq i \leq m$, extending for non-scaled objectives the method of Ponweiser *et al.* (2008) and references therein. The scaling and additional parameters are some of the drawbacks of multi-objective infill criteria, as discussed in Wagner *et al.* (2010) and Svenson (2011).

A brief comparison of the different criteria is given in Table 1.

Name	Indicator	Analytical	m	Cost	Scaling dependent	Parameters
`crit_EHI`	Hypervolume	$m = 2$ only	Any	+ to +++	Yes	`refPoint`, `nb.samp` ($m > 2$)
`crit_EMI`	Additive-ϵ	No	Any	++	Yes	`nb.samp` ($m > 2$)
`crit_SMS`	Hypervolume	Yes	Any	+	Yes	`refPoint`
`crit_SUR`	Probability of non-domination	No	$m \leq 3$	+++	No	`integration points`

Table 1: Summary of the characteristics of infill criteria available in **GPareto**. The computational costs are given for a bi-objective example. Note that the cost of `crit_EHI` is low in this case but increase exponentially with the output dimension. `SURcontrol` is a list of parameters depending on the integration strategy chosen.

Test functions are provided in **GPareto**, such as problems in the MOP (Van Veldhuizen and Lamont 1999), ZDT (Zitzler, Deb, and Thiele 2000) and DTLZ (Deb, Thiele, Laumanns, and Zitzler 2005) test suites.

3.3. Functions related to uncertainty quantification and post-processing

User-friendly wrapper: `plotGPareto`

Results given by `easyGParetoptim` or `GParetoptim` can be visualized using the `plotGPareto` function. The default output of this function is to display only the points visited during optimization along with optimal points. Depending on the number of objectives, the Pareto front approximation is a simple plot (two objectives), a perspective view of the Pareto front (three) or a representation in parallel coordinates (Inselberg 2009) (more than three).

Then, three different outputs are possible to improve insight on the algorithm results. These can be obtained either by setting some options of `plotGPareto` or directly by calling the corresponding functions:

- an estimation of the Vorob'ev expectation giving the expected location of the Pareto front along with a visualization of the corresponding uncertainty (option `UQ_PF = TRUE` or with `CPF` and `plotSymDevFun`);
- an estimation of the density of Pareto optimal points in the variable space (option `UQ_dens = TRUE` or with `ParetoSetDensity`);
- a visualization of the probability of non-domination in the variable space (option `UQ_PF = TRUE` or with `plot_uncertainty`).

Uncertainty quantification on Pareto front

The entry function is the creator of the `'CPF'` class (for conditional Pareto front), which deals with computing the probability for a target in the objective space to be dominated, also known as the attainment function, Vorob'ev expectation (VE) and Vorob'ev deviation (VD), from a grid discretization. It takes as main arguments:

- `fun1sims`, `fun2sims`, the sets of conditional simulations for both objectives, that can be computed for instance using the `simulate` function of **DiceKriging**;
- `response`, the known objective values.

The empirical attainment function is calculated on a grid in the objective space from the CPF sets given by the conditional simulations. Taking advantage of the regularity of the grid to compute volumes, the Vorob'ev expectation is computed quickly by dichotomy. Then the Vorob'ev deviation is a sum of hypervolume indicator values. The `plot` method applied to `CPF` objects displays the attainment function in gray-scale, and possibly the VE. In addition, the `plotSymDefFun` function can be used to display the spread of conditional simulations of Pareto fronts around the Vorob'ev expectation. See Binois *et al.* (2015a) for details.

Uncertainty quantification on Pareto set

The function `plot_uncertainty`, based on the `print_uncertainty_nd` function of the **KrigInv** package Chevalier *et al.* (2014a), draws contour lines of the probability of non-domination. In dimension larger than two, contour lines are drawn for each couple of two variables representing either the average, maximum or minimum of the probability over the other variables.

The function `ParetoSetDensity` relies from one end on conditional simulations of the objectives given by the `simulate` function of **DiceKriging**, and on the other end on a kernel density estimation of the probability of belonging to the Pareto set. It returns an object of class `'kde'` from the package **ks** (Duong 2016). This object can be displayed in small dimension (which is done by `plotGPareto`), or may be used to sample points.

Search for target designs

Finally, **GPareto** allows the user to search for additional points corresponding to a particular target in the objective space. Given a target point (for instance, a location along the estimated Pareto front based on the Vorob'ev expectation), the function `getDesign` returns the closest design, that is, the design that maximizes the probability of dominating the target in the variable space. This step requires running an optimization algorithm, which can be tuned similarly to `crit_optimizer` using an `optimcontrol` argument.

3.4. Some technical aspects

Fast objectives

Motivated by applications where some objective functions are computable at a negligible cost compared to other objectives, **GPareto** offers an option for MOO in case of co-existing cheap- and expensive-to-evaluate objectives. As an example, in structural mechanics one objective is

typically the mass (which is directly derived from the design variables) and the other depends on the response of the system, hence involving a finite element model. To ensure compatibility with the infill criteria, fast objectives are wrapped in the `'fastfun'` class which mimics the behavior of methods such as `predict` or `update`. Then predicting the value at a new point amounts to evaluating the fast function, which returns the corresponding value with a zero prediction variance, exactly like what happens for already evaluated points. They may be used with the `cheapfn` argument in `easyGParetoptim`, `GParetoptim` and `crit_optimizer`.

Numerical stability

Another computational challenge with kriging, discussed, e.g., in Roustant *et al.* (2012), is the numerical non invertibility of covariance matrices. It usually happens whenever design points are too close. This is especially troublesome in optimization since, when converging, points are likely to be added close to each other[2]. In **GPareto**, preventing this problem is achieved with the `checkPredict` function. Before evaluating the selected criterion, `checkPredict` tests whether the new point $\mathbf{x}$ is too close to existing ones, with a tunable threshold that can be passed as argument. Three options are available to define when designs are considered as "too close":

- minimal Euclidean distance in the input space: $\min_{1\leq i\leq n} d(\mathbf{x}, \mathbf{x}_i)$;
- ratio of the predictive variance $s_n(\mathbf{x})^2$ over the variance parameter for stationary kernels;
- minimal *canonical* distance coupled with k_n: $\min_{1\leq i\leq n} \sqrt{k_n(\mathbf{x},\mathbf{x}) - 2k_n(\mathbf{x},\mathbf{x}_i) + k_n(\mathbf{x}_i,\mathbf{x}_i)}$.

The first two options are also used in **KrigInv** and **DiceOptim** respectively. The first one is less computationally demanding but also less robust.

Moreover, to improve stability of the update of already existing models with new observations, it is possibly attempted twice. First, an update with re-estimation of the hyperparameters is performed. Then, if it has failed, a new update is tested with the old hyperparameters. If this is still insufficient to train the model with all observations, the user may try to remove some points or apply the *jitter* technique consisting in adding a small constant to the diagonal of the covariance matrix to improve its condition number, see e.g., Roustant *et al.* (2012). Replacing two close observations by one observation and its estimated directional derivative as proposed in Osborne (2010) is another appealing solution.

4. Illustrating examples using GPareto

This section shows the different functionalities of **GPareto** on three classical toy examples.

4.1. Two objectives, unidimensional example

We consider the following simple 1-dimensional bi-objective optimization problem from the literature, see e.g., Van Veldhuizen and Lamont (1999), re-scaled to $[0, 1]$, to illustrate the dif-

[2]Repeating the same observations exactly, when there is no noise, is prevented since the criteria EHI, EMI and SUR are equal to zero for an existing design, while it is penalized with SMS.

ferent steps of the procedure and the key concepts of GP-based multi-objective optimization:

$$\text{MOP2}(x) = \begin{cases} f_1 &= 1 - \exp\left((1-x)^2\right) \\ f_2 &= 1 - \exp\left((1+x)^2\right) \end{cases}$$

We first define the initial design of experiments (`design.init`, six points evenly spaced between zero and one) and compute the corresponding set of observations `response.init`, which we use to build two kriging models with **DiceKriging**'s `km` function and put them into a single list (`model`):

```
R> design.init <- matrix(seq(0, 1, length.out = 6), ncol = 1)
R> response.init <- MOP2(design.init)
R> mf1 <- km(~1, design = design.init, response = response.init[, 1])
R> mf2 <- km(~1, design = design.init, response = response.init[, 2])
R> model <- list(mf1, mf2)
```

Then, we call the main function `GParetoptim` to perform seven optimization steps using the EHI criterion. Note that EHI requires a *reference point* as a parameter, which corresponds to an upper bound for each objective (here $[2, 2]$, if not provided, it is estimated at each iteration, see Section 3.2.4). The other mandatory inputs are the GP models `model`, the objective function `fn`, number of steps (`nsteps`) and the design bounds (`lower` and `upper`).

```
R> res <- GParetoptim(model = model, fn = MOP2, crit = "EHI", nsteps = 7,
+    lower = 0, upper = 1, critcontrol = list(refPoint = c(2, 2)))
----------------------------
Starting optimization with :
 The criterion EHI
 The solver genoud
----------------------------
Ite / Crit / New x / New y
1 / 0.0603 / 0.5 / 0.632 0.632
2 / 0.04 / 0.255 / 0.98 0.000325
3 / 0.0361 / 0.748 / 4.57e-05 0.981
4 / 0.0154 / 0.547 / 0.483 0.756
5 / 0.0154 / 0.453 / 0.756 0.483
6 / 0.00903 / 0.651 / 0.144 0.924
7 / 0.00891 / 0.349 / 0.923 0.146
```

By default, `GParetoptim` prints the points chosen and their corresponding evaluations, along with the value of the sampling criterion. The criterion here decreases almost monotonically, since as the exploration progresses the remaining improvement (hypervolume gain obtained by a new observation) decreases. Figure 3 illustrates the results of this 1D problem, and shows the ability of the GP models to accurately learn functions on target regions (the Pareto set) based on a few observations.

We consider now the more complex optimization problem (P1) given in Parr (2012):

$$\text{P1}(x) = \begin{cases} f_1 = \left(x_2 - 5.1(x_1/(2\pi))^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(\left(1 - \frac{1}{8\pi}\right)\cos(x_1) + 1\right) \\ f_2 = -\sqrt{(10.5 - x_1)(x_1 + 5.5)(x_2 + 0.5)} - \frac{(x_2 - 5.1(x_1/(2\pi))^2 - 6)^2}{30} - \frac{(1-\frac{1}{8\pi})\cos(x_1)+1}{3} \end{cases}$$

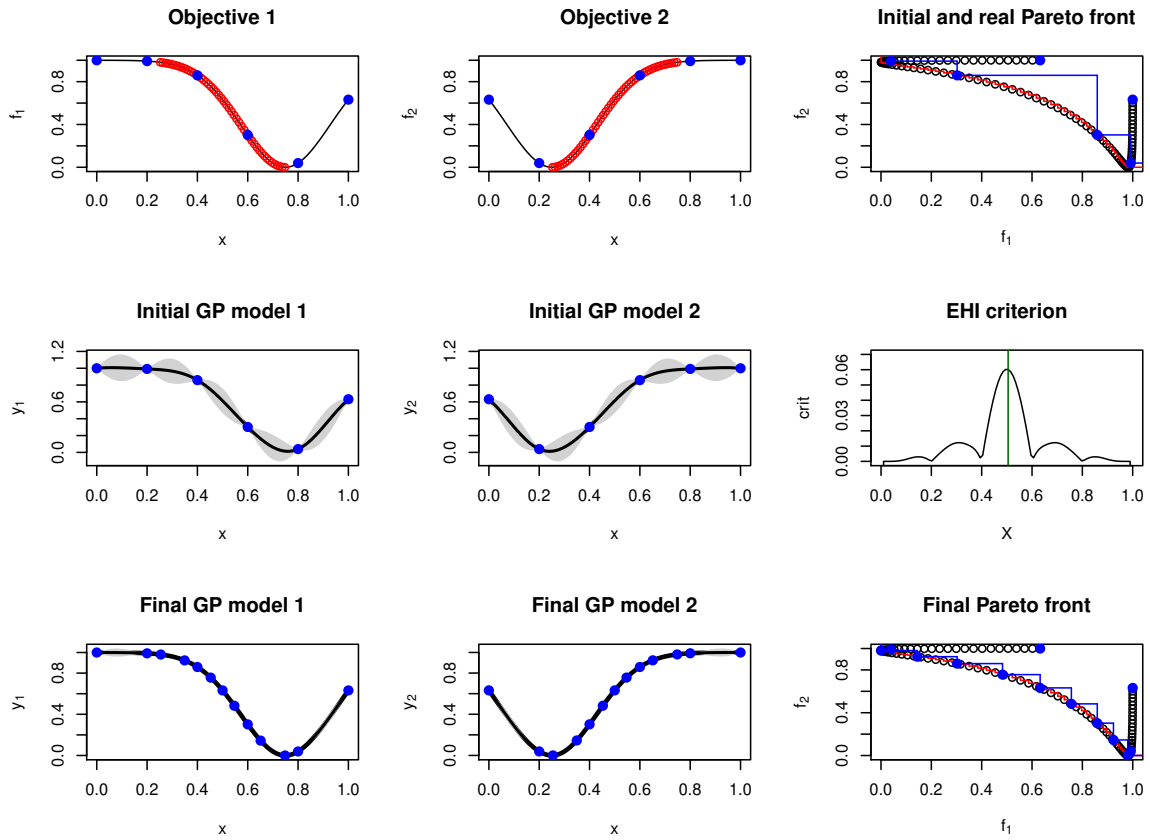

Figure 3: Summary of the optimization procedure on the 1-dimensional example. Top: objective functions are in black, with design points in blue. The red points show the Pareto set. The right figure shows the problem in the objective space (f_1 vs. f_2 for all x). The red line shows all the Pareto-optimal solutions of the problem and the blue line is the current Pareto front based on the six observations. Middle: GP models corresponding to both objectives based on the initial observations and corresponding acquisition criterion (expected hypervolume improvement) that is maximized to select the next observation. Bottom: GP models at the end of the optimization process and Pareto front returned by the method.

with $x_1 \in [-5, 10]$ and $x_2 \in [0, 15]$ (rescaled to $[0, 1]^2$ in **GPareto**). In particular, the first objective is the Branin-Hoo function that introduces multi-modality.

We use this example to show three important features of the package:

- the possibility to access different steps of the EGO strategy,
- the use of `'fastfun'` objects and,
- the post-processing functionalities.

On this analytical example, it is possible to display the true Pareto front and set using the `plotParetoGrid` function:

```
R> plotParetoGrid(P1)
```

The graphical output is shown in Figure 4.

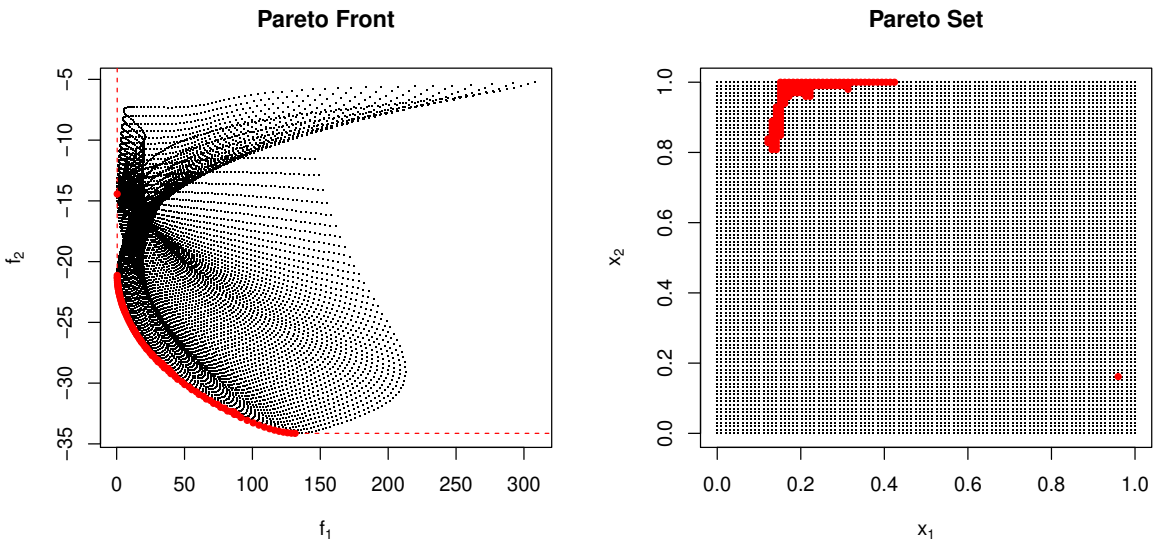

Figure 4: Actual Pareto front and set for (P1).

As in the previous example, we first build an initial set of (seven) observations and a list of two GP models:

```
R> set.seed(1)
R> d <- 2; ninit <- 7; fun <- P1
R> design <- lhsDesign(ninit, d, seed = 42)$design
R> response <- t(apply(design, 1, fun))
R> mf1 <- km(~., design = design, response = response[, 1])
R> mf2 <- km(~., design = design, response = response[, 2])
R> model <- list(mf1, mf2)
```

Now, we call directly the function `crit_optimizer` to choose the next point to evaluate using the SUR criterion. Here, the `optimcontrol` input is used to choose the `genoud` algorithm for the criterion optimization. The `critcontrol` input allows us to choose the integration points for the criterion, here a regular 21×21 grid.

```
R> x.grid <- seq(0, 1, length.out = 21)
R> test.grid <- expand.grid(x.grid, x.grid)
R> SURcontrol <- list(integration.points = test.grid)
R> omEGO1 <- crit_optimizer(crit = "SUR", model = model,  lower = c(0, 0),
+    upper = c(1, 1), critcontrol = list(SURcontrol = SURcontrol),
+    optimcontrol = list(method = "genoud", pop.size = 20,
+    int.seed = 2, unif.seed = 3)
+    )
```

Now, let us assume that f_2 is considerably faster to evaluate than f_1. We split the objective into two separate functions, `fun1` and `fun2`, and we replace the second GP model by a `'fastfun'` object:

```
R> fun1 <- function(x) P1(x)[, 1]
R> fun2 <- function(x) P1(x)[, 2]
R> fastmf2 <- fastfun(fn = fun2, design = design, response = response[, 2])
R> model2 <- list(mf1, fastmf2)
```

The script to search for the next observation is identical.

In Figure 5, we show the initial set of observations and the next point to evaluate according to each setup. For illustration purposes, the contour lines of the criteria are also computed. We see that using the `'fastfun'` object (hence, additional information), the SMS criterion points clearly to a narrower region, which is in addition quite different from the ones given by the other setup. On both cases, the inner optimization loops successfully find the global maxima of the criteria surfaces.

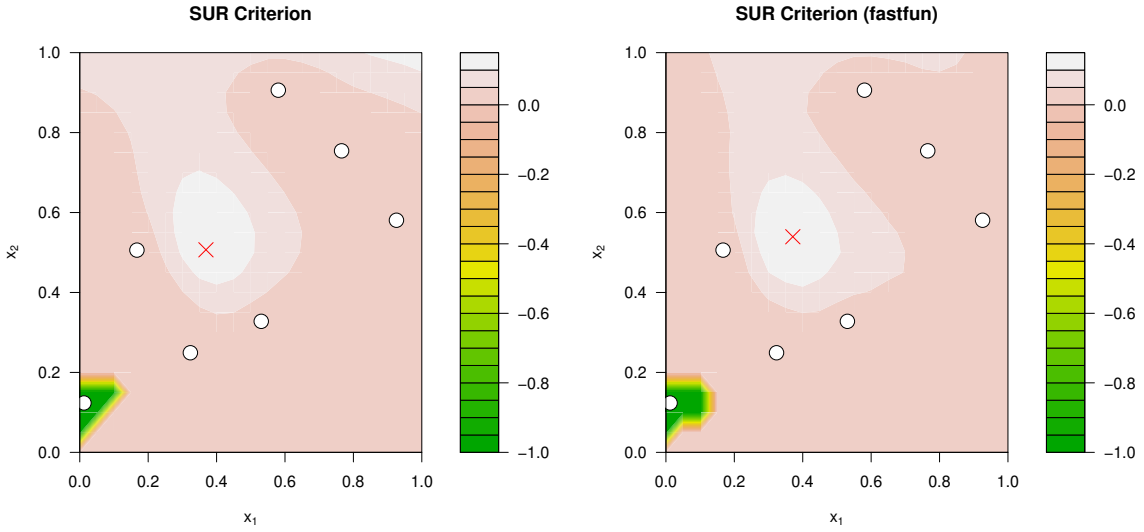

Figure 5: SUR criterion with regular setup (left) and `fastfun` setup (right). The red crosses show the optimal sampling points according to the criteria, found using `genoud` (left) and `pso` (right), respectively.

Now, we apply ten steps of SUR, first with two regular objectives, then with the `fastfun` setting:

```
R> sol <- GParetoptim(model = model, fn = fun, crit = "SUR", nsteps = 7,
+    lower = c(0, 0), upper = c(1, 1), optimcontrol = list(method = "pso"),
+    critcontrol = list(SURcontrol = list(distrib = "SUR", n.points = 50)))
```

```
R> solFast <- GParetoptim(model = list(mf1), fn = fun1, cheapfn = fun2,
+    crit = "SUR", nsteps = 7, lower = c(0, 0), upper = c(1, 1),
+    optimcontrol = list(method = "pso"),
+    critcontrol = list(SURcontrol = list(distrib = "SUR", n.points = 50)))
```

Then, we generate the post-treatment processes using `plotGPareto`. The graphical outputs are given in Figure 6. Optional parameters `f1lim` and `f2lim` are used to fix bounds for the top graphs to allow better comparison.

```
R> lim1 <- seq(-50, 240, length.out = 101)
R> lim2 <- seq(-35, 0, length.out = 101)
R> plotGPareto(sol, UQ_PF = TRUE, UQ_PS = TRUE, UQ_dens = TRUE,
+    control = list(f1lim = lim1, f2lim = lim2))
Vorob'ev deviation:  616.8202


R> plotGPareto(solFast, UQ_PF = TRUE, UQ_PS = TRUE, UQ_dens = TRUE,
+    control = list(f1lim = lim1, f2lim = lim2))
Vorob'ev deviation:  197.4616
```

First, we see the interest of using the '`fastfun`' class when some objectives are cheap to compute: The Pareto front obtained this way is much more accurate (Figure 6, top), in particular for low values of the second objective.

Interestingly, the two Vorob'ev expectations are similar, and provide a very good prediction of the actual Pareto front (Figure 4), except for the lowest values of the first objective. However, the Vorob'ev deviations (gray areas) show a higher local uncertainty for this part of the front. Overall the Vorob'ev deviation values (346 and 212, respectively) indicate a substantially better confidence on the predicted Pareto front using `fastfun`.

The probability and density plots (Figure 6, second and third rows, respectively) provide complementary information on the Pareto set (input space). The probability plots indicate interesting (white) and uninteresting (black) regions, as well as uncertain ones (gray), but do not provide a clear insight on the Pareto set. Here, on both cases, the large gray areas show that additional observations may be beneficial, which is consistent with the large difference between the current Pareto front and the Vorob'ev expectation (Figure 6, top). On the other hand, the densities provide a rather accurate estimates of the Pareto set, in particular for the fastfun setup.

Finally, one may want to extract points from the Vorob'ev expectation of the Pareto front (that is, the input realizing a particular trade-off) that have not been observed yet. To this end, the `getDesign` function returns the most probable design given a target in the objective space, and can be called as follow:

```
R> newPoint <- getDesign(model = sol$lastmodel, target = c(42, -26),
+    lower = c(0, 0), upper = c(1, 1), optimcontrol = list(method = "pso"))
R> newPoint
$par
```

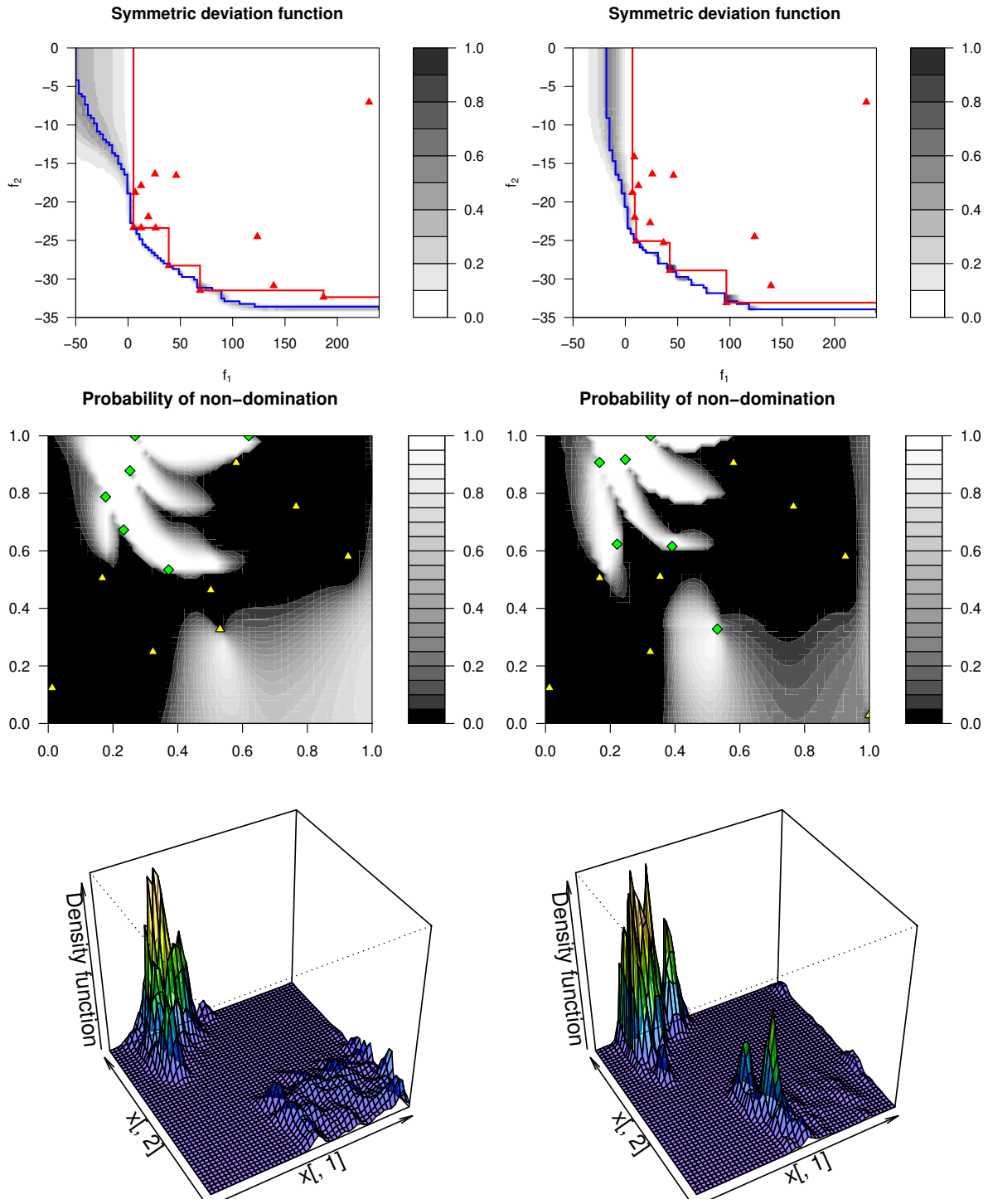

Figure 6: Graphical outputs of optimization runs in terms of Pareto front (top), probability of non-domination (middle) and density of Pareto optimal points (bottom) when both objectives are expensive (left column) or when the second is cheap, using the `cheapfn` argument (middle).

```
            X1        X2
[1,] 0.2552864 0.8176908

$value
     [,1]
[1,]    1

$mean
     init
 31.58098 -27.11136

$sd
      init
0.94154318 0.03586287
```

Here, we have chosen a target $[42, -26]$ that is on the Vorob'ev expectation, where the uncertainty is small but where no observation is near (Figure 6, top left). The `getDesign` output is a list with the value of the design (`par`), the value of the criterion, i.e., the probability that the `newPoint` objective is not dominated by the target) (`value`, here 60%) and the GP prediction of each objective with the associated uncertainty ((`mean`), (`sd`) and confidence intervals). Here, the value of the second objective reaches the target with large confidence, but the first objective value is quite uncertain.

4.2. Four variables, three objectives

Here we consider the DTLZ2 optimization problem (Deb *et al.* 2005) with four variables and three objectives:

$$\text{DTLZ2}(x) = \begin{cases} f_1 &= (1+g(\mathbf{x}))\cos(x_1\frac{\pi}{2})\cos(x_2\frac{\pi}{2}) \\ f_2 &= (1+g(\mathbf{x}))\cos(x_1\frac{\pi}{2})\sin(x_2\frac{\pi}{2}) \\ f_3 &= (1+g(\mathbf{x}))\sin(x_1\frac{\pi}{2}) \end{cases} \text{ with } g(\mathbf{x})) = \sum_{i=3}^{4}\left(x_i - \frac{1}{2}\right)^2$$

whose Pareto front is concave.

This time we simply use `easyGParetoptim` to solve the problem without having to train or prepare models.

```
R> res <- easyGParetoptim(fn = DTLZ2, budget = 50, lower = rep(0, 4),
+    upper = rep(1, 4))
```

Then, we visualize the output using `plotGPareto`. Note that with dimensions larger than two and more than two objectives, only the Pareto front visualization and the probability plots are available. For the latter, we changed the grid size parameter (`resolution`) and the number of integration points (`nintegpoints`) to avoid overly costly figures.

Then, we visualize the output using `plotGPareto`. Note that with dimensions larger than two and more than two objectives, only the Pareto front visualization and the probability plots are available. For the latter, we changed the grid size parameter(`resolution`) and the number of integration points (`nintegpoints`) to avoid overly costly figures.

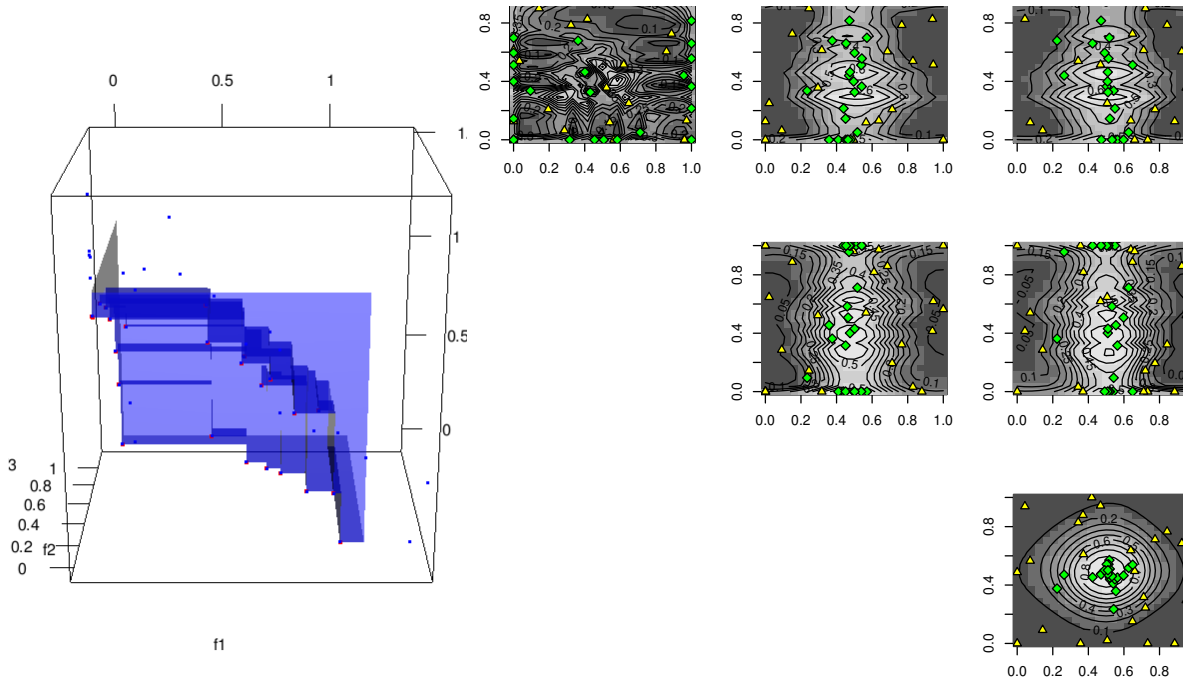

Figure 7: Perspective view of the Pareto front (left) and uncertainty in the variable space (right) for example 3.

```
R> library("rgl", quietly = TRUE)
R> r3dDefaults$windowRect <- c(0,50, 800, 800) # for better looking figure
R> plotGPareto(res, UQ_PS = TRUE, control = list(lower = rep(0, 4),
+    upper = rep(1, 4), nintegpoints = 100, option = "mean",
+    resolution = 25))
```

The graphical outputs are shown in Figure 7. From the definition of DTLZ2, the optimal value for both x_3 and x_4 is $1/2$. This is clearly visible on the probability of non-domination graphs: The (x_3, x_4) surface (bottom right) is unimodal with its maximum at $(0.5, 0.5)$, the other graphs show a ridge at 0.5 for one of the variables. From this representation, optimal sets for x_1 and x_2 are more difficult to observe.

Acknowledgements

Part of this work has been conducted within the frame of the ReDice Consortium, gathering industrial (CEA, EDF, IFPEN, IRSN, Renault) and academic (Ecole des Mines de Saint-Etienne, INRIA, and the University of Bern) partners around advanced methods for Computer Experiments. The authors would like to thank Yves Deville (Alpestat), David Ginsbourger (University of Bern) and Olivier Roustant (Mines Saint-Étienne) for their feedback and suggestions of improvements on the package.

References

Álvarez MA, Rosasco L, Lawrence ND (2011). "Kernels for Vector-Valued Functions: A Review." *Foundations and Trends in Machine Learning*, **4**(3), 195–266.

Bartz-Beielstein T, Zaefferer M (2012). "A Gentle Introduction to Sequential Parameter Optimization." *Technical Report 2*, Bibliothek der Fachhochschule Koeln. URL http://opus.bsz-bw.de/fhk/volltexte/2012/19.

Bendtsen C (2012). *pso: Particle Swarm Optimization*. R package version 1.0.3, URL http://CRAN.R-project.org/package=pso.

Bhardwaj P, Dasgupta B, Deb K (2014). "Modelling the Pareto-optimal Set Using B-spline Basis Functions for Continuous Multi-Objective Optimization Problems." *Engineering Optimization*, **46**(7), 912–938.

Binois M, Ginsbourger D, Roustant O (2015a). "Quantifying Uncertainty on Pareto Fronts With Gaussian Process Conditional Simulations." *European Journal of Operational Research*, **243**(2), 386–394. URL http://www.sciencedirect.com/science/article/pii/S0377221714005980.

Binois M, Ginsbourger D, Roustant O (2015b). "A Warped Kernel Improving Robustness in Bayesian Optimization Via Random Embeddings." In C Dhaenens, L Jourdan, ME Marmion (eds.), *Learning and Intelligent Optimization*, volume 8994 of *Lecture Notes in Computer Science*, pp. 281–286. Springer International Publishing. ISBN 978-3-319-19083-9.

Calandra R, Peters J, Deisenroth M (2014). "Pareto Front Modeling for Sensitivity Analysis in Multi-Objective Bayesian Optimization." In *NIPS Workshop on Bayesian Optimization 2014*.

Carnell R (2016). *lhs: Latin Hypercube Samples*. R package version 0.13, URL https://CRAN.R-project.org/package=lhs.

Chevalier C, Picheny V, Ginsbourger D (2014a). "KrigInv: An Efficient and User-Friendly Implementation of Batch-Sequential Inversion Strategies Based on Kriging." *Computational Statistics & Data Analysis*, **71**, 1021–1034.

Chevalier C, Picheny V, Ginsbourger D (2014b). *KrigInv: Kriging-based Inversion for Deterministic and Noisy Computer Experiments*. R package version 1.3.1, URL https://CRAN.R-project.org/package=KrigInv.

Collette Y, Siarry P (2003). *Multiobjective Optimization: Principles and Case Studies*. Springer.

Deb K, Pratap A, Agarwal S, Meyarivan T (2002). "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II." *Evolutionary Computation, IEEE Transactions on*, **6**(2), 182–197. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=996017.

Deb K, Thiele L, Laumanns M, Zitzler E (2005). "Scalable Test Problems for Evolutionary Multiobjective Optimization." In A Abraham, L Jain, R Goldberg (eds.), *Evolutionary Multiobjective Optimization*, Advanced Information and Knowledge Processing, pp. 105–145. Springer London. ISBN 978-1-85233-787-2.

Deville Y, Ginsbourger D, Roustant O (2015). *kergp: Gaussian Process Laboratory*. R package version 0.2.0, URL http://CRAN.R-project.org/package=kergp.

Dixon L, Szegö G (1978). *Towards Global Optimisation 2*, volume 2. North Holland.

Duong T (2016). *ks: Kernel Smoothing*. R package version 1.10.1, URL https://CRAN.R-project.org/package=ks.

Dupuy D, Helbert C, Franco J (2015). "DiceDesign and DiceEval: Two R Packages for Design and Analysis of Computer Experiments." *Journal of Statistical Software*, **65**(11), 1–38. URL http://www.jstatsoft.org/v65/i11/.

Eddelbuettel D (2013). *Seamless R and C++ Integration with Rcpp*. Springer, New York. ISBN 978-1-4614-6867-7.

Eddelbuettel D, François R (2011). "Rcpp: Seamless R and C++ Integration." *Journal of Statistical Software*, **40**(8), 1–18. URL http://www.jstatsoft.org/v40/i08/.

Emmerich MT, Deutz AH, Klinkenberg JW (2011). "Hypervolume-Based Expected Improvement: Monotonicity Properties and Exact Computation." In *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pp. 2147–2154. IEEE.

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

Gramacy R (2007). "tgp: An R Package for Bayesian Nonstationary, Semiparametric Nonlinear Regression and Design by Treed Gaussian Process Models." *Journal of Statistical Software*, **19**(1), 1–46. URL https://www.jstatsoft.org/index.php/jss/article/view/v019i09.

Gramacy R, Taddy M (2010). "Categorical Inputs, Sensitivity Analysis, Optimization and Importance Tempering with tgp Version 2, an R Package for Treed Gaussian Process Models." *Journal of Statistical Software*, **33**(1), 1–48. URL https://www.jstatsoft.org/index.php/jss/article/view/v033i06.

Henkenjohann N, Kunert J (2007). "An Efficient Sequential Optimization Approach Based on the Multivariate Expected Improvement Criterion." *Quality Engineering*. URL http://www.tandfonline.com/doi/abs/10.1080/08982110701621312.

Inselberg A (2009). *Parallel Coordinates*. Springer.

Jones DR (2001). "A Taxonomy of Global Optimization Methods Based on Response Surfaces." *Journal of global optimization*, **21**(4), 345–383.

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

Keane AJ (2006). "Statistical Improvement Criteria for Use in Multiobjective Design Optimization." *AIAA journal*, **44**(4), 879–891.

Kleijnen JP, Mehdad E (2014). "Multivariate Versus Univariate Kriging Metamodels for Multi-Response Simulation Models." *European Journal of Operational Research*, **236**(2), 573–582.

Knowles J (2006). "ParEGO: A Hybrid Algorithm with On-Line Landscape Approximation for Expensive Multiobjective Optimization Problems." *IEEE Transactions on Evolutionary Computation*, **10**(1), 50–66.

Kobilinsky A, Bouvier A, Monod H (2015). *PLANOR: an R package for the automatic generation of regular fractional factorial designs*. INRA, MIA, Jouy en Josas, France. R package version 0.2-4.

MacDonald B, Ranjan P, Chipman H (2015). "GPfit: An R Package for Fitting a Gaussian Process Model to Deterministic Simulator Outputs." *Journal of Statistical Software*, **64**(1), 1–23.

Mebane WRJ, Sekhon JS, *et al.* (2011). "Genetic Optimization Using Derivatives: The rgenoud Package for R." *Journal of Statistical Software*, **42**(11), 1–26.

Mersmann O (2012). *emoa: Evolutionary Multiobjective Optimization Algorithms*. R package version 0.5-0, URL https://CRAN.R-project.org/package=emoa.

Mersmann O (2014). *mco: Multiple Criteria Optimization Algorithms and Related Functions*. R package version 1.0-15.1, URL https://CRAN.R-project.org/package=mco.

Molchanov IS (2005). *Theory of Random Sets*. Springer.

Naval P (2013). *mopsocd: MOPSOCD: Multi-objective Particle Swarm Optimization with Crowding Distance*. R package version 0.5.1, URL https://CRAN.R-project.org/package=mopsocd.

Novomestky F (2008). *goalprog: Weighted and Lexicographical Goal Programming and Optimization*. R package version 1.0-2.

Osborne M (2010). *Bayesian Gaussian Processes for Sequential Prediction, Optimisation and Quadrature*. Ph.D. thesis, Oxford University New College.

Parr JM (2012). *Improvement Criteria for Constraint Handling and Multiobjective Optimization*. Ph.D. thesis, University of Southampton.

Picheny V (2015). "Multiobjective Optimization using Gaussian Process Emulators via Stepwise Uncertainty Reduction." *Statistics and Computing*, **25**(6), 1265–1280.

Ponweiser W, Wagner T, Biermann D, Vincze M (2008). "Multiobjective Optimization on a Limited Budget of Evaluations Using Model-Assisted S-Metric Selection." In G Rudolph, *et al.* (eds.), *PPSN X*, volume 5199 of *LNCS*, pp. 784–794. Springer.

R Core Team (2019). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

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

Santana-Quintero L, Montano A, Coello C (2010). "A Review of Techniques for Handling Expensive Functions in Evolutionary Multi-Objective Optimization." *Computational Intelligence in Expensive Optimization Problems*, pp. 29–59. URL http://www.springerlink.com/index/v411231h61378w17.pdf.

Shapiro A (2003). "Monte Carlo Sampling Methods." *Handbooks in operations research and management science*, **10**, 353–425.

Svenson J, Santner T (2016). "Multiobjective Optimization of Expensive-to-Evaluate Deterministic Computer Simulator Models." *Computational Statistics & Data Analysis*, **94**, 250–264. URL http://www.sciencedirect.com/science/article/pii/S0167947315001991.

Svenson JD (2011). *Computer Experiments: Multiobjective Optimization and Sensitivity Analysis*. Ph.D. thesis, The Ohio State University.

Tabatabaei M, Hakanen J, Hartikainen M, Miettinen K, Sindhya K (2015). "A Survey on Handling Computationally Expensive Multiobjective Optimization Problems using Surrogates: Non-Nature Inspired Methods." *Structural and Multidisciplinary Optimization*, **52**(1), 1–25.

Theussl S, Borchers H (2015). "CRAN Task View: Optimization and Mathematical Programming." *Technical report*, Version 2015-08-30, URL http://CRAN. R-project. org/view= Optimization.

Tsou CS (2013). *nsga2R: Elitist Non-dominated Sorting Genetic Algorithm based on R*. R package version 1.0, URL https://CRAN.R-project.org/package=nsga2R.

Van Veldhuizen DA, Lamont GB (1999). "Multiobjective Evolutionary Algorithm Test Suites." In *Proceedings of the 1999 ACM symposium on Applied computing*, pp. 351–357. ACM.

Varadhan R (2014). "Numerical Optimization in R: Beyond optim." *Journal of Statistical Software*, **60**(1).

Wagner T, Emmerich M, Deutz A, Ponweiser W (2010). "On Expected-Improvement Criteria for Model-Based Multi-Objective Optimization." *Parallel Problem Solving from Nature–PPSN XI*, pp. 718–727. URL http://www.springerlink.com/index/Y23K4442J3H73444.pdf.

Wang G, Shan S (2007). "Review of Metamodeling Techniques in Support of Engineering Design Optimization." *Journal of Mechanical Design*, **129**(4), 370.

Wang Z, Zoghi M, Hutter F, Matheson D, de Freitas N (2013). "Bayesian Optimization in High Dimensions via Random Embeddings." *In IJCAI*.

Zhang Q, Liu W, Tsang E, Virginas B (2010). "Expensive Multiobjective Optimization by MOEA/D With Gaussian Process Model." *Evolutionary Computation, IEEE Transactions on*, **14**(3), 456–474.

Zitzler E, Deb K, Thiele L (2000). "Comparison of Multiobjective Evolutionary Algorithms: Empirical Results." *Evolutionary computation*, **8**(2), 173–195.

Zuluaga M, Sergent G, Krause A, Püschel M (2013). "Active Learning for Multi-Objective Optimization." In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pp. 462–470.

Affiliation:

Mickaël Binois
Mines Saint-Étienne
EMSE-FAYOL, CNRS UMR 6158, LIMOS
F-42023 Saint-Étienne, France
E-mail: mickael.binois@mines-stetienne.fr

Victor Picheny
MIAT, Université de Toulouse, INRA
F-31326 Castanet-Tolosan, France
E-mail: victor.picheny@inra.fr