Simulation conditionally to a subvariety and application to Bayesian optimization: a dichotomous approach

Frédéric Dambreville

Simulation conditionally to a subvariety and application to Bayesian optimization: a dichotomous approach

Frédéric Dambreville[1 [0000−0002−1460−4126]]

ONERA/DTIS, Université Paris Saclay, F-91123 Palaiseau Cedex, France
frederic.dambreville@onera.fr

Abstract. This chapter concerns the issue of simulating a random vector conditionally to a subvariety. In practice, the subvariety is approximated by a thin shell surrounding it. These simulations are unusual cases of rare event simulation, which are made even more difficult with the dimension, the thinness of the shell and the nonlinearity of the subvariety. We propose a generic approach dedicated to the simulation of this type of rare event, which is dichotomous and inspired by interval analysis, with a particular effort to reduce the dimension curse. Non-incremental and incremental versions of the algorithm are designed in order to handle possible incremental constraint sets. Details on a multithread implementation are given especially in terms of data structure. These simulation methods are then used to propose a non-linear approach to Bayesian optimization. Examples of simulations and of Bayesian optimization are given in order to illustrate and compare the performances of the methods.

Keywords: Rare Event Simulation, Subvariety, Bayesian Optimization, Interval Analysis.

Notations

- $\mathbb{R}$ is the set of reals and x, y, z are real variables,
- $[x] \triangleq [x^-, x^+]$, $[y]$, $[z]$ are notations for intervals,
- Bold notations $[\boldsymbol{x}] \triangleq \prod_{k=1}^n [x_k] = \prod_{k=1}^n [x_k^-, x_k^+]$ and $[\boldsymbol{x}[\triangleq \prod_{k=1}^n [x_k^-, x_k^+[$ are used for boxes and half-open boxes respectively,
- $[\mathbb{R}] \triangleq \{[x^-, x^+] \ : \ [x^-, x^+] \subset \mathbb{R}\}$ is the set of interval subsets of $\mathbb{R}$. $[\mathbb{R}^n] \triangleq \{\prod_{k=1}^n [x_k] \ : \ \forall k, [x_k] \in [\mathbb{R}]\}$ is the set of box subsets of $\mathbb{R}^n$,
- $\rho([x]) = x^+ - x^-$ and $\rho([\boldsymbol{x}]) = \max\limits_{1 \leq k \leq n} \rho([x_k])$ are the length of interval & box,
- $g : \mathbb{R}^k \to \mathbb{R}^j$ is a (multivariate) real function,
- $g([\boldsymbol{x}]) \triangleq \{g(\boldsymbol{y}) \ : \ \boldsymbol{y} \in [\boldsymbol{x}]\}$. *Set* $g([\boldsymbol{x}])$ *is not necessarily a box*,
- $[g] : [\mathbb{R}^k] \to [\mathbb{R}^j]$ is an interval function. *Set* $[g]([\boldsymbol{x}])$ *is a box.*

1 Introduction

This chapter is the long version of conference paper [3]. As a main topic, it addresses the issue of sampling a random vector defined on an n-dimension box

conditionally to a subvariety of this box. As a secondary topic and a main application of the simulation, the chapter proposes a non-linear algorithmic approach for Bayesian optimisation [10].

Thus, given a random vector $\boldsymbol{X}$ of density $f_{\boldsymbol{X}}$ defined on $\mathbb{R}^n$, given box $[\boldsymbol{b}] \in [\mathbb{R}^n]$, *small* box $[\boldsymbol{\epsilon}] \in [\mathbb{R}^m]$ and map $g : [\boldsymbol{b}] \to \mathbb{R}^m$, our main objective is to sample conditional vector $[\boldsymbol{X} \,|g(\boldsymbol{X}) \in [\boldsymbol{\epsilon}]]$ [1].

When space dimension n and constraint dimension m increase and length $\rho([\boldsymbol{\epsilon}])$ of $[\boldsymbol{\epsilon}]$ is small, then event $[g(\boldsymbol{X}) \in [\boldsymbol{\epsilon}]]$ has very low probability. We are dealing here with a particular case of rare event simulation. Moreover, as $[g(\boldsymbol{X}) \in [\boldsymbol{\epsilon}]]$ approximates a subvariety, it is foreseeable that conditional random vector $[\boldsymbol{X} \,|g(\boldsymbol{X}) \in [\boldsymbol{\epsilon}]]$ is essentially and "continuously" multimodal. Sketch figure 1 illustrates this well: to sample conditionally to the blue curve by a mixture of Gaussians, a large number of Gaussians are needed.

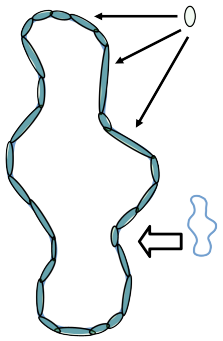


Fig. 1: Gaussian mixture

In survey [11], Morio, Balesdent et al. have evaluated the advantages and drawbacks of various rare event sampling methods. At this point, a comment may be done. In the literature, rare events are generally modelled by a function of risk being above an acceptable threshold. This formalism is quite general, but it suggests that the simulation of a rare event is tightly related to the maximization of a function. In practice, the maximizing set of a function is unimodal or somewhat multimodal. It is uncommon that this maximizing set is a subvariety. Not surprisingly, many methods evaluated in [11] are working well when the rare event is unimodal or moderately multimodal. In the case of conditional vector $[\boldsymbol{X} \,|g(\boldsymbol{X}) \in [\boldsymbol{\epsilon}]]$, these approaches do not work properly or need additional studies to take beneficially into account the subvariety structure.

This chapter promotes an alternative approach based on a dichotomous exploration in order to sample conditionally to a subvariety characterized by a known function. By design, this approach produces independent samples and avoids the phenomenon of sample impoverishement. These properties make it particularly well suited for an accurate approximation of a law conditionally to the subvariety. However, the method must fight the curse of dimensionality and we will be faced with two orthogonal dimensional problems: the exponential increase in sampling exploration and the degeneracy of particle weights. This paper presents an approach as well as generic and weakly parameterized algorithms, allowing good sampling performance for moderate dimensions (up to 11 for now) with balanced management of dimensional issues.

One main applicative purpose of this work is to contribute to the domain of Bayesian optimization from a nonlinear point of view. Bayesian optimization is among the key techniques for black-box optimization. In this kind of problems, one have to optimize fonctions which do not have usable formalization and for

[1] $[\boldsymbol{X} \,|g(\boldsymbol{X}) \in [\boldsymbol{\epsilon}]]$ is used as an abbreviation for $[\boldsymbol{X} \,|\boldsymbol{X} \in [\boldsymbol{b}] \,\&\, g(\boldsymbol{X}) \in [\boldsymbol{\epsilon}]]$.

which the request to an evaluator is generally particularly expensive. Accordingly, an inherent goal of such optimization is to save as much as possible on the number of candidate solutions to be evaluated. Approaches based on Gaussian surrogate models [8] have demonstrated the great interest of these Bayesian methods, but suffer from a drop in efficiency with the increase in dimension. Our contribution aims to address this dimensional difficulty by considering a known non-linear criterion depending on a model noise (a random vector) rather than a surrogate model of the criterion.

The chapter is divided in four main parts. Section 2 introduces concepts of interval analysis and derives some first ideas for a generic sampler. Section 3 deepens the intuitions introduced in section 2 and presents the working generic algorithms. Section 4 applies the sampling method for the design of a nonlinear algorithm for Bayesian optimization. Section 5 presents tests and analyses.

The algorithm already presented in conference paper [3] has slightly evolved and its implementation has been deeply revisited in terms of multithreading and data structure. In comparison to the conference paper, this chapter also describes a new incremental sampling algorithm, includes a new chapter dedicated to Bayesian optimization and presents rebuilt and supplemented tests and results.

2 Toward a generic sampling approach

This section presents some useful concepts and ideas for defining actually working sampling algorithms. Interval analysis is one of these tools. Interval analysis is a performing and accurate tool for dealing with constraints, and it provides a precise control on the approximation errors. Another interesting point is that intervals and boxes combine well with probability distributions.

2.1 Condiderations about interval analysis

It is not our purpose to perform a good introduction on interval analysis [1, 7]. However, we refer to some key concepts, which are inspiring ideas for this work.

functions and operators. As a main ingredient throughout this chapter, it is reasonable to assume that for most common real functions g, it is possible to derive an interval function $[g]$ that satisfies properties:

- $g([\boldsymbol{x}]) \subset [g]([\boldsymbol{x}])$ for any $[\boldsymbol{x}] \in [\mathbb{R}^n]$,[2]
- $[g]([\boldsymbol{x}]) \subset [g]([\boldsymbol{y}])$ when $[\boldsymbol{x}] \subset [\boldsymbol{y}]$,
- If $\rho([\boldsymbol{x}])$ vanishes, then $\rho([g]([\boldsymbol{x}]))$ vanishes:

$$\rho\left(g([\boldsymbol{x}])\right) \xrightarrow[\rho([\boldsymbol{x}])\to 0]{} 0 \tag{1}$$

[2] In particular, it is better if $[g]$ is minimal, that is $[g]([\boldsymbol{x}])$ is the minimal box containing $g([\boldsymbol{x}])$ as subset for all $[\boldsymbol{x}] \in [\mathbb{R}^n]$.

These properties express that $[g]$ implies a bound on the error propagated by g, and this bound has good convergence behavior in regards to the error. Incidentally, we assume from now on that notation $[g]$ refers to an interval function, which verifies these good properties with respect to the function g.

Let us see how to build the interval functions on some common examples:

1. Reference functions, $g \in \{\ln, \exp, \sin, \cos, .^n, \dots\}$, are continuous onto $\mathbb{R}$, so that $g([x]) \in [\mathbb{R}]$ for all $[x] \in [\mathbb{R}]$. Then, it is optimal to set $[g]([x]) = g([x])$ for all $[x] \in [\mathbb{R}]$. As a consequence, the interval functions are easily optimaly implementable for most reference functions. Here are some incomplete examples of definitions:

$$[\ln]([x]) \triangleq [\ln(x^-), \ln(x^+)] \quad \text{and} \quad [x]^n \triangleq \begin{cases} [x_-^n, x_+^n] \text{ for } n > 0, \\ [x_+^n, x_-^n] \text{ for } n < 0 \ \& \ 0 \notin [x], \end{cases} \tag{2}$$

$$[\cos][x] \triangleq \begin{cases} [\cos(x^-), \cos(x^+)] \text{ for } [x] \subset [-\pi, 0] \\ [\cos(x^+), \cos(x^-)] \text{ for } [x] \subset [0, \pi] \\ [\min(\cos(x^-), \cos(x^+)), 1] \quad \text{for} \quad 0 \in [x] \subset [-\pi, \pi] \\ \text{etc.} \end{cases} \tag{3}$$

2. Minimal interval functions for classical operators $+$, $\cdot$, $-$, $/$ are also easily defined. For example:

$$[x] + [y] \triangleq [x^- + y^-, x^+ + y^+] \quad \text{and} \quad [x] - [y] \triangleq [x^- - y^+, x^+ - y^-]. \tag{4}$$

3. Function g defined by $g(\theta) = (\cos(\theta), \sin(\theta))$ is an example, which is such that $g([\theta]) \notin [\mathbb{R}^2]$. One would rather define $[g]([\theta]) = [\cos]([\theta]) \times [\sin]([\theta])$ ($\times$ is the Carthesian product) which is a strict supset of $g([\theta])$ in general. By the way, this construction is an illustration on how the interval functions of reference are used to define complex interval functions straightforwardly.

There is no uniqueness in the construction of $[g]$, unless it is chosen minimal. Unfortunately, constructing a minimal $[g]$ is not always easy or automatic. Let us consider the case of function $g : \theta \mapsto \cos^2\theta + \sin^2\theta$. Then, there are two obvious definitions for $[g]$:

1. By using the reference functions cos, sin, $.^2$ and $+$ one may derive:

$$[g]([\theta]) = ([\cos]([\theta]))^2 + ([\sin]([\theta]))^2. \tag{5}$$

   For example, we compute $[g]\left(\left[0, \frac{\pi}{2}\right]\right) = [0, 2]$ which is a bad error bound on theoretical value 1. Now, we also compute $[g]\left(\left[-\frac{1}{10}, \frac{1}{10}\right]\right) \simeq [0.99, 1.01]$ which is a tight error bound on 1. This example holds confirmation that $[g]([\theta])$ has a good behavior for small boxes $[\theta]$.
2. By noticing that $\cos^2\theta + \sin^2\theta = 1$, it is optimal to define $[g]([\theta]) = [1, 1]$.

Although approach 2 gives the best solution, in practice approach 1 is prefered since it is generic, it is based on already implemented functions of reference, and it provides a way to construct automatically $[g]$ without any specific knowledge.

Subpaving and set inversion. As $[g]$ implies a bound on errors propagated by g with good convergence behavior, it may be used combined with a dichotomous process to produce a subpaving which efficiently approximates a set inversion $g^{-1}([\boldsymbol{y}])$. The example on figure 2 is kindly given by professor Jaulin and is also taken from [4]. It shows a resulting subpaving which approximates a set inversion. The decomposition is clearly dichotomous.

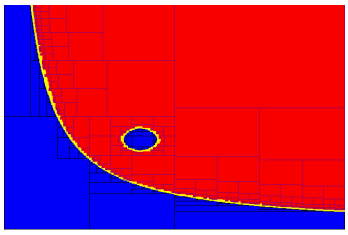


Fig. 2: Subpaving

A bisection process is iterated starting from main box $[\boldsymbol{b}]$; at each iteration, sub-boxes $[\boldsymbol{x}]$ are tested against the constraint $g([\boldsymbol{x}]) \subset [\boldsymbol{y}]$. Three cases arise:

case a: $[g]([\boldsymbol{x}]) \subset [\boldsymbol{y}]$, then $[\boldsymbol{x}]$ is among *red* boxes, which constitute a *subpaving* of $g^{-1}([\boldsymbol{y}])$.

case b: $[g]([\boldsymbol{x}]) \cap [\boldsymbol{y}] = \emptyset$, then $[\boldsymbol{x}]$ is among *blue* boxes, which constitute a *subpaving* of $\mathbb{R}^n \setminus g^{-1}([\boldsymbol{y}])$.

case c: Otherwise, bisection has to be repeated on $[\boldsymbol{x}]$ until sufficient convergence (*yellow* color).

Property (1) plays a key role in the decomposition process, ensuring that case **a** or case **b** are finally achieved when sub-boxes $[\boldsymbol{x}]$ are sufficiently small and sufficiently far from the frontier of set $g^{-1}([\boldsymbol{y}])$.

2.2 Naive dichotomous approach for sampling

Let μ be Borel measure on $\mathbb{R}^n$. It is given from now on:

- a random vector $\boldsymbol{X}$ on $\mathbb{R}^n$ characterized by a *bounded* density $f_{\boldsymbol{X}}$. Cumulative distribution function $F_{\boldsymbol{X}}$ of $\boldsymbol{X}$, defined by $F_{\boldsymbol{X}}(\boldsymbol{x}) = P(\boldsymbol{X} \leq \boldsymbol{x})$ for all $\boldsymbol{x} \in \mathbb{R}^n$, is assumed to be easily computable,
- a box $[\boldsymbol{b}] \in [\mathbb{R}^n]$ and a small box $[\boldsymbol{\epsilon}] \in [\mathbb{R}^m]$,
- a continuous map $g : [\boldsymbol{b}] \to \mathbb{R}^m$ built of functions and operators of reference,
- $[g]$ derived from g and related interval functions and operators of reference.

Sampling within boxes. We point out that it is easy to compute $P(\boldsymbol{X} \in [\boldsymbol{y}])$ or to sample $[\boldsymbol{X} \,|\boldsymbol{X} \in [\boldsymbol{y}]]$ when $F_{\boldsymbol{X}}$ is available, especially when the components of $\boldsymbol{X}$ are jointly independent. These results are well known, and details are given in [4]. Thus, these features are taken for granted in this paper.

Sampling by means of a subpaving. Assume that set $g^{-1}([\boldsymbol{\epsilon}])$ has been approximated (by excess) by a subpaving. Thus, there is $\mathbb{P} \subset [R^n]$ such that:

- For all $[\boldsymbol{x}], [\boldsymbol{y}] \in \mathbb{P}$, boxes $[\boldsymbol{x}[$ and $[\boldsymbol{y}[$ are disjoint,
- $g^{-1}([\boldsymbol{\epsilon}]) \subseteq \bigcup_{[\boldsymbol{x}] \in \mathbb{P}} [\boldsymbol{x}]$ and $\sqcup_{\mathbb{P}} \simeq g^{-1}([\boldsymbol{\epsilon}])$, where $\sqcup_{\mathbb{P}} \triangleq \bigsqcup_{[\boldsymbol{x}] \in \mathbb{P}} [\boldsymbol{x}[$.

Set $\mathbb{P}$ is typically composed from boxes of case **a** and case **c** (red and yellow colors) after a dichotomous subpaving construction of the set inversion.

The quality of the approximation may be quantified by set measures:

$$\alpha_{\mathbb{P}} = \mu\Big(\sqcup_{\mathbb{P}} \setminus g^{-1}([\boldsymbol{\epsilon}])\Big) = \sum_{[\boldsymbol{x}]\in\mathbb{P}} \mu\left([\boldsymbol{x}[\right) - \mu\left(g^{-1}([\boldsymbol{\epsilon}])\right) . \tag{6}$$

Smaller is $\alpha_{\mathbb{P}}$, better is the approximation. Interval based set inversions are able to reach arbitrary precision for small dimensions (2 or 3 typically).

Now, for all $\boldsymbol{y}\in\mathbb{R}^n$, it happens that:

$$\begin{aligned} f_{\boldsymbol{X}|\boldsymbol{X}\in\sqcup_{\mathbb{P}}}(\boldsymbol{y}) &= \frac{f_{\boldsymbol{X}}(\boldsymbol{y})\delta_{\boldsymbol{y}\in\sqcup_{\mathbb{P}}}}{P\left(\boldsymbol{X}\in\sqcup_{\mathbb{P}}\right)} \qquad (7)\\ &= \frac{f_{\boldsymbol{X}}(\boldsymbol{y})\sum\limits_{[\boldsymbol{x}]\in\mathbb{P}}\delta_{\boldsymbol{y}\in[\boldsymbol{x}[}}{\sum\limits_{[\boldsymbol{x}]\in\mathbb{P}} P\left(\boldsymbol{X}\in[\boldsymbol{x}[\right)} = \frac{\sum\limits_{[\boldsymbol{x}]\in\mathbb{P}} P\left(\boldsymbol{X}\in[\boldsymbol{x}[\right) f_{\boldsymbol{X}|\boldsymbol{X}\in[\boldsymbol{x}[}(\boldsymbol{y})}{\sum\limits_{[\boldsymbol{x}]\in\mathbb{P}} P\left(\boldsymbol{X}\in[\boldsymbol{x}[\right)}\\ &= \sum_{[\boldsymbol{x}]\in\mathbb{P}} \frac{P\left(X\in[\boldsymbol{x}]\right)}{\sum\limits_{[\boldsymbol{x}]\in\mathbb{P}} P\left(X\in[\boldsymbol{x}]\right)} f_{\boldsymbol{X}|\boldsymbol{X}\in[\boldsymbol{x}[}(\boldsymbol{y}) , \qquad (8)\end{aligned}$$

where $\delta_{\mathbf{true}} = 1$ and $\delta_{\mathbf{false}} = 0$ else. Since $f_{\boldsymbol{X}}$ is bounded, equation (7) implies:

$$[\boldsymbol{X}\,|\boldsymbol{X}\in\sqcup_{\mathbb{P}}] \xrightarrow[\alpha_{\mathbb{P}}\to 0]{\mathcal{L}} \left[\boldsymbol{X}\,\middle|\,\boldsymbol{X}\in g^{-1}([\boldsymbol{\epsilon}])\right] \tag{9}$$

Now, equation (8) shows clearly that $f_{\boldsymbol{X}|\boldsymbol{X}\in\sqcup_{\mathbb{P}}}$ may be sampled by applying two steps: first sample a box $[\boldsymbol{x}]\in\mathbb{P}$ according to the discrete probability $\frac{P(\boldsymbol{X}\in[\boldsymbol{x}])}{\sum_{[\boldsymbol{x}]\in\mathbb{P}} P(\boldsymbol{X}\in[\boldsymbol{x}])}$, then sample $\boldsymbol{y}$ by the conditional law $f_{\boldsymbol{X}|\boldsymbol{X}\in[\boldsymbol{x}[}$. At last, we have here an efficient method for sampling $[\boldsymbol{X}\,|g(\boldsymbol{X})\in[\boldsymbol{\epsilon}]]$ (algorithm 1).

The approach is efficient on conditional events like $[\boldsymbol{X}\,|g(\boldsymbol{X})\in[\boldsymbol{\epsilon}]]$. But this application of the interval-based inversion is only applicable to rather small dimensions. Taking inspiration of this preliminary approach, we address now the sampling problem in higher dimensions.

Function *Sampling* $[\boldsymbol{X}\,|g(\boldsymbol{X})\in[\boldsymbol{\epsilon}]]$
  input : α,g,N **output:** $\boldsymbol{y}_{1:N}$
  Build subpaving $\mathbb{P}$ such that $\alpha_{\mathbb{P}} < \alpha$
  for $k\leftarrow 1$ **to** N **do**
    Select $[\boldsymbol{x}]\in\mathbb{P}$ with proba. $\frac{P(\boldsymbol{X}\in[\boldsymbol{x}])}{\sum_{[\boldsymbol{x}]\in\mathbb{P}} P(\boldsymbol{X}\in[\boldsymbol{x}])}$
    Build $\boldsymbol{y}_k$ by sampling $[\boldsymbol{X}\,|\boldsymbol{X}\in[\boldsymbol{x}]]$
  end
end

Algorithm 1: Based on a subpaving.

Toward an improved approach. A key point of algorithm 1 is to be able to sample a box $[\boldsymbol{x}]$ of a subpaving of $g^{-1}([\boldsymbol{\epsilon}])$. It is noticeable that an entire build of the subpaving is not needed here. Indeed, if we were able to construct a box $[\boldsymbol{x}]$ of the subpaving *on demand*, together with its *relative weight* within the subpaving, then we would be able to build sample $\boldsymbol{y}$.

Therefore, it may be opportune to merge the sampling process with the dichotomous construction of the subpaving itself. Now, a main ingredient of a dichotomous approach is also how the algorithm divides and conquers. In general, bisections are often used in dichotomous processes, as there is a garanty of exponential volume decrease of the search area. Our approach is less constrained, so as to better tune the exploration strategy.

Function *Sampling* $[\boldsymbol{X}\,|g(\boldsymbol{X}) \in [\boldsymbol{\epsilon}]]$
  input : r, g, ω, N **output:** $(\boldsymbol{y}_k, w_k)_{1:N}$
  for $k \leftarrow 1$ **to** N **do**
    $([\boldsymbol{x}_0], \pi_k, j) \leftarrow ([\boldsymbol{b}], 1, 0)$
    while $\rho([\boldsymbol{x}_j]) > r$ **and** $[g]([\boldsymbol{x}_j]) \not\subset [\boldsymbol{\epsilon}]$ **do**
      $([\boldsymbol{l}_{j+1}], [\boldsymbol{r}_{j+1}]) \leftarrow \texttt{Cut}([\boldsymbol{x}_j])$
      $j \leftarrow j + 1$
      $[\boldsymbol{x}_j] \leftarrow \texttt{Bern}(([\boldsymbol{l}_j], \omega_{[\boldsymbol{l}_j]}), ([\boldsymbol{r}_j], \omega_{[\boldsymbol{r}_j]}))$
      $\pi_k \leftarrow \frac{\omega_{[\boldsymbol{x}_j]}}{\omega_{[\boldsymbol{l}_j]} + \omega_{[\boldsymbol{r}_j]}} \pi_k$
    end
    $w_k \leftarrow P\left(\boldsymbol{X} \in [\boldsymbol{x}_j]\right) \;/\; \pi_k$
    Build $\boldsymbol{y}_k$ by sampling $[\boldsymbol{X}\,|\boldsymbol{X} \in [\boldsymbol{x}_j]]$
  end
end

Algorithm 2: Based on a weighting function.

Here, we speak in terms of cuts, which are more general, *being implied that an appropriate management of the box length is made in order to ensure the convergence.* In this paper, a cut is defined as follows:

- A cut of box $[\boldsymbol{x}] \in [\mathbb{R}^n]$ is a pair $([\boldsymbol{l}], [\boldsymbol{r}]) \in [\mathbb{R}^n]^2$ such that:

$$[\boldsymbol{l}[\cap [\boldsymbol{r}[= \emptyset \quad \text{and} \quad [\boldsymbol{l}[\sqcup [\boldsymbol{r}[= [\boldsymbol{x}[\,, \tag{10}$$

- A bisection is a cut $([\boldsymbol{l}], [\boldsymbol{r}])$ such that $[\boldsymbol{l}]$ and $[\boldsymbol{r}]$ are same-sized.

In order to drive the dichotomous sampling process, we assume that a predictive weighting function is available:

$$\begin{cases} \omega_{[\boldsymbol{x}]} = 0 \quad \text{if} \quad [g]([\boldsymbol{x}]) \cap [\boldsymbol{\epsilon}] = \emptyset \\ \omega_{[\boldsymbol{x}]} \simeq P\left(\boldsymbol{X} \in [\boldsymbol{x}] \,\&\, g(\boldsymbol{X}) \in [\boldsymbol{\epsilon}]\right) \quad \text{otherwise.} \end{cases} \tag{11}$$

Algorithm 2 implicitly builds a partial subpaving, and produces at the same time a weighted particle cloud as a result of the sampling of $[\boldsymbol{X}\,|g(\boldsymbol{X}) \in [\boldsymbol{\epsilon}]]$. The algorithm iterates (**for** loop) the same sampling process, that is the following successive steps, until $[\boldsymbol{x}_j]$ is sufficiently small (*i.e.* $\rho([\boldsymbol{x}_j]) \leq r$) or is inside an implied suppaving (*i.e.* $[g]([\boldsymbol{x}_j]) \subset [\boldsymbol{\epsilon}]$)[3]:

- Build a cut of $[\boldsymbol{x}_j]$ by means of function $\texttt{Cut}([\boldsymbol{x}_j])$. This function is designed so as to ensure that $\rho([\boldsymbol{x}_j])$ vanishes,
- Select randomly one box of cut $([\boldsymbol{l}_j], [\boldsymbol{r}_j])$ in proportion to their weight, by mean of Bernoulli process $\texttt{Bern}(([\boldsymbol{l}_j], \omega_{[\boldsymbol{l}_j]}), ([\boldsymbol{r}_j], \omega_{[\boldsymbol{r}_j]}))$,
- Update π_k which computes the processed probability of $[\boldsymbol{x}_j]$ in regards to the Bernoulli sequence.

[3]Recall that $[g]([\boldsymbol{x}_j])$ is easily computable while $g([\boldsymbol{x}_j])$ is not.

Assume that J is the last value reached by parameter j after the **while** loop. Then, the corrected weight $w_k = \frac{1}{\pi_k} P(\boldsymbol{X} \in [\boldsymbol{x}_J])$ is computed for $[\boldsymbol{x}_J]$ and for $\boldsymbol{y}_k$, and $\boldsymbol{y}_k$ is sampled from $[\boldsymbol{x}_J]$.

Notice that $\omega_{[\boldsymbol{x}]} = 0$ when $[g]([\boldsymbol{x}]) \cap [\boldsymbol{\epsilon}] = \emptyset$, so that boxes $[\boldsymbol{x}_J]$ are necessary within a subpaving of $g^{-1}([\boldsymbol{\epsilon}])$ or its border, thanks to the Bernoulli process. Combined with loop constraint $\rho([\boldsymbol{x}_j]) > r$ **and** $[g]([\boldsymbol{x}_j]) \not\subset [\boldsymbol{\epsilon}]$, it follows that a subpaving of $g^{-1}([\boldsymbol{\epsilon}])$ (or its border) is implicitely and partially built during the sampling process.

When $[g]([\boldsymbol{x}_J]) \subset [\boldsymbol{\epsilon}]$, we have $w_k = \frac{P(\boldsymbol{X} \in [\boldsymbol{x}_J])}{\pi_k}$ where π_k evaluates the processed probability for $[\boldsymbol{x}_J]$. As a result, the weighted particles $(\boldsymbol{y}_k, w_k)$ provide an unbiased estimation of $f_{\boldsymbol{X}|g(\boldsymbol{X})\in[\boldsymbol{\epsilon}]}$ in a subpaving of $g^{-1}([\boldsymbol{\epsilon}])$. It is not the same at the border of $g^{-1}([\boldsymbol{\epsilon}])$, but this case is neglected. However, the sampler is not at all efficient when considering its variance.

Assume $\omega_{[\boldsymbol{x}]} = P(\boldsymbol{X} \in [\boldsymbol{x}] \,\&\, g(\boldsymbol{X}) \in [\boldsymbol{\epsilon}])$, a case which works perfectly. In this ideal case, the weight along a **while** loop is computed by:

$$\frac{\omega_{[\boldsymbol{x}_j]}}{\omega_{[\boldsymbol{l}_j]} + \omega_{[\boldsymbol{r}_j]}} = \frac{P(\boldsymbol{X} \in [\boldsymbol{x}_j] \,\&\, g(\boldsymbol{X}) \in [\boldsymbol{\epsilon}])}{P(\boldsymbol{X} \in [\boldsymbol{l}_j] \cup [\boldsymbol{r}_j] \,\&\, g(\boldsymbol{X}) \in [\boldsymbol{\epsilon}])} = \frac{P(\boldsymbol{X} \in [\boldsymbol{x}_j] \,\&\, g(\boldsymbol{X}) \in [\boldsymbol{\epsilon}])}{P(\boldsymbol{X} \in [\boldsymbol{x}_{j-1}] \,\&\, g(\boldsymbol{X}) \in [\boldsymbol{\epsilon}])}\,, \tag{12}$$

and then:

$$\pi_k = \prod_{j=1}^{J} \frac{\omega_{[\boldsymbol{x}_j]}}{\omega_{[\boldsymbol{l}_j]} + \omega_{[\boldsymbol{r}_j]}} = \frac{P(\boldsymbol{X} \in [\boldsymbol{x}_J] \,\&\, g(\boldsymbol{X}) \in [\boldsymbol{\epsilon}])}{P(\boldsymbol{X} \in [\boldsymbol{b}] \,\&\, g(\boldsymbol{X}) \in [\boldsymbol{\epsilon}])}\,. \tag{13}$$

Three cases potentially arise:

- $[g]([\boldsymbol{x}_J]) \subset [\boldsymbol{\epsilon}]$, *i.e.* $[\boldsymbol{x}_J]$ is in implied subpaving. Since $g([\boldsymbol{x}_J]) \subset [g]([\boldsymbol{x}_J])$, it comes $P(\boldsymbol{X} \in [\boldsymbol{x}_J] \,\&\, g(\boldsymbol{X}) \in [\boldsymbol{\epsilon}]) = P(\boldsymbol{X} \in [\boldsymbol{x}_J])$. Then:
$$\begin{aligned} w_k &= \frac{P(\boldsymbol{X} \in [\boldsymbol{x}_J])}{\frac{P(\boldsymbol{X} \in [\boldsymbol{x}_J] \,\&\, g(\boldsymbol{X}) \in [\boldsymbol{\epsilon}])}{P(\boldsymbol{X} \in [\boldsymbol{b}] \,\&\, g(\boldsymbol{X}) \in [\boldsymbol{\epsilon}])}} = \frac{P(\boldsymbol{X} \in [\boldsymbol{x}_J])}{\frac{P(\boldsymbol{X} \in [\boldsymbol{x}_J])}{P(\boldsymbol{X} \in [\boldsymbol{b}] \,\&\, g(\boldsymbol{X}) \in [\boldsymbol{\epsilon}])}} = P(\boldsymbol{X} \in [\boldsymbol{b}] \,\&\, g(\boldsymbol{X}) \in [\boldsymbol{\epsilon}]) \\ &= P(g(\boldsymbol{X}) \in [\boldsymbol{\epsilon}]) \end{aligned} \tag{14}$$
- $[g]([\boldsymbol{x}_J]) \cap [\boldsymbol{\epsilon}] \neq \emptyset$ but $[g]([\boldsymbol{x}_J]) \not\subset [\boldsymbol{\epsilon}]$, *i.e.* $[\boldsymbol{x}_J]$ is within the border of the implied subpaving. These cases are negligible for small precision r.
- $[g]([\boldsymbol{x}_J]) \cap [\boldsymbol{\epsilon}] = \emptyset$, *i.e.* $[\boldsymbol{x}_J]$ is outside the implied subpaving and its border. This case is simply impossible from the Bernoulli process.

Equation (14) shows that the sampling process results in a cloud of same-weight particles over the implied subpaving. Border cases are negligible. Here we have a sampler of $[\boldsymbol{X} \,|g(\boldsymbol{X}) \in [\boldsymbol{\epsilon}]]$ with the best variance performance in regards to the number of particles. But hypothesis $\omega_{[\boldsymbol{x}]} = P(\boldsymbol{X} \in [\boldsymbol{x}] \,\&\, g(\boldsymbol{X}) \in [\boldsymbol{\epsilon}])$ is necessary. Of course such exact weighting function is almost never available.

Why Does It Generally Not Work? When $\omega_{[\boldsymbol{x}]} \neq P(\boldsymbol{X} \in [\boldsymbol{x}] \,\&\, g(\boldsymbol{X}) \in [\boldsymbol{\epsilon}])$, the accumulated error will explode with the dimension, which will result in dramatically uneven weights on the particles. The resulting weighted particles cloud is then useless for practical applications.

3 Generic dichotomous approaches for sampling

Algorithms 1 and 2 illustrate the two main dimensional issues, that we have to deal with. These approaches are complementary:

- By building a complete subpaving of $g^{-1}([\boldsymbol{\epsilon}])$, algorithm 1 makes possible a direct sampling of $[\boldsymbol{X}\,|g(\boldsymbol{X}) \in [\boldsymbol{\epsilon}]]$, and incidently an accurate computation of $P\left(\boldsymbol{X} \in [\boldsymbol{x}] \,\&\, g(\boldsymbol{X}) \in [\boldsymbol{\epsilon}]\right)$. However, this construction of a complete subpaving is only possible for small dimensions.
- Algorithm 2 avoids the construction of a complete subpaving. Instead, it builds the boxes of an implied subpaving on demand throughout the sampling iteration. However, the algorithm is inefficient unless the predictive weighting function $\omega_{[\boldsymbol{x}]}$ is a good approximation of $P\left(\boldsymbol{X} \in [\boldsymbol{x}] \,\&\, g(\boldsymbol{X}) \in [\boldsymbol{\epsilon}]\right)$. This condition is not accessible in general.

We propose now an intermediate approach which:

- keeps history of the subpaving construction throughout the sampling process,
- use this history to build an improved estimate of $P\left(\boldsymbol{X} \in [\boldsymbol{x}] \,\&\, g(\boldsymbol{X}) \in [\boldsymbol{\epsilon}]\right)$.

By these tricks, it is expected that the sampling precision will increase with the number of samples. In order to avoid useless exploration, we also truncate the dichotomous process on the basis of some predictive assessment of final weight w_k. Thus, the algorithm tends to favor breadth search instead of depth search at the early stages of the sampling process.

3.1 Some containment of the curse of dimension

From now on, it is assumed that:

$$0 \leq \omega_{[\boldsymbol{x}]} \leq P\left(X \in [\boldsymbol{x}]\right) \ , \tag{15}$$

and that:

$$\omega_{[\boldsymbol{x}]} = \begin{cases} 0 & \text{if} \quad [g]([\boldsymbol{x}]) \cap [\boldsymbol{\epsilon}] = \emptyset \ , \\ P\left(\boldsymbol{X} \in [\boldsymbol{x}]\right) & \text{if} \quad [g]([\boldsymbol{x}]) \subset [\boldsymbol{\epsilon}] \ . \end{cases} \tag{16}$$

Algorithm 3 is an evolution of algorithm 2. In addition, it builds an history of cuts, stored in map `cuts`, and computes dynamically from this history an improved weighting function, stored in map `omg`.

The lines of this algorithm are colored in black, blue or dark blue. Black lines are inherited from algorithm 2. Blue lines are new additions to the previous algorithm. Dark blue lines (4, 19, 23 and 2 partially) correspond to the **for** loop of algorithm 2 rewritten as a **while** loop:

$$k \leftarrow 0 \ \ \textbf{while} \ \ k < N \ \ \textbf{do} \ \ \cdots \ \ k \leftarrow k+1 \ \ \cdots \ \ \textbf{end}$$

Variable `cuts` is a dictionary and is used to register the history of computed cuts. At start, `cuts` is defined empty (line 2). For a given box $[\boldsymbol{x}_j]$, the cut on $[\boldsymbol{x}_j]$ is computed only once, if it is computed, by line 8:

$$\textbf{ifundef} \ \texttt{cuts}([\boldsymbol{x}_j]) \leftarrow \texttt{Cut}([\boldsymbol{x}_j])$$

1 **Function** *Sampling* $[\boldsymbol{X}\,|g(\boldsymbol{X})\in[\boldsymbol{\epsilon}]]$
  input : σ, r, g, ω, N **output:** $(\boldsymbol{y}_k, w_k)_{1:N}$
2 $(\mathsf{cuts}, \mathsf{omg}, k) \leftarrow (\emptyset, \emptyset, 0)$
3 $\mathsf{omg}([\boldsymbol{b}]) \leftarrow \omega_{[\boldsymbol{b}]}$
4 **while** $k < N$ **do**
5 $([\boldsymbol{x}_0], \pi_k, j) \leftarrow ([\boldsymbol{b}], 1, 0)$
6 **while** $\rho([\boldsymbol{x}_j]) > r$ **and** $[g]([\boldsymbol{x}_j]) \not\subset [\boldsymbol{\epsilon}]$ **do**
7 **if** $\left|\log_2\left(\frac{\mathsf{omg}([\boldsymbol{b}])}{\mathsf{omg}([\boldsymbol{x}_j])}\pi_k\right)\right| > \sigma$ **goto** 20
8 **ifundef** $\mathsf{cuts}([\boldsymbol{x}_j])$ $\leftarrow \mathtt{Cut}([\boldsymbol{x}_j])$
9 $([\boldsymbol{l}_{j+1}], [\boldsymbol{r}_{j+1}]) \leftarrow \mathsf{cuts}([\boldsymbol{x}_j])$
10 $j \leftarrow j+1$
11 **ifundef** $\mathsf{omg}([\boldsymbol{r}_j]) \leftarrow \omega_{[\boldsymbol{r}_j]}$
12 **ifundef** $\mathsf{omg}([\boldsymbol{l}_j]) \leftarrow \omega_{[\boldsymbol{l}_j]}$
13 $(\nu_{[\boldsymbol{l}_j]}, \nu_{[\boldsymbol{r}_j]}) \leftarrow (\mathsf{omg}([\boldsymbol{l}_j]), \mathsf{omg}([\boldsymbol{r}_j]))$
14 $[\boldsymbol{x}_j] \leftarrow \mathtt{Bern}(([\boldsymbol{l}_j], \nu_{[\boldsymbol{l}_j]}), ([\boldsymbol{r}_j], \nu_{[\boldsymbol{r}_j]}))$
15 $\pi_k \leftarrow \frac{\nu_{[\boldsymbol{x}_j]}}{\nu_{[\boldsymbol{l}_j]}+\nu_{[\boldsymbol{r}_j]}}\pi_k$
16 **end**
17 $w_k \leftarrow P(\boldsymbol{X}\in[\boldsymbol{x}_j]) \;/\; \pi_k$
18 Build $\boldsymbol{y}_k$ by sampling $[\boldsymbol{X}\,|\boldsymbol{X}\in[\boldsymbol{x}_j]]$
19 $k \leftarrow k+1$
20 **for** $i \leftarrow j$ **to** 1 **do**
21 $\mathsf{omg}([\boldsymbol{x}_{i-1}]) \leftarrow \mathsf{omg}([\boldsymbol{l}_i]) + \mathsf{omg}([\boldsymbol{r}_i])$
22 **end**
23 **end**
24 **end**

Algorithm 3: Based on cuts history.

Keyword **ifundef** tests if $\mathsf{cuts}([\boldsymbol{x}_j])$ is defined ; if still undefined, then $\mathsf{cuts}([\boldsymbol{x}_j])$ is set to $\mathtt{Cut}([\boldsymbol{x}_j])$.

Variable omg is a dictionary which records the predictive weighting function and its possible updates, when needed. At start, omg is only defined for $[\boldsymbol{b}]$ and is set to $\omega_{[\boldsymbol{b}]}$ (lines 2 and 3). Variable $\mathsf{omg}([\boldsymbol{r}_j])$ is set to $\omega_{[\boldsymbol{r}_j]}$, if it has not been initialized yet (line 11). The same is done for variable $\mathsf{omg}([\boldsymbol{l}_j])$ at line 12. When the cuts sequence is done (second **while**), then the weighting function is updated by the **for** loop (lines 20, 21, 22). This ensures the computation of $\mathsf{omg}([\boldsymbol{x}])$ as the sum of the weights $\mathsf{omg}([\boldsymbol{z}])$ of the leaves $[\boldsymbol{z}]$ of the cuts tree rooted on $[\boldsymbol{x}]$. Then, property (16) ensures that $\mathsf{omg}([\boldsymbol{x}])$ gets closer to $P(\boldsymbol{X}\in[\boldsymbol{x}]\,\&\,g(\boldsymbol{X})\in[\boldsymbol{\epsilon}])$ when the cuts tree rooted on $[\boldsymbol{x}]$ gets more refined.

Algorithm 3 is similar to algorithm 2, except that:

- cut $([\boldsymbol{l}_{j+1}], [\boldsymbol{r}_{j+1}])$ is recovered from the history, when it is possible (line 9),
- box selection is done by means of $(\nu_{[\boldsymbol{l}_j]}, \nu_{[\boldsymbol{r}_j]}) \triangleq (\mathsf{omg}([\boldsymbol{l}_j]), \mathsf{omg}([\boldsymbol{r}_j]))$.

There is an interesting property here. Assume that J is the last value reached by j and that $J' < J$ *is such that* $\mathsf{omg}([\boldsymbol{l}_j])$ *and* $\mathsf{omg}([\boldsymbol{r}_j])$ *are already defined for*

all $1 \leq j \leq J'$. Weighting functions are updated in these cases. Then, it comes for all $1 \leq j \leq J'$ that:

$$\mathsf{omg}([\boldsymbol{x}_{j-1}]) = \mathsf{omg}([\boldsymbol{l}_j]) + \mathsf{omg}([\boldsymbol{r}_j])\,.$$

The computation of π_k is then simplified:

$$\begin{aligned}\pi_k &= \prod_{j=1}^{J'} \frac{\nu_{[\boldsymbol{x}_j]}}{\nu_{[\boldsymbol{l}_j]} + \nu_{[\boldsymbol{r}_j]}} \prod_{j=J'+1}^{J} \frac{\nu_{[\boldsymbol{x}_j]}}{\nu_{[\boldsymbol{l}_j]} + \nu_{[\boldsymbol{r}_j]}} = \prod_{j=1}^{J'} \frac{\nu_{[\boldsymbol{x}_j]}}{\nu_{[\boldsymbol{x}_{j-1}]}} \prod_{j=J'+1}^{J} \frac{\nu_{[\boldsymbol{x}_j]}}{\nu_{[\boldsymbol{l}_j]} + \nu_{[\boldsymbol{r}_j]}} \\ &= \frac{\nu_{[\boldsymbol{x}_{J'}]}}{\nu_{[\boldsymbol{b}]}} \prod_{j=J'+1}^{J} \frac{\nu_{[\boldsymbol{x}_j]}}{\nu_{[\boldsymbol{l}_j]} + \nu_{[\boldsymbol{r}_j]}}\,. \end{aligned} \tag{17}$$

Thus, the error on π_k grows exponentially only within the newly explored cuts, that is here from $J'+1$ to J. This is a reason for setting a certain restriction on the depth-oriented aspect of this sampling process. Another good reason is to prevent degenerate particle weights, w_k. Algorithm 3 thus implements some code (line 7) for testing the degeneracy of π_k and eventually restarting the sampling loop (second **while**):

$$\textbf{if} \quad \left|\log_2\Big(\frac{\mathsf{omg}([\boldsymbol{b}])}{\mathsf{omg}([\boldsymbol{x}_j])}\pi_k\Big)\right| > \sigma \quad \textbf{goto} \;\; 20$$

This code tests the logarithmic distance between the weight of $[\boldsymbol{x}_j]$, $\mathsf{omg}([\boldsymbol{x}_j])$, and the weight resulting from the sampling process, $\mathsf{omg}([\boldsymbol{b}])\ \pi_k$. If it is higher than σ, then the loop is stopped by going to line 20. By doing that, the incrementation of k is skipped, so that the sampling loop is restarted for the same indice k. However, the update of variable omg is done, and of course, the history of cuts stays incremented. So, although the sampling loop has been interrupted in this case, the sampling structure has been upgraded. This results in an adaptive process which will balance depth and breadth explorations when running the sampling. Breadth exploration is favored on the first sampling iterations, but the tendency becomes inverted after several samples.

3.2 Incremental algorithm

In section 4, we present an application of the conditional simulation for Bayesian optimization. In this applicative context, we have to successively simulate a random vector conditionally to incremental constraints.

More precisely, let $U \geq 1$ and let $[\boldsymbol{\epsilon}_u] \in [\mathbb{R}^{m_u}]$ and $g_u : [\boldsymbol{b}] \to \mathbb{R}^{m_u}$ be defined for $1 \leq u \leq U$. For $1 \leq v \leq U$ are defined:

$$[\boldsymbol{\epsilon}_{1:v}] = \prod_{u=1}^{v} [\boldsymbol{\epsilon}_u]\,, \tag{18}$$

and $g_{1:v} : [\boldsymbol{b}] \to \prod_{u=1}^{v} \mathbb{R}^{m_u}$ by:

$$g_{1:v}(\boldsymbol{x}) = (g_1(\boldsymbol{x}), \cdots, g_v(\boldsymbol{x})) \quad \text{for all} \quad \boldsymbol{x} \in [\boldsymbol{b}]\,. \tag{19}$$

Function *Sampling* $[\boldsymbol{X}\,|g_{1:v}(\boldsymbol{X})\in[\boldsymbol{\epsilon}_{1:v}]]$
  input : σ, r, g, ω, N **output:** $(\boldsymbol{y}_k, w_k)_{1:N}$
  $(\mathsf{cuts}, \mathsf{omg}, \mathsf{lev}) \leftarrow (\emptyset, \emptyset, \emptyset)$
  for $v \leftarrow 1$ **to** U **do**
    $k \leftarrow 0$
    ifundef $(\mathsf{omg}([\boldsymbol{b}]), \mathsf{lev}([\boldsymbol{b}])) \leftarrow (\omega^{1:v}_{[\boldsymbol{b}]}, v)$
    else
      $(t, \mathsf{lev}([\boldsymbol{b}])) \leftarrow (\mathsf{lev}([\boldsymbol{b}]), v)$
      $\mathsf{omg}([\boldsymbol{b}]) \leftarrow \mathsf{omg}([\boldsymbol{b}]) \prod_{t<u\leq v} \varpi^u_{[\boldsymbol{b}]}$
    while $k < N$ **do**
      $([\boldsymbol{x}_0], \pi_k, j) \leftarrow ([\boldsymbol{b}], 1, 0)$
      while $\rho([\boldsymbol{x}_j]) > r$ **and** $[g_{1:v}]([\boldsymbol{x}_j]) \not\subset [\boldsymbol{\epsilon}_{1:v}]$ **do**
        if $\left|\log_2\left(\frac{\mathsf{omg}([\boldsymbol{b}])}{\mathsf{omg}([\boldsymbol{x}_j])}\pi_k\right)\right| > \sigma$ **goto** 31
        ifundef $\mathsf{cuts}([\boldsymbol{x}_j]) \leftarrow \mathtt{Cut}([\boldsymbol{x}_j], v)$
        $([\boldsymbol{l}_{j+1}], [\boldsymbol{r}_{j+1}]) \leftarrow \mathsf{cuts}([\boldsymbol{x}_j])$
        $j \leftarrow j+1$
        ifundef $(\mathsf{omg}([\boldsymbol{r}_j]), \mathsf{lev}([\boldsymbol{r}_j])) \leftarrow (\omega^{1:v}_{[\boldsymbol{r}_j]}, v)$
        else
          $(t, \mathsf{lev}([\boldsymbol{r}_j])) \leftarrow (\mathsf{lev}([\boldsymbol{r}_j]), v)$
          $\mathsf{omg}([\boldsymbol{r}_j) \leftarrow \mathsf{omg}([\boldsymbol{r}_j]) \prod_{t<u\leq v} \varpi^u_{[\boldsymbol{r}_j]}$
        ifundef $(\mathsf{omg}([\boldsymbol{l}_j]), \mathsf{lev}([\boldsymbol{r}_j])) \leftarrow (\omega^{1:v}_{[\boldsymbol{l}_j]}, v)$
        else
          $(t, \mathsf{lev}([\boldsymbol{l}_j])) \leftarrow (\mathsf{lev}([\boldsymbol{l}_j]), v)$
          $\mathsf{omg}([\boldsymbol{l}_j) \leftarrow \mathsf{omg}([\boldsymbol{l}_j]) \prod_{t<u\leq v} \varpi^u_{[\boldsymbol{l}_j]}$
        $(\nu_{[\boldsymbol{l}_j]}, \nu_{[\boldsymbol{r}_j]}) \leftarrow (\mathsf{omg}([\boldsymbol{l}_j]), \mathsf{omg}([\boldsymbol{r}_j]))$
        $[\boldsymbol{x}_j] \leftarrow \mathtt{Bern}(([\boldsymbol{l}_j], \nu_{[\boldsymbol{l}_j]}), ([\boldsymbol{r}_j], \nu_{[\boldsymbol{r}_j]}))$
        $\pi_k \leftarrow \frac{\nu_{[\boldsymbol{x}_j]}}{\nu_{[\boldsymbol{l}_j]}+\nu_{[\boldsymbol{r}_j]}}\pi_k$
      end
      $w_k \leftarrow P(\boldsymbol{X}\in[\boldsymbol{x}_j]) \;/\; \pi_k$
      Build $\boldsymbol{y}_k$ by sampling $[\boldsymbol{X}\,|\boldsymbol{X}\in[\boldsymbol{x}_j]]$
      $k \leftarrow k+1$
      for $i \leftarrow j$ **to** 1 **do**
        $[\boldsymbol{x}_{i-1}] \Leftarrow \mathtt{Shrink}([\boldsymbol{l}_i], [\boldsymbol{r}_i])$
        $\mathsf{omg}([\boldsymbol{x}_{i-1}]) \leftarrow \mathsf{omg}([\boldsymbol{l}_i]) + \mathsf{omg}([\boldsymbol{r}_i])$
      end
    end
  end
end

Algorithm 4: Incremental sampling.

Our purpose is to sample successively the sequence of conditional vectors:

$$[\boldsymbol{X}\,|g_{1:1}(\boldsymbol{X})\in[\boldsymbol{\epsilon}_{1:1}]]\,,\cdots,[\boldsymbol{X}\,|g_{1:v}(\boldsymbol{X})\in[\boldsymbol{\epsilon}_{1:v}]]\,,\cdots,[\boldsymbol{X}\,|g_{1:U}(\boldsymbol{X})\in[\boldsymbol{\epsilon}_{1:U}]]\,, \tag{20}$$

in an efficient way. When the sampling is done by means of algorithm 3, the sampling structure produced for each conditional vector, $[\boldsymbol{X} \,|g_{1:v}(\boldsymbol{X}) \in [\boldsymbol{\epsilon}_{1:v}]]$ where $v \in [\![1, U]\!]$, is lost from a simulation process to another. This is inefficient, since it is foreseeable that the sampling structure for vector $[\boldsymbol{X} \,|g_{1:v}(\boldsymbol{X}) \in [\boldsymbol{\epsilon}_{1:v}]]$ is certainly informative for sampling vector $[\boldsymbol{X} \,|g_{1:v+1}(\boldsymbol{X}) \in [\boldsymbol{\epsilon}_{1:v+1}]]$.

Algorithm 4 is an update of algorithm 3, with the purpose of incrementally building the sampling structures for the sequence $([\boldsymbol{X} \,|g_{1:v}(\boldsymbol{X}) \in [\boldsymbol{\epsilon}_{1:v}]])_{1 \leq v \leq U}$. Predictive weight is now incremental and takes multiplicative form:

$$\omega_{[\boldsymbol{x}]}^{1:v} = P\left(\boldsymbol{X} \in [\boldsymbol{x}]\right) \prod_{u=1}^{v} \varpi_{[\boldsymbol{x}]}^{u} \,. \tag{21}$$

Prefix $P\left(\boldsymbol{X} \in [\boldsymbol{x}]\right)$ in (21) is not out of place: recall that $\omega_{[\boldsymbol{x}]}^{1:v}$ should approximate $P\left(\boldsymbol{X} \in [\boldsymbol{x}] \,\&\, g_{1:v}(\boldsymbol{X}) \in [\boldsymbol{\epsilon}_{1:v}]\right) = P\left(\boldsymbol{X} \in [\boldsymbol{x}]\right) P\left(g_{1:v}(\boldsymbol{X}) \in [\boldsymbol{\epsilon}_{1:v}] \,|\boldsymbol{X} \in [\boldsymbol{x}]\right)$. The multiplicative form is then a consequence of the incremental constraint.

In order to deal with the incremental construction of the samplers, algorithm 4 introduces new dictionary, lev, complementary to cuts and omg. Constraint level $t = \mathsf{lev}([\boldsymbol{x}])$ indicates that weight $\mathsf{omg}([\boldsymbol{x}])$ has been actually computed with constraints sequence $g_{1:t}(\boldsymbol{X}) \in [\boldsymbol{\epsilon}_{1:t}]$. Indeed, the sampling structure is not garanteed to be up-to-date for all subtrees of the sampling structure, and it is thus necessary to trace this upgrade level. On the other hand, function call $\mathtt{Cut}(v, [\boldsymbol{x}])$, which creates a cut of box $[\boldsymbol{x}]$, now depends on constraint level v. So, news cuts are created by considering all constraints information, although this will of course not be the case for cuts related to past levels $u < v$. At last, new function $\mathtt{Shrink}$ is introduced. Code $[\boldsymbol{x}] \Leftarrow \mathtt{Shrink}([\boldsymbol{l}], [\boldsymbol{r}])$ replaces $[\boldsymbol{x}]$ by the smallest box containing both $[\boldsymbol{l}]$ and $[\boldsymbol{r}]$; *this replacement is deep* and thus concerns dictionaries cuts, omg and lev as well. Even root variable $[\boldsymbol{b}]$ in algorithm 4 can have its value replaced. Nevertheless, deep replacements are cost-free here thanks to the way the structures are implemented.

In algorithm 4, parts taken unchanged from algorithm 3 are in black while evolutions are in blue. A **for** loop over the number of constraints, v, is added at lines 3 and 36 and repeats the entire sampling process for a given set of constraints. The sampling process itself does not change much, but incremental computations are now done on omg and lev is updated in consequence: lines 5 to 8, lines 16 to 19 and lines 20 to 23. Incremental computation means that undefined value $\mathsf{omg}([\boldsymbol{x}])$ is set equal to $\omega_{[\boldsymbol{x}]}^{1:v}$, while already defined value $\mathsf{omg}([\boldsymbol{x}])$ is multiplicatively completed by $\prod_{u=t+1}^{v} \varpi_{[\boldsymbol{x}]}^{u}$. At line 32, the shape of box $[\boldsymbol{x}_{i-1}]$ is recomputed from its cut. This is better for improving future incremental computation of omg during the sampling process.

3.3 Practical implementation and parallelism

Practical implementation. Algorithm 3 and 4 draw the main principles of our sampling methods. Some implementation details are described now.

Initial predictive weight. We implement the following definition of ω:

$$\omega_{[\boldsymbol{x}]} = \frac{\mu\big([g]([\boldsymbol{x}]) \cap [\boldsymbol{\epsilon}]\big)}{\mu\big([g]([\boldsymbol{x}])\big)} P\left(\boldsymbol{X} \in [\boldsymbol{x}]\right) , \tag{22}$$

where μ is Borel measure on $\mathbb{R}^m$. This definition checks properties (15) and (16). It also tries a rough approximation for $P\left(\boldsymbol{X} \in [\boldsymbol{x}] \,\&\, g(\boldsymbol{X}) \in [\boldsymbol{\epsilon}]\right)$. For incremental implementation, definition extends naturally as a product:

$$\omega^{1:v}_{[\boldsymbol{x}]} = P\left(\boldsymbol{X} \in [\boldsymbol{x}]\right) \prod_{u=1}^{v} \varpi^{u}_{[\boldsymbol{x}]} \text{ with } \varpi^{u}_{[\boldsymbol{x}]} = \frac{\mu\big([g_u]([\boldsymbol{x}]) \cap [\boldsymbol{\epsilon}_u]\big)}{\mu\big([g_u]([\boldsymbol{x}])\big)} . \tag{23}$$

Cutting strategy. The definition of `Cut` is an important choice. Our algorithm selects a cut $([\boldsymbol{y}], [\boldsymbol{z}])$ randomly in regards to the following criteria:

- Favor cuts such that $\omega_{[\boldsymbol{y}]} \ll \omega_{[\boldsymbol{z}]}$ or $\omega_{[\boldsymbol{y}]} \gg \omega_{[\boldsymbol{z}]}$,
- Avoid overly elongated $[\boldsymbol{x}]$, *i.e.* such that $\frac{\max_i(x_i^+ - x_i^-)}{\min_i(x_i^+ - x_i^-)} \gg 1$,

In addition, some cuts history simplifications are implemented (next point).

Reducing sampling structure. Our implementation tries to optimize the structure of cut, weight, and level history, when it is possible. Different cases are implying such reductions, which are not mentioned here. A typical example is shown in figure 3. Assume that $([\boldsymbol{y}], [\boldsymbol{z}])$ is a cut of $[\boldsymbol{x}]$, $([\boldsymbol{t}], [\boldsymbol{u}])$ is a cut of $[\boldsymbol{z}]$ and $[g]([\boldsymbol{u}]) \cap [\boldsymbol{\epsilon}] = \emptyset$, *i.e.* $[\boldsymbol{u}]$ is outside the subpaving and its border. Then, boxes $[\boldsymbol{z}]$ and $[\boldsymbol{u}]$ are useless and should be removed from the structure.

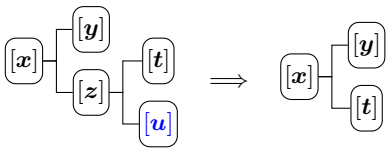


Fig. 3: Structure reduction

Discarding first samples. M first samples are discarded, so as to initialize the structure of the sampler. After that, N samples are sampled and returned.

Setup. The algorithms are rather simple to set up. Except for the choice of ω, which is structural, r, M and σ are the only parameters to be defined.

Parallelization. This paragraph gives additional information on how incremental algorithm 4 is implemented with parallelization. Data structures cuts, omg and lev deal rather well with parallel processing, so that our implementation is multithread. Nevertheless, the practical implementation needs some work.

In our implementation, the dictionaries cuts, omg and lev are stored in a same structure. This structure is composed of shared references to vertices which are stored within a Slab allocation [2]. Slab allocation is a structure, which allows easy and efficient deallocation and reallocation, while reducing the risk of framentation of the memory. A vertex is an element of the history of the sampling process, which contains the following fields:

- A box $[\boldsymbol{x}]$ with its weight $\mathsf{omg}([\boldsymbol{x}])$ and its level $v = \mathsf{lev}([\boldsymbol{x}])$, which is an indicator that $\mathsf{omg}([\boldsymbol{x}])$ has been computed from $g_{1:v}$ and $[\boldsymbol{\epsilon}_{1:v}]$,
- The type of vertex: a node or a leaf? Is property $[g_{1:v}]([\boldsymbol{x}]) \subset [\boldsymbol{\epsilon}_{1:v}]$ true?
- When vertex is a node and box $[\boldsymbol{x}]$ has a cut $([\boldsymbol{l}], [\boldsymbol{r}]) = \mathsf{cuts}([\boldsymbol{x}])$, then the vertex is completed with weak references to the vertices related to boxes $[\boldsymbol{l}]$ and $[\boldsymbol{r}]$ respectively.

Shared references allow multiple threads to access a same data; moreover, the existance of a shared reference will prevent the erasure of referenced data. Weak references are derived from shared references and also allow threads to access the data, but they do not prevent the erasure of referenced data. By restricting the shared references to the Slab allocation, we prevent the possibility of memory leak, especially when the sampling structure is reduced by a thread. Since the structure is shared among threads, the access to the vertices are protected by memory locks: the locks allow either many read-only accesses or only one read-write access to the data.

This data structure allowed some fair performance increase in comparison to conference paper [4]. The memory management is now smoother and more efficient. In particular, we no longer have process freezing phenomena following heavy memory management.

The definition of an efficient data structure is a main achievement when designing a multithread implementation. Now, we give some clarification on how the sampling process is shared between the threads:

- Multiple instances (usually as many as the number of processor threads) of the sampling loop are run simultaneously. This includes descending through the sampling structure, possible extension of this structure, possible sampling of box and point if conditions are met, and backward reconstruction of weights and restructuring of the sampling structure,
- If a vertex currently processed by an instanced loop has been deleted by another instance, then the vertex subtree is first deleted by the process, and afterward, the loop is restarted.
- Processes are stop when the desired number of samples is obtained,

4 Application to Bayesian optimization

Bayesian optimization is a key technique for black-box optimization, and it is actually a great motivation for this work.

Assume that one needs to optimize a function which is not well known, and which may be computed by a highly costly process (a heavy simulation, tests made by human teams on the grounds, etc.). Of course the optimization should be made by sparing at best the number of calls to the costly evaluation.

In [8], Jones, Schonlau and Welch proposed the efficient global optimization method (EGO) for addressing such kind of problem. The idea is to use a surrogate model under the form of a functional random variable. This functional random variable is described by means of a Gaussian process with correlation depending

on spatial distance (kriging). Based on such modelling, the construction of an *optimal* parameter sequence to be evaluated is obtained by iterating:

- Compute the posterior law of the functional random variable, according to the past evaluations,
- Compute the expected improvement function, in regards to the posterior law and the already best computed value. This expected improvement is an indicator of the next parameter to be evaluated,
- Find parameter optimizing the expected improvement and evaluate it.

The approach relies on the construction of the Gaussian modelling, and it tends to be less efficient when the dimension of the optimization space increases. Works have been made in order to deal with this dimensional issue and improve the modelling; *e.g.* [6]. But this issue is still a true challenge.

Beside, EGO itself is a form of Bayesian optimization introduced by Mockus [10, 9, 13]. We proposed in [3] a Bayesian optimization approach in the context of a nonlinear function depending on a model noise; the realization of the noise was the unknown information of the problem. We hoped by a nonlinear approach to better handle some of the difficulty induced by the dimension. The work was not complete, since we were not able to build a good conditional sampling at that time. The purpose of this paper and of [4] was to build such sampler.

It is beyond the scope of this chapter to detail the seminal works of Mockus, or those of Jones, Schonlau and Welch. We will introduce the subject by a short description of EGO algorithm. Then, we present our nonlinear approach.

Formalization: Function $\gamma \mapsto g(\gamma, \boldsymbol{x}^o)$ is to be minimized. Parameter $\boldsymbol{x}^o$ is unknown, but $\boldsymbol{x}^o$ is a realization of random vector $\boldsymbol{X}$, whose law $F_{\boldsymbol{X}}$ is known. In order to optimize γ, we are allowed to request an evaluator for computing $g(\gamma, \boldsymbol{x}^o)$, but each call to this evaluator is costly. The objective is then to solve $\gamma^o \in \arg\min_\gamma g(\gamma, \boldsymbol{x}^o)$ by optimizing at the same time the sequence of evaluated cases γ_u and of their evaluations $e_u = g(\gamma_u, \boldsymbol{x}^o)$.

EGO method: Main idea of EGO consists in approximating $g(\gamma, \boldsymbol{X})$ by a Gaussian process (GP) $\gamma \mapsto G(\gamma)$. Given past evaluated cases γ_u and evaluations $e_u = g(\gamma_u, \boldsymbol{x}^o)$ for $u \leq v$, posterior variable $[G(\gamma) \,|\forall u \leq v, G(\gamma_u) = e_u\,]$ is Gaussian and mathematically computed. This conditional variable is a main ingredient for computing the expected improvement, $EI(\gamma)$, which is an indicator of value γ for which next evaluation of $g(\gamma, \boldsymbol{x}^o)$ is promizing:

$$EI(\gamma) = E_{G(\gamma)|\forall u \leq v, G(\gamma_u)=e_u} \max\{m_\gamma - G(\gamma), 0\} \text{ where } m_\gamma = \min_{1 \leq u \leq v} e_u\,. \quad (24)$$

It is computed mathematically. Since $EI(\gamma)$ indicates where evaluations are promising, it is maximized in order to chose next case $\gamma_{v+1} \in \arg\max_\gamma EI(\gamma)$ and request its evaluation $e_{v+1} = g(\gamma_{v+1}, \boldsymbol{x}^o)$.

Nonlinear algorithm: We took inspiration of EGO, but now, random function $g(\cdot, \boldsymbol{X})$ is used directly instead of GP approximation. Parameters γ_v and evaluations $e_v = g(\gamma_v, \boldsymbol{x}^o)$ are optimized by iterating algorithm 5. While $EI(\gamma)$ is formed differently in this algorithm, it is essentially the same as in definition (24) modulo an affine transform. It is theoretically defined by:

$$EI(\gamma) = E_{\boldsymbol{X}|\forall u \le v, g(\gamma_u, \boldsymbol{X}) = e_u} \min\{g(\gamma, \boldsymbol{X}), m_\gamma\} \quad \text{with} \quad m_\gamma = \min_{1 \le u \le v} e_u \,. \tag{25}$$

In practice, $EI(\gamma)$ is obtained by a Monte Carlo derived from an approximated sampling of $[\boldsymbol{X} \,|\forall u \le v, g(\gamma_u, \boldsymbol{X}) = e_u\,]$. *As a crucial ingredient:*

- Vector $[\boldsymbol{X} \,|\forall u \le v, g(\gamma_u, \boldsymbol{X}) = e_u\,]$ is approximated by $[\boldsymbol{X} \,|g_{1:v}(\boldsymbol{X}) \in [\boldsymbol{\epsilon}_{1:v}]]$, where $g_u = g$ and $\boldsymbol{\epsilon}_u$ is a small box around e_u.
- The sampling is done by the way of this approximation.

This approximated random vector, conditional to a subvariety, incorporates incrementally all constraints related to the past evaluations. The sampling is done by algorithm 3 or preferably by incremental algorithm 4.

Algorithm thus consists of:

1. Generating samples of posterior random vector $[\boldsymbol{X} \,|g_{1:v}(\boldsymbol{X}) \in [\boldsymbol{\epsilon}_{1:v}]]$,
2. Building a Monte Carlo approximation of $EI(\gamma)$,
3. Choosing parameter γ_{v+1} by minimizing $EI(\gamma)$ and evaluating it.

Function *Process next measure*
  input : γ_u and $e_u \triangleq g(\gamma_u, \boldsymbol{x}^o)$ for $1 \le u \le v$
  output: γ_{v+1} and $e_{v+1} \triangleq g(\gamma_{v+1}, \boldsymbol{x}^o)$
  Make samples of $[\boldsymbol{X} \,|g_{1:v}(\boldsymbol{X}) \in [\boldsymbol{\epsilon}_{1:v}]]$
  Compute $m_\gamma = \min_{1 \le u \le v} e_u$ and:
   $EI(\gamma) \simeq E_{\boldsymbol{X}|g_{1:v}(\boldsymbol{X}) \in [\boldsymbol{\epsilon}_{1:v}]} \min\{g(\gamma, \boldsymbol{X}), m_\gamma\}$
  Compute $\gamma_{v+1} \in \arg\min_\gamma EI(\gamma)$
  Compute $e_{v+1} \triangleq g(\gamma_{v+1}, \boldsymbol{x}^o)$
end

Algorithm 5: Sampler-based Bayesian optimizer

Implementations of step 4 are not detailed here. Essentially, we used a metaheuristic method, the cross-entropy algorithm for optimization [12], for this task; this method is related to parameterized rare-event simulation.

5 Examples and tests

Regarding the conference publication [4], all the results presented here have been recomputed or supplemented by recent implementations of the algorithms.

5.1 Simulation: test cases

The tests presented here are performed for sampling algorithm 3 or its incremental version 4. The algorithms have been implemented in `Rust` language (`www.rust-lang.org`) and were processed on 7 threads. The algorithms are tested on mathematically simple simulation problems, in order to make the statistics of the results clear enough to analyze.

Parameters. All simulations have been achieved with the following parameters:

- $r = 0.001$ is the radius bound for second **while** stop condition,
- $M = 5000$ is the number of samples discarded during sampler initialization,
- $N = 50000$ is the number of sampled particles,
- $\sigma = 10$ on all tests.

Test Cases. Thorough the section, it is assumed that $\boldsymbol{X}$ follows the uniform law on $\boldsymbol{b} = [-2,2]^n$ with $n \in \{2,3,\cdots,11\}$. Three cases are investigated:

Case (a): Are defined $g_a(\boldsymbol{x}) = ||\boldsymbol{x}||_2 = \sqrt{\sum_{j=1}^n x_j^2}$ and $[\boldsymbol{\epsilon}_a] = [0.95, 1.05]$. Then $g_a^{-1}([\boldsymbol{\epsilon}_a])$ is a hyper-spherical shell, which approximates the unit hypersphere of dimension $n-1$.

Case (b): For $n = 11$ and $0 \leq v \leq 9$, it is considered:

$$g_b(\boldsymbol{x}) = \left[\begin{array}{cccc} ||\boldsymbol{x}||_2 & x_3 & \ldots & x_{2+v} \end{array} \right] \text{ and } [\boldsymbol{\epsilon}_b] = [\boldsymbol{\epsilon}_a] \times [\boldsymbol{z}]^v , \tag{26}$$

with $[\boldsymbol{z}] = [-0.05, 0.05]$. Similarly to **(a)**, but with additional constraints, $g_b^{-1}(\boldsymbol{\epsilon}_b)$ approximates a hypersphere of dimension $n-1-v$. When $v = 0$, we are back to case **(a)** with $n = 11$. When $v = 9$, then $g_b^{-1}(\boldsymbol{\epsilon}_b)$ approximates the unit circle $\mathcal{C}$ within the first two dimensions, related to coordinates x_1, x_2.

Case (c): For $n = 11$ and $0 \leq v \leq 9$, it is considered:

$$g_c(\boldsymbol{x}) = \left[\begin{array}{cccccc} ||\boldsymbol{x}||_2 & \min(|x_1|,|x_2|) & x_3 & \ldots & x_{2+v} \end{array} \right], \tag{27}$$

and $[\boldsymbol{\epsilon}_c] = [\boldsymbol{\epsilon}_a] \times [\boldsymbol{\alpha}] \times [\boldsymbol{z}]^v$ with $[\boldsymbol{\alpha}] = [0, 0.5]$. Thus, this case is obtained by adding constraint $0 \leq \min(|x_1|,|x_2|) \leq 0.5$ to subcases of **(b)**. For subcase $v = 9$ especially, we are approximately sampling on the (disjoint) union of the 4 subsegments $\mathcal{A}_1, \ldots, \mathcal{A}_4$ of the unit circle $\mathcal{C}$, defined by:

$$\mathcal{A}_j = \left\{ \left[x_1, x_2\right] \in \mathcal{C} \middle/ \arg\left(\left[x_1, x_2\right]\right) \in j\frac{\pi}{2} + \left[-\frac{\pi}{6}, \frac{\pi}{6}\right] \mod 2\pi \right\}. \tag{28}$$

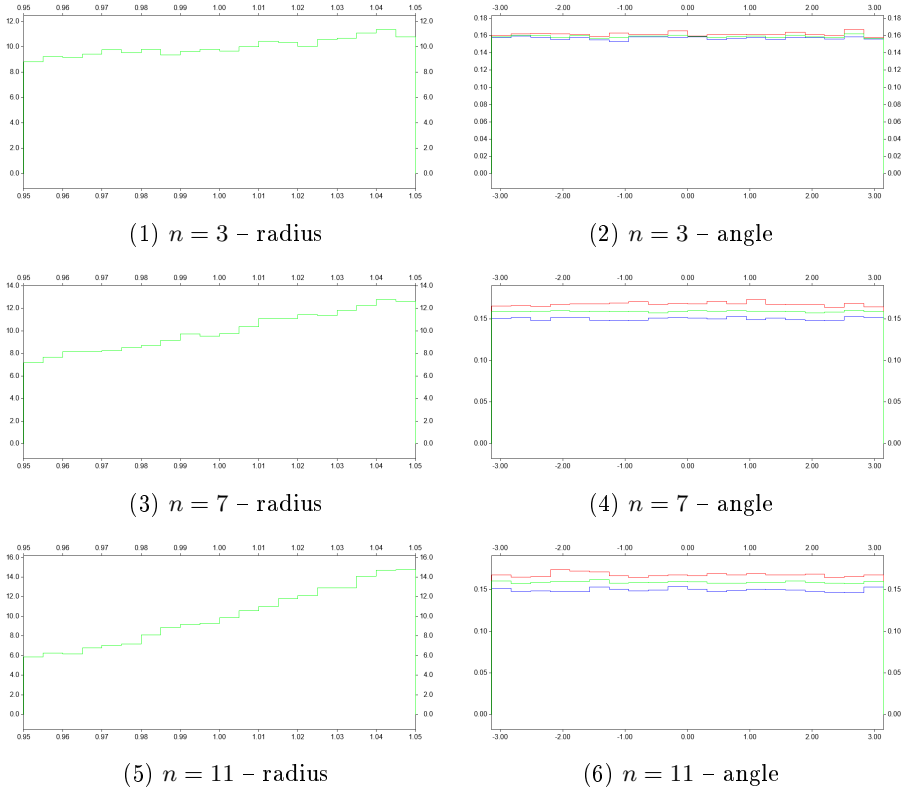
These 4 subsegments have the same size so that their probabilities are the same in regard to X.

Purpose of the Test Cases. Subsequently, case **(a)** is used in order to evaluate the performance of the sampling process both in accuracy and in efficiency for different dimensions. Case **(b)** is used in order to evaluate the efficiency of the sampling when the number of constaints increases; algorithm 3 and incremental algorithm 4 are also compared. Case **(c)** is used in order to evaluate the accuracy of the sampling in case of complex constaints which introduce disjoint modes.

5.2 Simulation: case (a)

This case is mathematically easy to predict. In [5], Dezert and Musso proposed a method, which may be used for uniformly sampling on an ellipsoid shell. Whatever, one must keep in mind that our approach is generic and can be applied to an infinite number of configurations.

Histograms. Figure 4 presents some histograms for case (a). For each subcase $n \in \{3, 7, 11\}$, we have computed the radius of all samples $\boldsymbol{x}$ and built the associated histograms – subfigures (1), (3) and (5). For each $n \in \{3, 7, 11\}$ and for all $1 \leq i < j \leq n$, we have computed the angle of all samples (x_i, x_j) and built the associated histograms. From these $\frac{n(n-1)}{2}$ histograms of each subcase, we have computed the minimal, mean and maximal histograms. The results are shown in blue, green and red, respectively, and provide an hint on the error of the estimation – subfigures (2), (4) and (6).



(1) $n = 3$ – radius (2) $n = 3$ – angle

(3) $n = 7$ – radius (4) $n = 7$ – angle

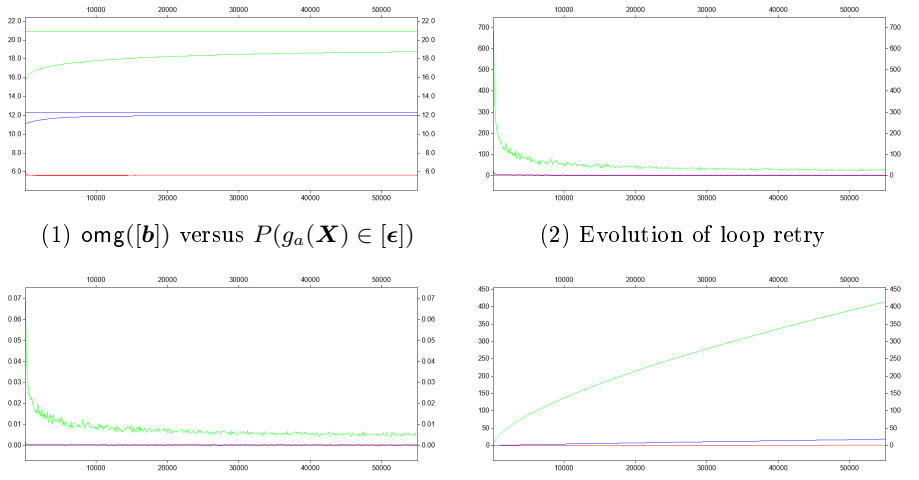
(5) $n = 11$ – radius (6) $n = 11$ – angle

Fig. 4: Case **(a)** – histograms – 20 divisions

By symmetry, the *theoretical angle histograms* are uniform. The errors should be of the order of $\sqrt{\frac{20}{50000}} = 0,02$. In comparison, the errors figured in the angular histograms are quite acceptable, even for the highest dimension. The most interesting point is that there is no rupture in the histogram, which shows that the sampler does manage the subvariety structure. We do not have an error estimation for the radius histograms. Actually, the local probability should theoretically increase with the radius, this property being accentuated with the

dimension. This is what is obtained on the histograms. It is noteworthy however that the sides of these histograms are subject to additional errors implied by the border of the subvariety.

Process Statistics: Figure 5 presents some statistics graphs for case (a).



(1) $\mathsf{omg}([\boldsymbol{b}])$ versus $P(g_a(\boldsymbol{X}) \in [\boldsymbol{\epsilon}])$ (2) Evolution of loop retry

(3) Evolution of cpu time (s) (4) Cumulative cpu time (s)

Fig. 5: Case **(a)** – statistics – $n = 3, 7, 11$

Results are plotted against the number of generated samples $k = 1 : 55000$. Cases $n = 3, 7, 11$ are drawn with respective colors, red, blue and green.

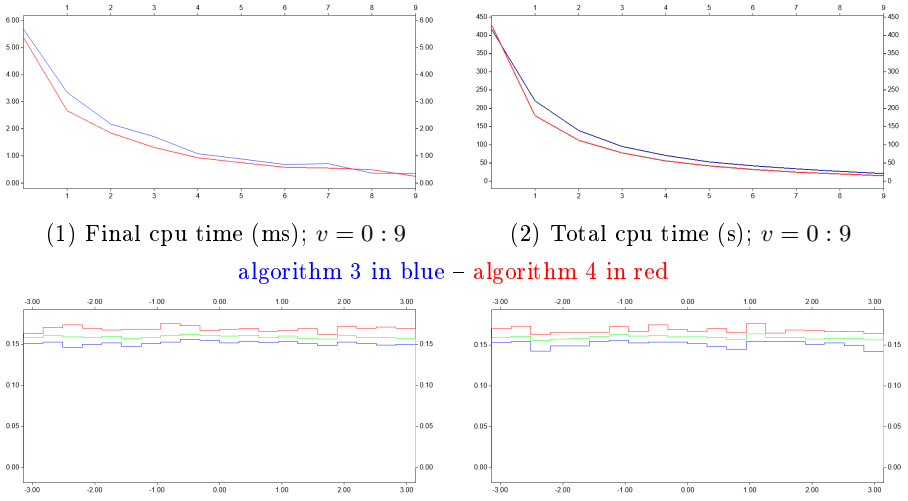
Subfigure (1) shows how value $-\log_2(\mathsf{omg}([\boldsymbol{b}]))$ evolves and approximates $-\log_2(P(g_a(\boldsymbol{X}) \in [\boldsymbol{\epsilon}]))$. Each curve increases to theoretical value (same color line), but performance decreases with dimension.

Subfigures (3) and (4) present the cumulative cpu-time per thread and the evolution of the cpu-time per sample and per thread consumed by the process (expressed in second). In comparison to [4], there is no more time discontinuity caused by intermittent memory allocations. Our memory management has been improved as shown in section 3.3. We notice clearly that the sampling efficiency increases with the number of generated samples. However, the cumulative cpu time still increases dramatically with the dimension (the memory use evolves similarly). Although the curse of dimension has been delayed by our approach, it is still there.

The number of loop retries during the sampling is plotted in subfigure (2). It is an interesting indication of the achievement of the sampling structure. It decreases with the number of samples and becomes small, even for the highest dimension (around 28 for $n = 11$). This result should be compared to the probability of the subvariety (around 10^{-6} for $n = 11$).

5.3 Simulation: case (b)

Figure 6 presents synthetic performance curves, as well as comparative histogram examples for algorithm 3 and algorithm 4 on case **(b)**.



(1) Final cpu time (ms); $v = 0 : 9$ (2) Total cpu time (s); $v = 0 : 9$

algorithm 3 in blue – algorithm 4 in red

(3) Angle histogram – $v = 5$ (4) Angle histogram – $v = 5$

Fig. 6: Case **(b)** – speed and histograms

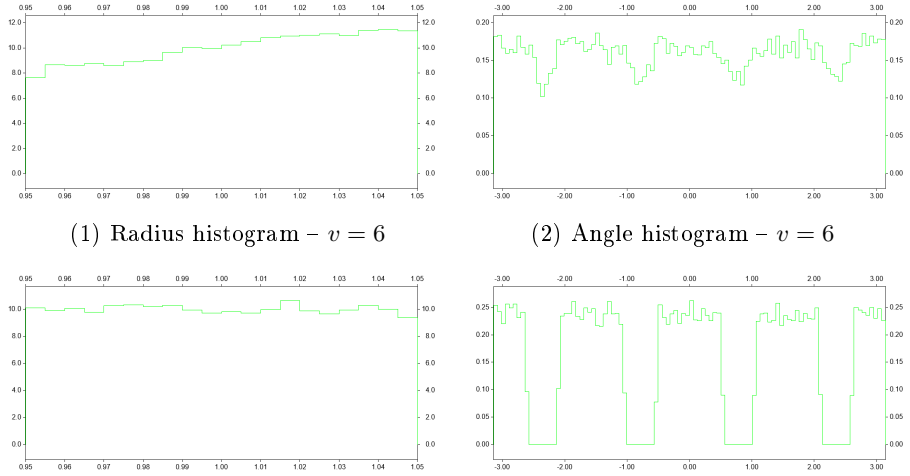
Subfigures (1) and (2) are plotted against the number of additional constraints $v = 0 : 9$. Subfigure (1) presents final cpu-time (in ms) per sample and per thread consumed by algorithm 3 (in blue) and algorithm 4 (in red). Both curves decrease and approach to zero. Subfigure (2) presents total cpu-time (in s) per thread consumed by these algorithms. Both curves decrease. Red curves are dominated by blue curves, which means better efficiency of algorithm 4 against algorithm 3.

Subfigure (3) presents angle histogram for subcase $v = 5$ from computations by algorithm 3. Subfigure (4) presents angle histogram for same subcase from computations by algorithm 4. Both results are comparable in terms of accuracy. The theoretical histogram is uniform, which conforms to these results.

As a conclusion here, the performance of the sampler is likely to increase with the number of constraints, and this is a useful quality. Incremental algorithm 4 is more efficient than algorithm 3, while ensuring equivalent accuracy. Algorithm 4 must therefore be favored for incremental constraints. In last example of section 5.5 on Bayesian optimisation, implementation of algorithm 4 was even necessary for an acceptable computation time.

5.4 Simulation: case (c)

Figure 7 presents computed radial histograms and angular histograms (for coordinates x_1 and x_2 – there is only one) for subcases $v = 6$ and $v = 9$.



(1) Radius histogram – $v = 6$ (2) Angle histogram – $v = 6$

(3) Radius histogram – $v = 9$ (4) Angle histogram – $v = 9$

Fig. 7: Case **(c)** – histograms – radius: 20 divisions; angle: 100 divisions

Subfigure (3) presents radial histogram for final case $v = 9$. Subfigure (4) presents angle histogram for final case $v = 9$. The quality of the histograms is comparable to what was observed previously. Due to the constraint configuration of subcase $v = 9$, the *theoretical radius histogram* is uniform, and the *theoretical angular histogram* is uniform around each subsegment, $\mathcal{A}_1$ to $\mathcal{A}_4$, with a gradual decrease on the borders. The generated histograms actually comply with these properties.

Subfigures (1) and (2) present intermediate subcase $v = 6$. Since some other coordinates than x_1 and x_2 are relaxed, contraint $x_1^2 + x_2^2 \simeq 1$ is also relaxed. Actually, theoretical angular histogram is even no longer discontinuous around subsegments, $\mathcal{A}_i$, for this reason.

As a preliminary conclusion, we consider that our sampling method is globally performant in sampling conditionally to subvarieties. A future issue will be to increase the dimension of the sampling space.

5.5 Bayesian optimization

A simple geometric problem. We intend to find the isobarycenter $\gamma = (a, b)$ of 4 *unknown points* $M_i = (x^o_{2i-1}, x^o_{2i}) \in [-5, 5]^2$ with $i \in \{1, 4\}$. The only

approach that is possible for us is to test some solutions by requesting for a costly measurement; for requirement $\gamma_u = (a_u, b_u)$, this measurement evaluates:

$$e_u = g(\gamma_u, \boldsymbol{x}^o) = ||\gamma_u - h(\boldsymbol{x}^o)||_2 \quad \text{with} \quad h(\boldsymbol{x}^o) = \frac{1}{4}\sum_{i=1}^{4} M_i = \frac{1}{4}\sum_{i=1}^{4}(x^o_{2i-1}, x^o_{2i})\,. \tag{29}$$

Our purpose is to optimize (a, b) by minimally requesting evaluation $g(\gamma, \boldsymbol{x}^o)$.

Geometric solution. Each measure restricts the solution to a circle. After 2 measures, we usualy have to choose between two points, and the solution is found equiprobably at step 3 or 4.

Tests and Results. Points $M_1, \ldots, M_4$ are $(2, -1)$, $(3, 2)$, $(-\frac{3}{2}, 4)$, $(\frac{1}{2}, 3)$. Their isobarycenter is $(1, 2)$. We used a sampler with $M = 5000$, $N = 10000$ and $[\boldsymbol{\epsilon}] = [-\frac{1}{100}, \frac{1}{100}]^u$. Variable $\boldsymbol{X}$ is considered uniform on $[-5, 5]^8$. Table 1 presents a typical optimized sequence for parameters $\gamma_u = (a_u, b_u)$. Optimization of EI is done the cross-entropy method [12] and is thus near-optimal. In this example,

Table 1: Exemple of optimization sequence

u	1	2	3	4	5	6	7	8	9	10	11	12	13
a_u	0.01	−0.41	0.84	0.97	0.99	1.00	0.98	1.02	−3.79	0.98	0.98	−0.39	1.03
b_u	0.02	−1.94	2.05	1.99	2.02	1.98	2.02	1.99	6.49	2.00	1.99	5.40	2.00
e_u	2.21	4.19	0.16	0.03	0.02	0.02	0.03	0.02	6.57	0.02	0.03	3.67	0.03

best value for e_u is 0.02. Best values are generally found around 0.02: this is a consequence of error interval $[\boldsymbol{\epsilon}]$, which is not zero size.

Table 2: Results for 100 runs

u_o	1	2	3	4	5	6	7	8	9	10	11	12	13
$e_{u_o} \leq 0.02$	0%	0%	26%	40%	10%	9%	2%	3%	3%	3%	1%	1%	1%
$e_{u_o} \leq 0.04$	0%	0%	36%	57%	6%	1%	0%	0%	0%	0%	0%	0%	0%
$e_{u_o} \leq 0.09$	0%	0%	40%	57%	3%	0%	0%	0%	0%	0%	0%	0%	0%

Table 2 presents the results for 100 runs. For these runs, the mean cpu time is $2627s$, where *non-incremental* algorithm 3 has been used for simulation. Three convergence conditions, $e_u \leq e_{cond}$ with $e_{cond} \in \{0.02, 0.04, 0.09\}$, are considered for these runs. Value $u_o = \min\{u / e_u \leq e_{cond}\}$ is the final step for which convergence condition is met. Percentage is given for each final step value u_o. In all cases, we notice that most optimal values are found at steps 3 or 4. The result tends to be equilibrated on 3 and 4 when stopping criterion is relaxed.

EGO method. For the sake of comparison, we apply EGO to our problem. Gaussian process G with covariance $\mathrm{cov}(G(\gamma), G(\gamma')) = \frac{1}{4}\exp\left(-\frac{1}{2}||\gamma-\gamma'||_2\right)$ and mean 0 is used. Table 3 summarizes the results of 100 tests. Each test imple-

Table 3: Results for EGO method

e_{cond}	0.5	0.2	0.1	0.05	0.02	0.01	0.005	0.002	0.001
$\% : u_o \leq 100$	91%	74%	69%	64%	55%	46%	42%	29%	14%
worse(u_o)	55	78	96	97	97	84	93	93	99
mean(u_o)	11.8	19.4	23.1	25.5	30	31.3	37.5	56.3	60.5
best(u_o)	1	1	8	8	8	15	15	19	19

ments 100 successive evaluations. Convergence conditions are $e_u \leq e_{cond}$ with different values for e_{cond} ranging from 0.5 to 0.001. Again, value u_o is the final step for which convergence condition is met. For each value e_{cond}, the table indicates the percentage of tests which succeeded to reach convergence with less than 100 evaluations. In case of convergence, the worse, mean and best final steps are given. EGO is outperformed here. However, EGO is able to achieve more refined results – *e.g.* $e_{cond} = 0.001$ – if the evaluation budget is relaxed: indeed, EGO is based on exact mathematical computation of the expected improvement.

Decision space of higher dimension. Variable $\boldsymbol{X}$ is uniform on $[-5,5]^8$ with realization $\boldsymbol{x}^o = (2, -1, 3, 2, -\frac{3}{2}, 4, \frac{1}{2}, 3)$. Points $M_{i,j} = (m_{i,j,k})_{1\leq k\leq 28}$ with $1 \leq j < i \leq 8$ are built from $\boldsymbol{X}$ by setting $m_{i,j,k} = X_i X_j$ if $k = j + \frac{(i-2)(i-1)}{2}$ and $m_{i,j,k} = 0$ else. Function $g(\gamma, \boldsymbol{X})$ is defined by $g(\gamma, \boldsymbol{X}) = ||\gamma - h(\boldsymbol{X})||_2$ where $h(\boldsymbol{X}) = \frac{1}{28}\sum_{i=2}^{8}\sum_{j=1}^{i-1} M_{i,j}$. The problem is thus of much higher dimension.

We had some theoretical issues on this example. The Bayesian estimation was unable to resolve some remaining ponctual cases for zero dimension subvarieties. This difficulty has been overcome by adding exclusion constraints around already tested cases. Table 4 gives an example of run (progress is in bold). Incremental

Table 4: Example in a space of higher dimension

u	**1**	2	3	**4**	**5**	6	**7**	8	**9**	**10**	11	12	**13**	14	15	16	**17**	18	19	20
e_u	1.02	1.04	1.18	0.77	0.68	0.68	0.61	1	0.33	0.12	0.18	0.21	0.09	0.09	0.09	0.09	0.05	0.10	0.09	0.12

algorithm 4 were needed for simulation, and optimization has been performed within a day. On the other hand, algorithm 3 was far too slow in this case for optimizing in reasonable time.

6 Conclusion

We proposed an original dichotomous method for sampling a random vector conditionally to a subvariety. This method has been parallelized, and we proposed

non-incremental and incremental implementations. This generic approach, inspired from interval analysis, is accurate and efficient up to a space of dimension 11. We have shown how it could be applied efficiently to Bayesian optimization problems. The work is promizing from theoretical and applicative point of view and offers some improvement perspectives. A main issue is to enhance the efficiency of the approach with respect to higher dimensions. With this perspective in mind, some relaxation techniques applied to the subvariety may be considered.

References

1. Alefeld, G., Mayer, G.: Interval analysis: theory and applications. Journal of Computational and Applied Mathematics **121**(1), 421–464 (2000)
2. Bonwick, J., Microsystems, S.: The slab allocator: An object-caching kernel memory allocator. In: In USENIX Summer. pp. 87–98 (1994)
3. Dambreville, F.: Optimizing a sensor deployment with network constraints computable by costly requests. In: Thi, H.A.L., Dinh, T.P., Nguyen, N.T. (eds.) Modelling, Computation and Optimization in Information Systems and Management Sciences. Advances in Intelligent Systems and Computing, vol. 360, pp. 247–259. Springer (2015)
4. Dambreville, F.: Simulating a Random Vector Conditionally to a Subvariety: A Generic Dichotomous Approach. In: SIMULTECH'21, 11th International Conference on Simulation and Modeling Methodologies, Technologies and Applications. pp. 109–120 (Jul 2021)
5. Dezert, J., Musso, C.: An efficient method for generating points uniformly distributed in hyperellipsoids. In: Workshop on Estimation, Tracking and Fusion: A Tribute to Bar-Shalom. Monterey, California (May 2001)
6. Hebbal, A., Brevault, L., Balesdent, M., Taibi, E.G., Melab, N.: Efficient global optimization using deep gaussian processes. In: 2018 IEEE Congress on Evolutionary Computation (CEC). pp. 1–8 (2018)
7. Jaulin, L., Kieffer, M., Didrit, O., Walter, E.: Applied Interval Analysis with Examples in Parameter and State Estimation, Robust Control and Robotics. Springer London Ltd (Aug 2001)
8. Jones, D., Schonlau, M., Welch, W.: Efficient global optimization of expensive black-box functions. Journal of Global Optimization **13**, 455–492 (12 1998). https://doi.org/10.1023/A:1008306431147
9. Mockus, A., Mockus, J., Mockus, L.: Bayesian approach adapting stochastic and heuristic methods of global and discrete optimization. Informatica (lithuanian Academy of Sciences) **5**, 123–166 (1994)
10. Močkus, J.: On bayesian methods for seeking the extremum. In: Marchuk, G.I. (ed.) Optimization Techniques IFIP Technical Conference Novosibirsk, July 1–7, 1974. pp. 400–404. Springer Berlin Heidelberg, Berlin, Heidelberg (1975)
11. Morio, J., Balesdent, M., Jacquemart, D., Vergé, C.: A survey of rare event simulation methods for static input–output models. Simulation Modelling Practice and Theory **49**, 287–304 (2014)
12. Rubinstein, R.Y., Kroese, D.P.: The Cross Entropy Method: A Unified Approach To Combinatorial Optimization, Monte-Carlo Simulation (Information Science and Statistics). Springer-Verlag (2004)
13. Zilinskas, A., Zhigljavsky, A.: Stochastic global optimization: A review on the occasion of 25 years of informatica. Informatica **27**, 229–256 (01 2016)