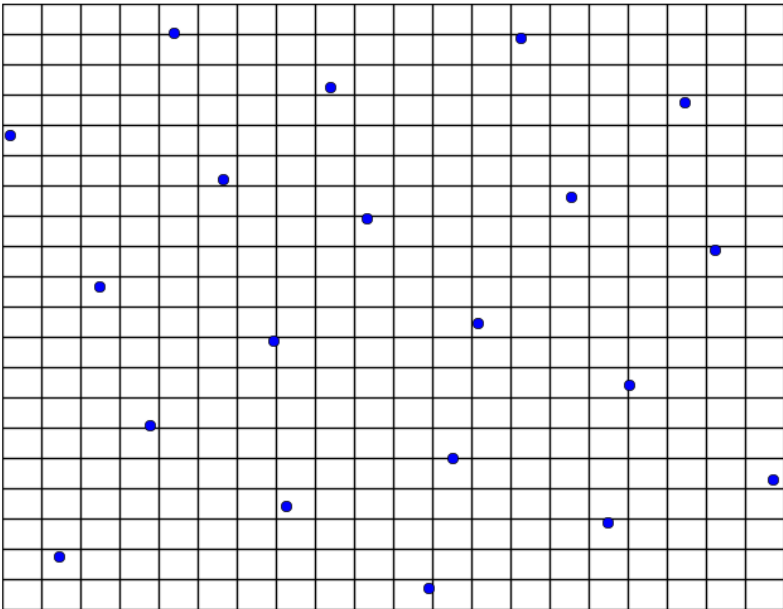# Documentation of the OTLHS module

Documentation built from package otlhs-1.3

November 18, 2016

Abstract

The purpose of this document is to present the OTLHS module.
This document is organized according to the OpenTURNS documentation:

- *Architecture Guide gives UML diagrams of all implemented classes,*

- *Reference Guide gives some theoretical basis,*

- *Use cases Guide details scripts in Python (the Textual Interface language of OpenTURNS) and helps to learn as quickly as possible the manipulation of the otlhs module,*

- *User Manual details the otlhs objects and gives the list of their methods,*

- *Validation Guide which provides use cases to validate the otlhs module.*

Contents

1 Architecture guide

This document makes up the general specification design for the architecture of the OTLHS module. Several classes implement the *pimpl* idiom (pointer to implementation) to hide implementation details.
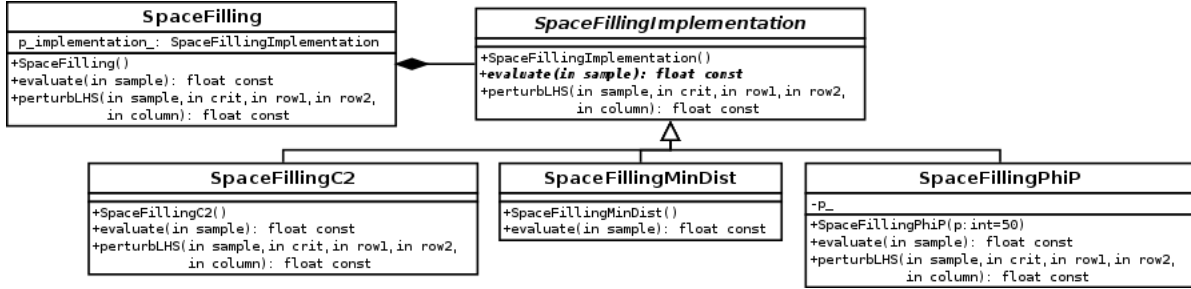
1.1 SpaceFilling

Figure 1: SpaceFilling classes

These classes implement different space filling criteria, which are consumed by `OptimalLHS` algorithms.
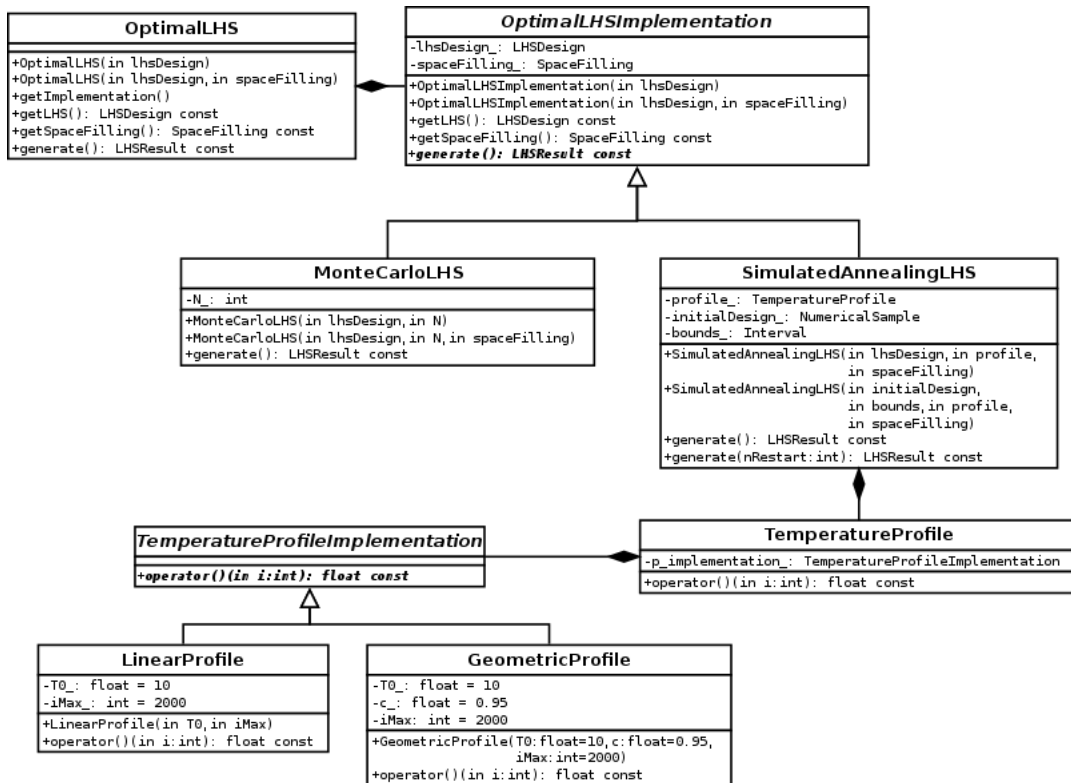
1.2 OptimalLHS

Figure 2: OptimalLHS classes

This is the main class hierarchy. The `OptimalLHS` class is an abstract class, inherited by `MonteCarloLHS` and `SimulatedAnnealingLHS`. They implement the `generate` method, which returns an `LHSResult` instance.

1.3 LHSResult

```
                         LHSResult
-bounds_: Interval
-spaceFilling_: SpaceFilling
-restart_: int
-optimalCriterion_;: float
-optimalIndex_: int
-collDesigns_: Collection<NumericalSample>
-collAlgoHistory_: Collection<NumericalSample>
-criteria_: NumericalSample
+LHSResult()
+LHSResult(in bounds:Interval,in spaceFilling:SpaceFilling)
+LHSResult(in bounds:Interval,in spaceFilling:SpaceFilling,
          in restart:int)
+add(in design:NumericalSample,in criterion:float,
     in C2:float,in PhiP:float,in MinDist:float,
     in algoHistory:NumericalSample)
+getBounds(): Interval const
+getNumberOfRestarts(): int const
+getOptimalDesign(): NumericalSample const
+getOptimalDesign(in restart:int): NumericalSample const
+getOptimalValue(): float const
+getOptimalValue(in restart:int): float const
+getAlgoHistory(): NumericalSample const
+getAlgoHistory(in restart:int): NumericalSample const
+getC2(): float const
+getC2(in restart:int): float const
+getPhiP(): float const
+getPhiP(in restart:int): float const
+getMinDist(): float const
+getMinDist()(in restart:int): float const
+drawHistoryCriterion(in title:String): Graph const
+drawHistoryCriterion(in restart:int,in title:String): Graph const
+drawHistoryProbability(in title:String): Graph const
+drawHistoryProbability(in restart:int,in title:String): Graph const
+drawHistoryTemperature(in title:String): Graph const
+drawHistoryTemperature(in restart:int,in title:String): Graph const
```

```
                      LHSDesign
-bounds_: Interval
-size_: int
-centeredDesign_: bool = false
+LHSDesign(in bounds:Interval,in size:int,
           in centeredDesign:bool=false)
+getBounds(): Interval const
+getSize(): int const
+isCenteredDesign(): bool const
+generate(): NumericalSample const
```

Figure 3: LHSDesign and LHSResult classes

As OpenTURNS already provides an `LHS` class, we had to rename it into `LHSDesign`. Its constructor takes as arguments variable bounds, the size of the generated sample, and an optional boolean parameter to tell whether randomized (default) or centered LHS have to be generated.

The `LHSResult` class is returned by `OptimalLHS.generate()` and contains algorithm results. The `add` method is called by `OptimalLHS` (once if there is no restart, and one plus one by restart otherwise), and informations can then be extracted by accessors. If no restart argument is specified, informations about global optimum are retrieved. If a restart number is provided, informations about this specific run are retrieved.

2   Reference Guide

The OTLHS library provide algorithms to compute optimized Latin Hypercube Sample designs.

2.1   Optimized LHS design

Mathematical description

Goal

Let $\underline{x}=(x_1,\cdots,x_d)$ be a random vector of input parameters. Latin Hypercube Sample (LHS) is a way to distribute N sample points: each parameter range is divided into N equal intervals, and sample points are chosen such that any hyperplane in that dimension contains one and only one sample point.

The goal of this module is to improve standard LHS techniques by minimizing a space filling criterion.

Principles

We may notice two types of LHS designs:

- Centered design is obtained by choosing for each point the center of the corresponding cell

- Randomized LHS is obtained by adding random perturbations inside each cell

Let us fix the following properties for the input vector $\underline{x}$:

- Its marginals are independent

- Its associated probabilistic measure is

$$\mathcal{L}_X(x_1,...,x_d) = \mathcal{U}(a_1,b_1) \otimes \mathcal{U}(a_2,b_2) \otimes \cdots \otimes \mathcal{U}(a_d,b_d) \tag{1}$$

   with $\mathcal{U}$ the uniform distribution.

   In practice, we look for a design in the space $[0,1]^d$ and we use an inverse iso-probabilistic transformation to get the result in the original domain.

Let $\phi : [0,1]^d \to \mathbb{R}^+$ be a a space filling criterion, which is a measure of "accuracy" of an optimal LHS design. Most of these criteria focus on discrepancy, which measures how far a given distribution of points deviates from a perfectly uniform one.
Two space filling criteria are implemented:

- The centered L^2-discrepancy, called C_2 and given by:

$$C_2(X_d^N)^2 = \left(\frac{13}{12}\right)^d - \frac{2}{N}\sum_{i=1}^{N}\prod_{k=1}^{d}\left(1 + \frac{1}{2}|x_k^{(i)} - 0.5| - \frac{1}{2}|x_k^{(i)} - 0.5|^2\right)$$
$$+ \frac{1}{N^2}\sum_{i,j=1}^{N}\prod_{k=1}^{d}\left(1 + \frac{1}{2}|x_k^{(i)} - 0.5| + \frac{1}{2}|x_k^{(j)} - 0.5| - \frac{1}{2}|x_k^{(i)} - x_k^{(j)}|\right) \tag{2}$$

   This discrepancy is to be minimized to get an optimal design.

- The mindist criterion (minimal distance between two points in the design):

$$\phi(X) = min||x^{(i)} - x^{(j)}||_{L^2}, \forall i \neq j = 1, \cdots N \tag{3}$$

This criterion is to be maximized.

- In practice, the ϕ_p criterion is used instead of mindist and writes:

$$\phi_p(X) = \left(\sum_{1 \leq i < j \leq N} ||x^{(i)} - x^{(j)}||_{L^2}^{-p} \right)^{\frac{1}{p}} \tag{4}$$

This is supposed to be more robust. When p tends to infinity, optimizing a design with ϕ_p is equivalent to optimizing a design with ¡¡mindist¿¿. This criterion is to be minimized to get an optimal design.

The objective is to a LHS design X_d^N that minimizes a space filling criterion ϕ (or maximizes mindist). For that purpose, two techniques are implemented and presented hereafter.

Monte Carlo
This problem can be approximated by a Monte Carlo algorithm: a fixed number of designs are generated, and the optimal one is kept. This algorithm is trivial, and described in Algorithm 1.
One of the major drawbacks of Monte Carlo sampling is the CPU time consumption, because the number of generated designs must be high.
Simulated Annealing
An alternate solution is to use an adapted simulated annealing method, which we will now describe.
Starting from an LHS design, a new design is obtained by permuting one random coordinate of two randomly chosen elements; by construction, this design is also an LHS design. If the new design is better than the previous one, it is kept. If it is worse, it may anyway be kept with some probability, which depends on how these designs compare, but also on a temperature profile T which decreases over time. This means that jumping away from local extrema becomes less probable over time. This algorithm is detailed in Algorithm 2.
It is important to highlight here that this specific permutation has been chosen in this algorithm because it allows highly efficient computations of criterion during simulated annealing process. The naive criterion evaluation, as is done in Monte Carlo algorithm, has a complexity of $\mathcal{O}(d \times N^2)$ for C_2 and ϕ_p criteria.
Let us first illustrate with the C_2 criterion. We set $z_{ik} = x_{ik} - 0.5$, equation (2) rewrites:

$$C_2(X_d^N)^2 = \left(\frac{13}{12} \right)^d + \sum_{i=1}^{N} \sum_{j=1}^{N} c_{ij}$$

with:

$$c_{ij} = \begin{cases} \dfrac{1}{N^2} \displaystyle\prod_{k=1}^{d} \dfrac{1}{2}(2 + |z_{ik}| + |z_{jk}| - |z_{ik} - z_{jk}|) & \text{if } i \neq j \\ \dfrac{1}{N^2} \displaystyle\prod_{k=1}^{d}(1 + |z_{ik}|) - \dfrac{2}{N} \displaystyle\prod_{k=1}^{d}(1 + \dfrac{1}{2}|z_{ik}| - \dfrac{1}{2}z_{ik}^2) & \text{otherwise} \end{cases} \tag{5}$$

We set c' the elements of a new design $X_d'^N$ obtained by permuting a coordinate of sample points i_1 and i_2. We can see that

$$\begin{cases} c'_{ij} = c_{ij} \ \forall i,j \text{ such that } 1 \leq i,j \leq N, \ i \notin \{i_1, i_2\}, \ j \notin \{i_1, i_2\} \\ c'_{i_1 i_2} = c_{i_1 i_2} \\ c_{ij} = c_{ji} \ \forall 1 \leq i,j \leq N \end{cases} \tag{6}$$

and thus, $C_2(X')$ becomes:

$$C_2(X_d'^N)^2 = C_2(X_d^N)^2 + c'_{i_1i_1} + c'_{i_2i_2} + 2 \sum_{\substack{1 \le j \le N \\ j \ne i_1,i_2}} (c'_{i_1j} + c'_{i_2j}) - c_{i_1i_1} - c_{i_2i_2} - 2 \sum_{\substack{1 \le j \le N \\ j \ne i_1,i_2}} (c_{i_1j} + c_{i_2j})$$

Updating C_2 criterion can be performed by a $\mathcal{O}(N)$ algorithm, which has a much better complexity than a naive computation.

The same trick can also be applied on ϕ_p criterion, because we can write

$$\phi_p(X)^p = \sum_{1 \le i < j \le N} ||x^{(i)} - x^{(j)}||_{L^2}^{-p} = \frac{1}{2} \sum_{i=1}^{N} \sum_{\substack{1 \le j \le N \\ j \ne i}} ||x^{(i)} - x^{(j)}||_{L^2}^{-p} = \sum_{i=1}^{N} \sum_{j=1}^{N} f_{ij} \tag{7}$$

with

$$f_{ij} = \begin{cases} \dfrac{||x^{(i)} - x^{(j)}||_{L^2}^{-p}}{2}, & i \ne j \\ 0, & i = j \end{cases} \tag{8}$$

These f_{ij} coefficients satisfy the same conditions as in (6), so the same computations give:

$$\phi_p(X_d'^N)^p = \phi_p(X_d^N)^p + 2 \sum_{\substack{1 \le j \le N \\ j \ne i_1,i_2}} (f'_{i_1j} + f'_{i_2j}) - 2 \sum_{\substack{1 \le j \le N \\ j \ne i_1,i_2}} (f_{i_1j} + f_{i_2j})$$

Remark In practice, a marginal transformation is performed to map the initial multivariate distribution into $[0,1]^d$. Optimization is performed in $[0,1]^d$ and the inverse transformation maps the design into the initial space.

Other notations

Link with OpenTURNS methodology

This method is part of the step C ¡¡Propagation of uncertainty¿¿ of the global methodology to evaluate a criterion for the output value defined in step A ¡¡Specifying the Criteria and the Case Study¿¿. It requires the specification of the joined probability density function of the input variables. The PDF must have an independent copula.

References and theoretical basics

G. Damblin, M. Couplet and B. Iooss. *Numerical studies of space filling designs: optimization of Latin hypercube samples and subprojection properties.* Journal of Simulation, 7:276-289, 2013.
K-T. Fang, R. Li, and A. Sudjianto. *Design and modeling for computer experiments.* Chapman & Hall/ CRC, 2006.
R. Jin, W. Chen, and A. Sudjianto. *An efficient algorithm for constructing optimal design of computer*

experiments. Journal of Statistical Planning and Inference, 134 :268-287, 2005.

J.R. Koehler and A.B. Owen. *Computer experiments. In S. Ghosh and C.R. Rao, editors, Design and analysis of experiments, volume 13 of Handbook of statistics.* Elsevier, 1996.

Johnson M, Moore L and Ylvisaker D (1990). *Minimax and maximin distance design.* Journal of Statistical Planning and Inference 26(2): 131-148.

McKay M, Beckman R and Conover W (1979). *A comparison of three methods for selecting values of input variables in the analysis of output from a computer code.* Technometrics 21(2): 239-245.

D. Morris and J. Mitchell. *Exploratory designs for computationnal experiments.* Journal of Statistical Planning and Inference, 43 :381-402, 1995.

Pronzato L and Müller W (2012). *Design of computer experiments: Space filling and beyond.* Statistics and Computing 22(3): 681-701.

input: bounds: the bounds of uniform distributions
    N: design size
    isCentered: boolean (centered if true, random otherwise)
    ϕ: space filling criterion
    MC: number of simulations
output: LHS: an optimal design
    phi: value of criterion

phi.opt ¡- 1e308;
for $i \leftarrow 1$ **to** MC **do**
    lhs ¡- LHSGenerate(bounds,N,isCentered);
    phi ¡- ϕ (lhs);
    if $phi < phi.opt$ **then**
        lhs.opt ¡- lhs;
        phi.opt¡-phi;
    end
end

Algorithm 1: Monte Carlo optimization of LHS design

input: bounds: the bounds of uniform distributions
        N: design size
        isCentered: boolean (centered if true, random otherwise)
        ϕ: space filling criterion
        nrIter: number of iterations
        T: temperature profile
output: LHS: an optimized design
        phi: value of criterion

lhs.opt ¡- LHSGenerate(bounds, N, isCentered) ;
phi.opt ¡- ϕ (lhs.opt);
i ¡- 1;
while *i ¡ nrIter* **do**
    point1, point2, dim ¡- `RandomGenerate` (3);
    LHSnew ¡-lhs.opt;
    swap(LHSnew[point1, dim] , lhs.opt[point2, dim]);
    phi ¡- ϕ (LHSnew);
    p ¡- $\min(\exp(-\frac{\text{phi}-\text{phi.opt}}{T(i)}), 1)$;
    if *p==1* **then**
        lhs.opt ¡- LHSnew;
        phi.opt¡-phi;
    else
        dist ¡- Bernoulli(p);
        b ¡- dist.generate();
        if *b==1* **then**
            lhs.opt ¡- LHSnew;
            phi.opt¡-phi;
        end
    i ¡- i + 1
end

Algorithm 2: LHS optimization using Simulated Annealing

3 Use Cases Guide

This section presents the main functionalities of the module *otlhs* in their context.

3.1 Latin Hypercube Sample

Class `LHSDesign` builds designs of a fixed size, type and bounds. Comparing to the `LHSExperiment` of the `OpenTURNS` library, when regenerating sample, the cells selection changes.

Python script for this use case:

```python
import openturns as ot
import otlhs
# Generating a design of size N=100
N = 100
# Considering independent Uniform distributions of dimension 3
# Bounds are (-1,1), (0,2) and (0, 0.5)
bounds = ot.Interval([-1,0,0], [1,2,0.5])
# Random LHS
lhs_random = otlhs.LHSDesign(bounds, N)
randomLHS = lhs_random.generate()
# Centered LHS
lhs_centered = otlhs.LHSDesign(bounds, N, True)
centeredLHS = lhs_centered.generate()
print(centeredLHS)
```

3.2 Plot of LHS

Class `PlotDesign` helps to plot a LHS design and draw it in a PDF, PNG, EPS or FIG formats.

Python script for this use case:

```python
import openturns as ot
import otlhs
# Generating a design of size N=10
N = 10
# Considering independent Uniform distributions of dimension 3
# Bounds are (-1,1), (0,2) and (0, 0.5)
bounds = ot.Interval([-1,0,0], [1,2,0.5])
# Centered LHS
lhs_centered = otlhs.LHSDesign(bounds, N, True)
centeredLHS = lhs_centered.generate()
# Draw the design + grids
drawable = otlhs.PlotDesign(centeredLHS, bounds, 10, 10)
# This is equivalent to:
# drawable = otlhs.PlotDesign(centeredLHS, bounds)
# Create ot graph
graph = ot.Graph()
```

```python
# Add drawable
graph.add(drawable)
# Export into PDF file. Refer to Graph class
graph.draw("design", 640, 480, ot.GraphImplementation.PDF)
```

Notice that there is no ¡¡show method¿¿ for that class. An alternative is to use the `PyPlotDesign` class. This requires the `matplotlib` module.

Python script for this use case:

```python
import openturns as ot
import otlhs
from otlhs.pyplotdesign import PyPlotDesign
import matplotlib.pylab as plt
#  Generating a design of size N=10
N = 10
# Considering independent Uniform distributions of dimension 3
# Bounds are (-1,1), (0,2) and (0, 0.5)
bounds = ot.Interval([-1,0,0], [1,2,0.5])
# Centered LHS
lhs_centered = otlhs.LHSDesign(bounds, N, True)
centeredLHS = lhs_centered.generate()
# Draw design and grid
plot = PyPlotDesign(centeredLHS, bounds, 10, 10)
# Show the graph
plt.show()
# Export the graph
plot.savefig("design.png")
```

3.3 LHS and space filling

`SpaceFilling` is an abstract class to compute the criterion value of a given design. `SpaceFillingC2` and `SpaceFillingPhiP` are implemented here. Note that these classes support only the computation of the criterion, and not its optimization.

Python script for this use case:

```python
import openturns as ot
import otlhs
#  Generating a design of size N=100
N = 100
# Considering independent Uniform distributions of dimension 3
# Bounds are (-1,1), (0,2) and (0, 0.5)
bounds = ot.Interval([-1,0,0], [1,2,0.5])
# Random LHS
lhs = otlhs.LHSDesign(bounds, N)
design = lhs.generate()
# C2
c2 = otlhs.SpaceFillingC2().evaluate(design)
```

```
# PhiP with default p
phip = otlhs.SpaceFillingPhiP().evaluate(design)
# mindist
mindist = otlhs.SpaceFillingMinDist().evaluate(design)
# For p->infinity
phip_inf = otlhs.SpaceFillingPhiP(100).evaluate(design)
```

3.4 Optimized LHS using Monte Carlo

Given a space filling criterion, it is possible to generate many designs and to return the "best" one. This method is trivial to implement but the number of generations may become impractical.

The class `MonteCarloLHS` gives the optimal design using the `LHSDesign` as factory and number of simulations. Default space filling criterion is `SpaceFillingMinDist`.
Python script for this use case:

```python
import openturns as ot
import otlhs

# Considering independent Uniform(0,1) distributions of dimension 3
bounds = ot.Interval(3)
designSize = 100
# Random LHS MonteCarlo
lhs = otlhs.LHSDesign(bounds, designSize)
# MonteCarlo, with nSimu=50000 and default space_filling
nSimu = 50000
algo = otlhs.MonteCarloLHS(lhs, nSimu)
result = algo.generate()
# optimal design
design = result.getOptimalDesign()
```

Results may be completed by the fixed criterion value history for all samples, the final criteria values.
Python script for this use case:

```python
import openturns as ot
import otlhs
# Generating a design of size N=100
N = 100
# Considering independent Uniform distributions of dimension 3
# Bounds are (-1,1), (0,2) and (0, 0.5)
bounds = ot.Interval([-1,0,0], [1,2,0.5])
# Random LHS MonteCarlo
lhs = otlhs.LHSDesign(bounds, N)
# Fixing C2 crit
space_filling = otlhs.SpaceFillingC2()
# Defining MonteCarlo, with nSimu=50000
nSimu = 50000
algo = otlhs.MonteCarloLHS(lhs, nSimu, space_filling)
result = algo.generate()
```

```
# optimal design
design = result.getOptimalDesign()
history = result.getAlgoHistory()
# Final criteria values
c2 = result.getC2()
phip = result.getPhiP()
mindist = result.getMinDist()
```

3.5   Optimized LHS using simulated annealing

As with Monte Carlo, user decides of a fixed number of iterations, but this time this number is part of the temperature profile. Two profiles are currently provided:

- Linear profile: $T(i) = T(0)\left(1 - \frac{i}{nrIter}\right)$

- Geometric profile: $T(i) = T(0)c^i$, $0 < c < 1$.

Starting from an LHS design, a new design is built by permuting a random coordinate of two randomly chosen sample points; this new design is also an LHS. but not necessary a "more efficient" design. A comparison of criteria of the two designs is done, and the new LHS is accepted with probability

$$min\left(exp\left[-\frac{\phi(\text{LHSnew}) - \phi(\text{LHS})}{T(i)}\right], 1\right)$$

The simulated annealing algorithm is detailed in Algorithm 2.

Minimal python script for the use case:

```
import openturns as ot
import otlhs
designSize = 100
# Considering independent Uniform(0,1) distributions of dimension 3
bounds = ot.Interval(3)
# Random LHS
lhs = otlhs.LHSDesign(bounds, N)
algo = otlhs.SimulatedAnnealingLHS(lhs)
result = algo.generate()
# Retrieve optimal design
design = result.getOptimalDesign()
```

One could also fix the criterion, the temperature profile and gets more results.
Python script for this use case:

```
import openturns as ot
import otlhs
#  Generating a design of size N=100
N = 100
# Considering independent Uniform distributions of dimension 3
# Bounds are (−1,1), (0,2) and (0, 0.5)
bounds = ot.Interval([−1,0,0], [1,2,0.5])
# Random LHS
```

```
lhs = otlhs.LHSDesign(bounds, N)
# Fixing C2 crit
space_filling = otlhs.SpaceFillingC2()
# Defining a temperature profile
# A geometric profile seems accurate with default parameters
# e.g. T0=10, c=0.95, iMax=2000
temperatureProfile = otlhs.GeometricProfile()
algo = otlhs.SimulatedAnnealingLHS(lhs, temperatureProfile, space_filling)
result = algo.generate()
# Retrieve optimal design
design = result.getOptimalDesign()
# Criteria for the optimal design
crit_c2 = result.getC2()
crit_phip = result.getPhiP()
crit_mindist = result.getMinDist()
# History of the criterion used for optimization
history = result.getAlgoHistory()
criterion_hist = history[:, 0]
# Additional results
temperature_hist = history[:, 1]
probability_hist = history[:, 2]
```

It is also possible to chain several iterations of the whole process with different starting points. Python script for this use case:

```
import openturns as ot
import otlhs
#  Generating a design of size N=100
N = 100
# Considering independent Uniform distributions of dimension 3
# Bounds are (−1,1), (0,2) and (0, 0.5)
bounds = ot.Interval([−1,0,0], [1,2,0.5])
# Random LHS
lhs = otlhs.LHSDesign(bounds, N)
# Fixing PhiP crit
space_filling = otlhs.SpaceFillingPhiP()
# Defining a temperature profile
# T0=10, iMax=3000
temperatureProfile = otlhs.LinearProfile(10.0, 3000)
algo = otlhs.SimulatedAnnealingLHS(lhs, temperatureProfile, space_filling)
restart = 50
result = algo.generate(restart)
# Retrieve optimal design
design = result.getOptimalDesign()
# Retrieve all optimal designs
designs = [result.getOptimalDesign(i) for i in xrange(restart)]
```

Finally, we could start the optimization process of LHS using a precomputed LHS design. Python script for this use case:

```python
import openturns as ot
import otlhs
# Generating a design of size N=100
N = 100
# Considering independent Uniform distributions of dimension 3
# Bounds are (0,1)^3
bounds = ot.Interval(3)
# Random LHS
lhs = otlhs.LHSDesign(bounds, N)
# Fixing C2 crit for example
space_filling = otlhs.SpaceFillingC2()
# Defining a temperature profile
# T0=10, iMax=3000
temperatureProfile = otlhs.LinearProfile(10.0, 3000)
algo = otlhs.SimulatedAnnealingLHS(lhs, temperatureProfile, space_filling)
result = algo.generate()
# optimal design
design = result.getOptimalDesign()
# check history ==> draw criterion
ot.Show( result.drawHistoryCriterion() )
# Convergence needs to be performed
# New algo starting from this design
algo = otlhs.SimulatedAnnealingLHS(design, bounds, temperatureProfile,
    space_filling)
result = algo.generate()
# New design
design = result.getOptimalDesign()
```

3.6   Ishigami test

The following example illustrates the generation of an optimal design for surrogate model learning purposes. The use case proposed here is the Ishigami function. The model requires 3 input parameters, which are known to be uniformly distributed in $[-\pi, \pi]^3$.

Python script for a detailed study is proposed hereafter:

```python
import openturns as ot
import otlhs
from math import sin, pi
import matplotlib.pylab as plt
from otlhs.pyplotdesign import PyPlotDesign

# Set seed
ot.RandomGenerator.SetSeed(0)
# Definition of the Ishigami function
dimension = 3
a = 7.0
```

```python
b = 0.1

# Create the Ishigami function
input_variables = ["xi1","xi2","xi3"]
formula = ["sin(xi1) + (" + str(a) + ") * (sin(xi2)) ^ 2 + (" + str(b) +
    ") * xi3^4 * sin(xi1)"]
ishigami_model = ot.NumericalMathFunction(input_variables, formula)
ishigami_model.setName("Ishigami")

#  Generating a design of size
N = 150
# Considering independent Uniform distributions of dimension 3
# Bounds are (-pi,pi), (-pi,pi) and (-pi,pi)
bounds = ot.Interval(dimension*[-pi], dimension*[pi])
# Random LHS
lhs = otlhs.LHSDesign(bounds, N)
# Fixing C2 crit
space_filling = otlhs.SpaceFillingC2()
# Defining a temperature profile
temperatureProfile = otlhs.GeometricProfile()
# Pre conditionning : generate an optimal design with MC
nSimu = 100
algo = otlhs.MonteCarloLHS(lhs, nSimu, space_filling)
result = algo.generate()
initialDesign = result.getOptimalDesign()
print("initial design pre-computed. Performing SA optimization...")
# Use of initial design
algo = otlhs.SimulatedAnnealingLHS(initialDesign, bounds,
    temperatureProfile, space_filling)
result = algo.generate()
print("initial design computed")
# Retrieve optimal design
input_database = result.getOptimalDesign()

fig = PyPlotDesign(input_database, bounds, 1, 1)
fig.set_size_inches(fig.get_size_inches() * 2)
plt.suptitle("Ishigami design")
plt.savefig("design_ishigami.png")
plt.close(fig)
# Response of the model
print "sampling size = ", N
output_database = ishigami_model(input_database)

# Learning input/output
# Usual chaos meta model
enumerate_function = ot.HyperbolicAnisotropicEnumerateFunction(dimension)
orthogonal_basis =
    ot.OrthogonalProductPolynomialFactory(dimension*[ot.LegendreFactory()],
```

```
        enumerate_function)
    basis_size = 100
    # Initial chaos algorithm
    adaptive_strategy = ot.FixedStrategy(orthogonal_basis, basis_size)
    # ProjectionStrategy ==> Sparse
    fitting_algorithm = ot.KFold()
    approximation_algorithm =
        ot.LeastSquaresMetaModelSelectionFactory(ot.LAR(), fitting_algorithm)
    projection_strategy = ot.LeastSquaresStrategy(input_database,
        output_database, approximation_algorithm)
    print("Surrogate model...")
    distribution_ishigami = ot.ComposedDistribution(dimension *
        [ot.Uniform(-pi, pi)])
    algo_pc = ot.FunctionalChaosAlgorithm(input_database, output_database,
        distribution_ishigami, adaptive_strategy, projection_strategy)
    algo_pc.run()
    chaos_result = algo_pc.getResult()
    print("Surrogate model computed")

    # Validation
    lhs_validation = ot.LHSExperiment(distribution_ishigami, 100)
    input_validation = lhs_validation.generate()
    output_validation = ishigami_model(input_validation)
    # Chaos model evaluation
    output_metamodel_sample = chaos_result.getMetaModel()(input_validation)

    # Cloud validation ==> change from matplotlib to pure OT
    fig = plt.figure()
    plt.plot(output_validation, output_validation, 'b-', label='Model')
    plt.plot(output_validation, output_metamodel_sample, 'r.', label='SM')
    plt.title("Surrogate Model validation - Ishigami use case")
    plt.savefig("ishigami_model_validation.png")
```

We illustrate hereafter the design obtained that optimize the C_2 criterion in figure 3.6.
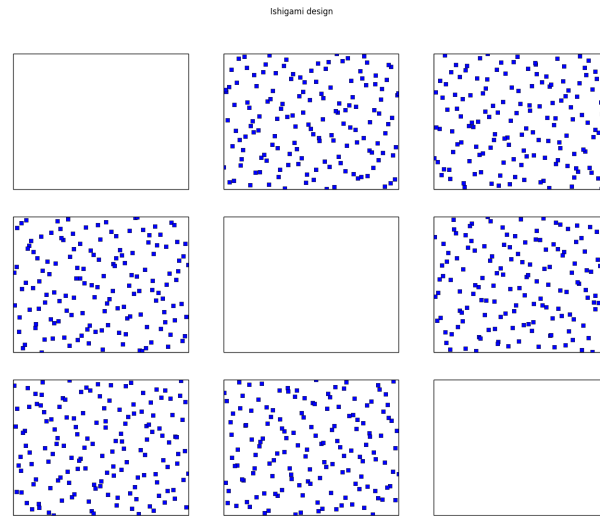
Figure 4: Input design for learning Ishigami model

The design seems accurate. To complete this example, the validation using an independent validation sample is done as illustrated in figure 3.6
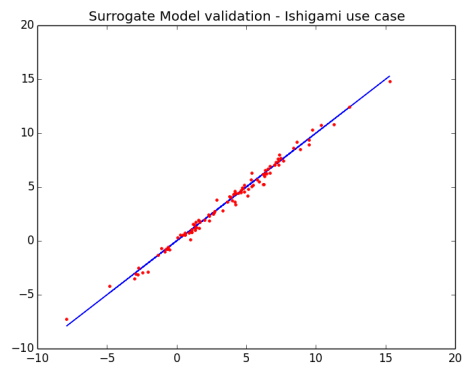

Figure 5: Validation of surrogate model

4 User Manual

This section gives an exhaustive presentation of the objects and functions provided by the *otlhs* module, in the alphabetic order.

4.1 GeometricProfile

Usage:

> *GeometricProfile()*
>
> *GeometricProfile(T0)*
>
> *GeometricProfile(T0,c)*
>
> *GeometricProfile(T0,c,iMax)*

Arguments:

> *T0*: a NumericalScalar, initial temperature (10 by default)
>
> *c*: a NumericalScalar, geometric factor, must be between 0 and 1 exclusive (0.95 by default)
>
> *iMax*: an UnsignedInteger, number of iterations (2000 by default)

Value : a GeometricProfile

Details :

> GeometricProfile constructor

operator()

> **Usage :** *operator()(it)*
>
> **Arguments :** an UnsignedInteger, iteration number
>
> **Value :** a NumericalScalar, the value of temperature at the given time

4.2 LHSDesign

Usage:

> *LHSDesign(bounds, size, randomDesign)*

Arguments:

> *bounds*: an Interval, bounds of variables
>
> *size*: an UnsignedInteger, number of sample points
>
> *randomDesign*: a Bool, `generate()` method builds a random LHS when true (which is the default), otherwise a centered LHS

Value : an LHSDesign

Details :

> LHSDesign constructor

getBounds

Usage : *getBounds()*

Value : an Interval, which are bounds of variables

getSize

Usage : *getSize()*

Value : an UnsignedInteger, sample size

isCenteredDesign

Usage : *isCenteredDesign()*

Value : a Bool, true if centered designs are generated, false otherwise

generate

Usage : *generate()*

Value : a NumericalSample, new design

4.3 LHSResult

Usage:

LHSResult()

LHSResult(bounds, spaceFilling)

LHSResult(bounds, spaceFilling, restart)

Arguments:

bounds: an Interval, specify bounds for each marginal

spaceFilling: a SpaceFilling, space filling criterion

restart: number of restarts performed during optimization

Value : an LHSResult

Details :

LHSResult constructor, which is only called by optimization algorithms

getBounds

Usage : *getBounds()*

Value : an Interval, which contains bounds for each marginal

getNumberOfRestarts

Usage : *getNumberOfRestarts()*

Value : an UnsignedInteger, number of restarts performed during optimization

getOptimalDesign

Usage : *getOptimalDesign()*

Value : a NumericalSample, contains the best LHS design

getOptimalDesign

> **Usage :** *getOptimalDesign(restart)*
>
> **Arguments :** an UnsignedInteger, to specify a restart number
>
> **Value :** a NumericalSample, contains the best LHS design found during this specific restart

getOptimalValue

> **Usage :** *getOptimalValue()*
>
> **Value :** a NumericalScalar, the criterion value for the optimal LHS design

getOptimalValue

> **Usage :** *getOptimalValue(restart)*
>
> **Arguments :** an UnsignedInteger, to specify a restart number
>
> **Value :** a NumericalScalar, the criterion value for the optimal LHS design found during this specific restart

getAlgoHistory

> **Usage :** *getAlgoHistory()*
>
> **Value :** a NumericalSample, contains informations gathered by optimization algorithm

getAlgoHistory

> **Usage :** *getAlgoHistory(restart)*
>
> **Arguments :** an UnsignedInteger, to specify a restart number
>
> **Value :** a NumericalSample, contains informations gathered by optimization algorithm during this specific restart

getC2

> **Usage :** *getC2()*
>
> **Value :** a NumericalScalar, the C2 value for the optimal LHS design

getC2

> **Usage :** *getC2(restart)*
>
> **Arguments :** an UnsignedInteger, to specify a restart number
>
> **Value :** a NumericalScalar, the C2 value for the optimal LHS design found during this specific restart

getPhiP

> **Usage :** *getPhiP()*
>
> **Value :** a NumericalScalar, the PhiP value for the optimal LHS design

getPhiP

> **Usage :** *getPhiP(restart)*
>
> **Arguments :** an UnsignedInteger, to specify a restart number

Value : a NumericalScalar, the PhiP value for the optimal LHS design found during this specific restart

getMinDist

    Usage : *getMinDist()*

    Value : a NumericalScalar, the mindist value for the optimal LHS design

getMinDist

    Usage : *getMinDist(restart)*

    Arguments : an UnsignedInteger, to specify a restart number

    Value : a NumericalScalar, the mindist value for the optimal LHS design found during this specific restart

4.4 LinearProfile

Usage:

    LinearProfile()
    LinearProfile(T0)
    LinearProfile(T0,iMax)

Arguments:

    T0: a NumericalScalar, initial temperature (10 by default)

    iMax: an UnsignedInteger, number of iterations (2000 by default)

Value : a LinearProfile

Details :

    LinearProfile constructor

operator()

    Usage : *operator()(it)*

    Arguments : an UnsignedInteger, iteration number

    Value : a NumericalScalar, the value of temperature at the given time

4.5 MonteCarloLHS

Usage:

    MonteCarloLHS(LHS, N)
    MonteCarloLHS(LHS, N, spaceFilling)

Arguments:

    LHS: an LHS, initial design

    N: an UnsignedInteger, number of random designs

    spaceFilling: a SpaceFilling, space filling criterion

Value : a MonteCarloLHS

Details :

> With the first usage, `spaceFilling` is fixed to `SpaceFillingMinDist`

generate

> **Usage :** *generate()*
>
> **Value :** an LHSResult, this instance contains the best design and some other values

4.6 OptimalLHS

Usage:

> *OptimalLHS(lhs)*
>
> *OptimalLHS(lhs, spaceFilling)*

Arguments:

> *lhs*: an LHS, initial design
>
> *spaceFilling*: a SpaceFilling, space filling criterion

Value : an OptimalLHS

Details :

> OptimalLHS constructor

getLHS

> **Usage :** *getLHS()*
>
> **Value :** an LHS, initial design

getSpaceFilling

> **Usage :** *getSpaceFilling()*
>
> **Value :** a SpaceFilling, space filling criterion

generate

> **Usage :** *generate()*
>
> **Value :** an LHSResult, this instance contains the best design and some other values

4.7 PlotDesign

Usage:

> *PlotDesign(data, bounds)*
>
> *PlotDesign(data, bounds, Nx, Ny)*
>
> *PlotDesign(data, bounds, Nx, Ny, title)*
>
> *PlotDesign(result)*

PlotDesign(result, Nx, Ny)

PlotDesign(result, Nx, Ny, title)

Arguments:

data: a NumericalSample, which represents an LHS design

bounds: an Interval, specify bounds for each marginal

Nx: an UnsignedInteger, number of grid intervals along X-axis (default value = size of design)

Ny: an UnsignedInteger, number of grid intervals along Y-axis (default value = size of design)

title: a String, plot title

result: an LHSResult, returned by an optimization algorithm

Value : a PlotDesign

Details :

PlotDesign constructor

4.8 SimulatedAnnealingLHS

Usage:

SimulatedAnnealingLHS(lhs)

SimulatedAnnealingLHS(lhs, profile)

SimulatedAnnealingLHS(lhs, profile, spaceFilling)

SimulatedAnnealingLHS(design, bounds)

SimulatedAnnealingLHS(design, bounds, profile)

SimulatedAnnealingLHS(design, bounds, profile, spaceFilling)

Arguments:

lhs: an LHS, initial design factory

design: a NumericalSample, an initial design

bounds: an Interval, bounds of design

profile: a TemperatureProfile, temperature profile

spaceFilling: a SpaceFilling, space filling criterion

Value : a SimulatedAnnealingLHS

Details :

With the first and fourth usages, profile and spaceFilling are respectively setted to default `GeometricProfile` and `SpaceFillingMinDist`

With the second and fifth usages, spaceFilling is setted to `SpaceFillingMinDist`

generate

Usage : *generate()*

Value : an LHSResult, this instance contains the best design and some other values

generate

Usage : *generate(restart)*

Arguments : an UnsignedInteger, the number of restarts

Value : an LHSResult, this instance contains the best design and some other values

4.9 SpaceFilling

Usage:

SpaceFilling(impl)

Arguments:

impl: a SpaceFillingImplementation, a specific implementation of SpaceFilling

Value : a SpaceFilling

Details :

SpaceFilling constructor

evaluate

Usage : *evaluate(sample)*

Arguments : a NumericalSample

Value : a NumericalScalar, value of the criterion evaluated on this sample

4.10 SpaceFillingC2

Usage:

SpaceFillingC2()

Value : a SpaceFillingC2

Details :

SpaceFillingC2 constructor

evaluate

Usage : *evaluate(sample)*

Arguments : a NumericalSample

Value : a NumericalScalar, value of the C2 criterion evaluated on this sample

4.11   SpaceFillingMinDist

Usage:

>   *SpaceFillingMinDist()*

Value : a SpaceFillingMinDist

Details :

>   SpaceFillingMinDist constructor

evaluate

>   **Usage :** *evaluate(sample)*
>
>   **Arguments :** a NumericalSample
>
>   **Value :** a NumericalScalar, value of the mindist criterion evaluated on this sample

4.12   SpaceFillingPhiP

Usage:

>   *SpaceFillingPhiP(p)*

Arguments:

> p: an UnsignedInteger (by default, 50)

Value : a SpaceFillingPhiP

Details :

>   SpaceFillingPhiP constructor

evaluate

>   **Usage :** *evaluate(sample)*
>
>   **Arguments :** a NumericalSample
>
>   **Value :** a NumericalScalar, value of the PhiP evaluated on this sample

5 Validation

This section aims at exposing the methodology used to validate numerical results of the module.

5.1 Methodology of validation

This module implements two different algorithms:

- Monte Carlo

- Simulated annealing algorithm; this algorithm is the novelty and requires several steps. It uses also some tips such as the update of criterion after a permutation instead of a complete calculation.

Both algorithms are to be validated.

5.2 Preliminary validations

For specific designs, criteria values (C_2, *mindist* and ϕ_p) obtained with *otlhs* module are compared with values computed by the `DiceDesign` R package. Those scripts are located in the `validation` folder of this module. Comparisons are very good, absolute error is less than 10^{-13}.

As mentionned previously, C_2 criterion can be computed efficiently when a small perturbation is performed on design. This specific method is compared to the `DiceDesign`'s ones: absolute error is less or equal to 10^{-10}.

Note that for ϕ_p criterion, `DiceDesign` computes the new value after a permutation without taking into account the oldest criterion. In this module, criterion update has been implemented, but is used only when parameter $p \geq 5$. Indeed, numerical experiments have shown instability of the update when p becomes large.

5.3 Validation of Monte Carlo algorithm

To get an optimal design using Monte Carlo algorithm, several designs are to be generated and the returned one minimizes the space filling function.

As the evaluation of the criterion does not require any complexity, validation of Monte Carlo algorithm is trivial:

- Fix a criterion to minimize;

- Fix the RandomSeed to a fixed value (0 for example);

- Generate $N = 100$ designs: get the optimal one and the associated criterion value;

- Fix again the RandomSeed;

- Generate $N = 10000$ designs: get the optimal one and its associated criterion value;

- Check that the last criterion value is less or equal than to the previous one;

5.4 Validation of simulated annealing algorithm

Simulated annealing is compared to Monte Carlo algorithm (with $N = 10000$) and should return better results. Indeed the optimal design is built such as the space filling criterion decreases at each iteration.

Several use cases are proposed for the validation and illustrated hereafter:
Final criteria should meet requirements.

Test id	Dimension	Size	Temperature profile	Profile parameters	Requirement
1	2	10	Geometric	$T_0 = 10, c = 0.999, iMax = 50000$	$C_2 \leq 0.0664$
2	2	10	Linear	$T_0 = 10, iMax = 50000$	mindist ≥ 0.272
3	50	100	Geometric	$T_0 = 10, c = 0.999, iMax = 50000$	$C_2 \leq 22.176$
4	50	100	Geometric	$T_0 = 10, c = 0.999, iMax = 50000$	mindist ≥ 2.653

Table 1: Use cases for the validation & requirements

5.5 Results

Designs are generated according to the multivariate distribution $U[0,1]^d$.

5.5.1 MonteCarlo results

We first check that Monte Carlo scales linearly with respect to the number of simulations. Random generator seed is reinitialized to the same value when starting Monte Carlo algorithm, this is why criterion always decreases.
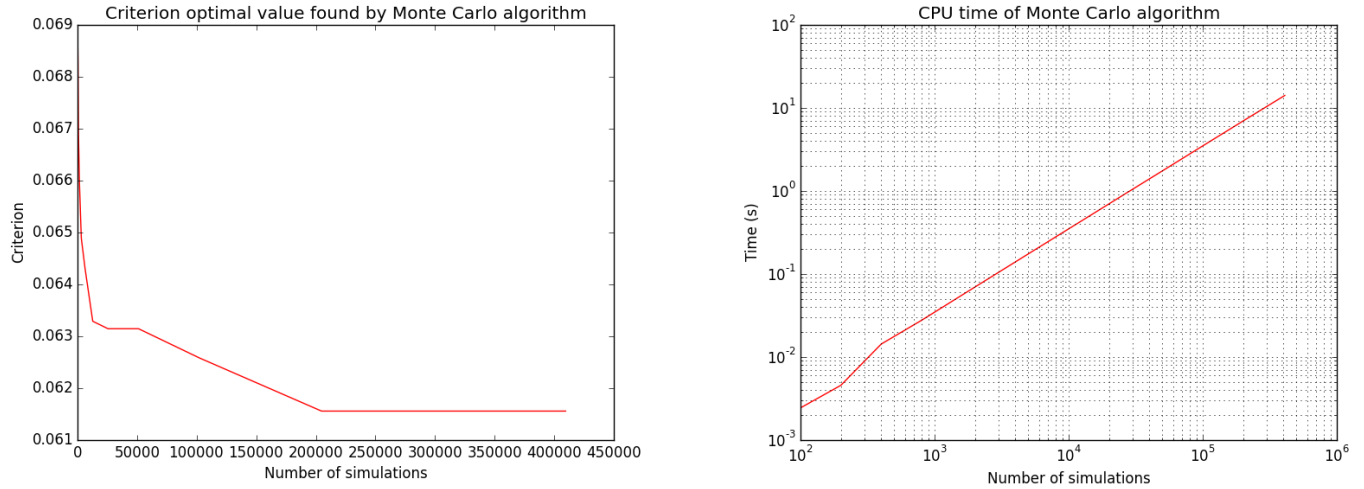

Figure 6: Scaling of Monte Carlo results when increasing number of simulations

Tests corresponding to use cases in table 1 are implemented and results obtained using Monte Carlo are given in table 2.

Dimension	Size	Criterion	Criterion value	CPU time (s)	Requirement
2	10	C_2	0.0643	0.72	$C_2 \leq 0.0664$
2	10	mindist	0.2666	0.47	mindist ≥ 0.272
50	100	C_2	24.427	109.48	$C_2 \leq 22.176$
50	100	mindist	2.198	53.36	mindist ≥ 2.653

Table 2: Monte Carlo results after $N_{simu} = 10000$ simulations

We use $N_{simu} = 10000$ simulations in order to get the optimal design (designs are not centered). As shown in

figure 6, 10000 iterations give a good solution for the small case; but it the larger case, it is expected that this number is way too small, so results are quite close to expectations.

In addition, we give design plots in dimension 2 only (in dimension 50, the interest is very limited).
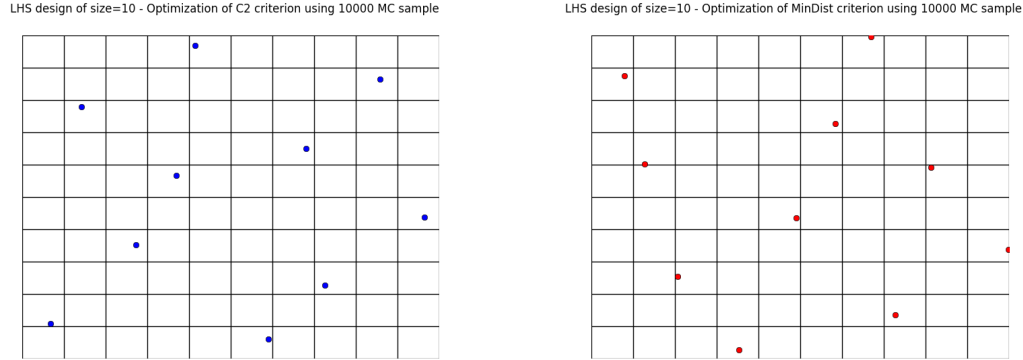

Figure 7: Monte Carlo results

5.5.2 Simulated annealing results

Using the `SimulatedAnnealingLHS` class and use cases described in table 1, tests are implemented and results are resumed in table 3. These are compared to those produced by `DiceDesign` R package in terms of criteria and CPU time.
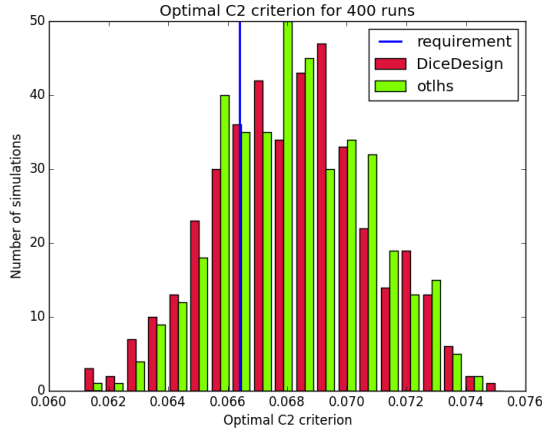
Test id	Requirement	otlhs		R	
		Criterion	CPU time (s)	Criterion	CPU time (s)
1	$C_2 \leq 0.0664$	0.0699	0.04	0.06153	89.8
2	mindist ≥ 0.272	0.254	0.246	0.258	36.37
3	$C_2 \leq 22.176$	22.190	2.69	22.15	618.7
4	mindist ≥ 2.653	2.643	55.8	2.64	220.6

Table 3: Numerical results obtained with the module

CPU time is much lower with *otlhs*. It must be noted that speedup of test 4 is not in par with speedups of other tests. We believe that this is not due to some performance problems, but is the combination of several factors:

1. R implementation of mindist is better than C2 because it does not contain loops, but only few high-level operations on matrices.

2. In *otlhs* implementations, mindist is slower than C2 because it calls `evaluate` instead of `perturbLHS`. It may be interesting to try with $p = 5$ instead of $p = 50$, mindist would then be as fast as C2, and many restarts could be tried. Unfortunately, we did not have time to make these tests.

Results are close to expectations, but do not meet all requirements. In order to understand why *otlhs* results are sometimes out of bounds, we performed 400 runs of tests 1 and 2 with DiceDesign and *otlhs*, 40 runs of test 3 and 80 runs of test 4. Results are shown in figure 8. Diagrams look similar, thus in our opinion, *otlhs*
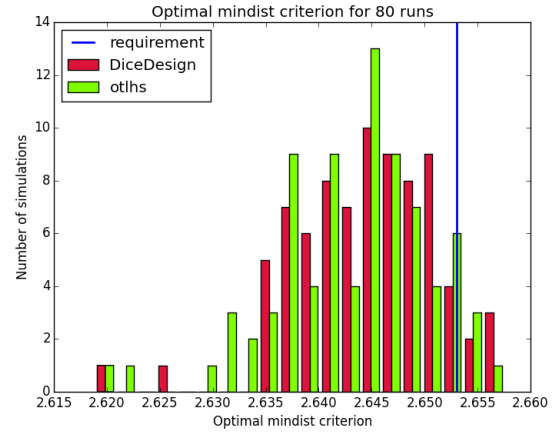
Comparison on 400 runs for test id 1
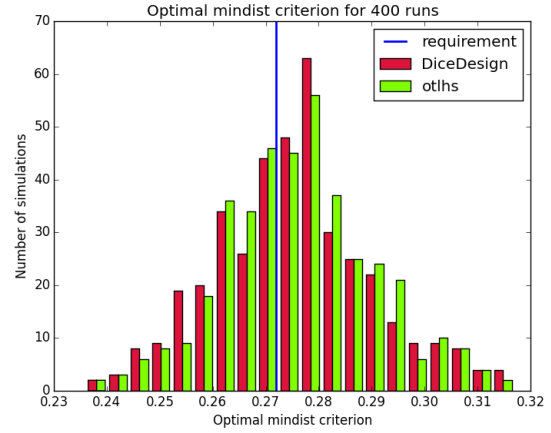
Comparison on 400 runs for test id 2
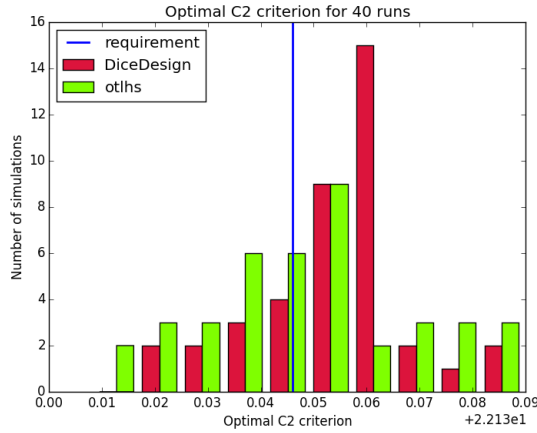
Comparison on 40 runs for test id 3

Comparison on 80 runs for test id 4

Figure 8: Results comparison with restarts

does meet requirements. Moreover, as *otlhs* is much faster than R, the same CPU budget will give better results with *otlhs*.

In addition, designs, optimized criterion convergence and elementary perturbation probability are given in figures 9, 10 and 11 (for dimension 50, only criterion history is displayed).

Results are very similar between the two implementations. It must be noted that there are many plots with probability 1. The reason is that DiceDesign accepts both row indices to be equal when checking for elementary perturbations.
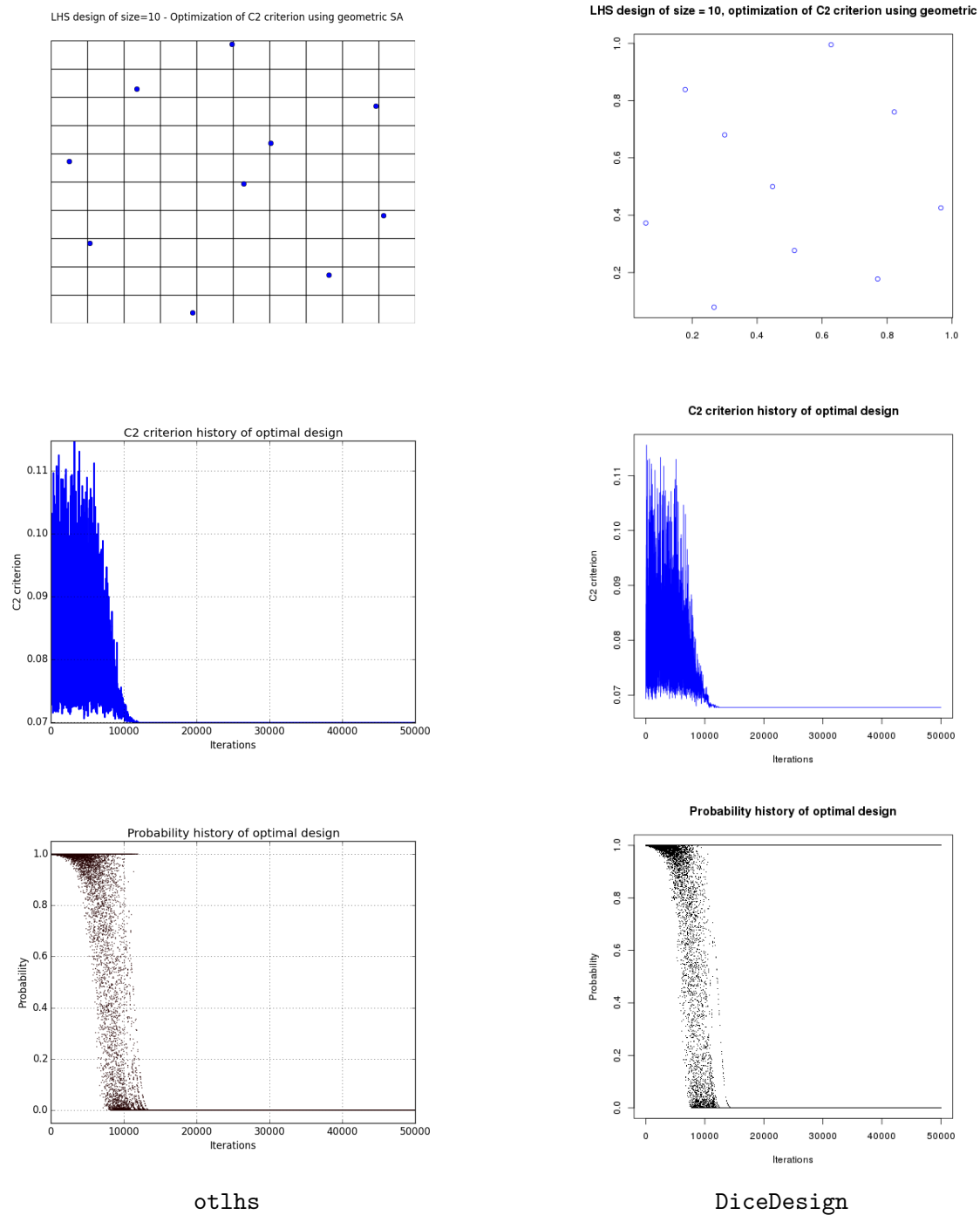
Figure 9: Simulated annealing results - Test id 1

LHS design of size=10 - Optimization of MinDist criterion using linear SA

LHS design of size = 10, optimization of MinDist criterion using linear

PhiP criterion history of optimal design

PhiP criterion history of optimal design

Probability history of optimal design

Probability history of optimal design

otlhs   DiceDesign
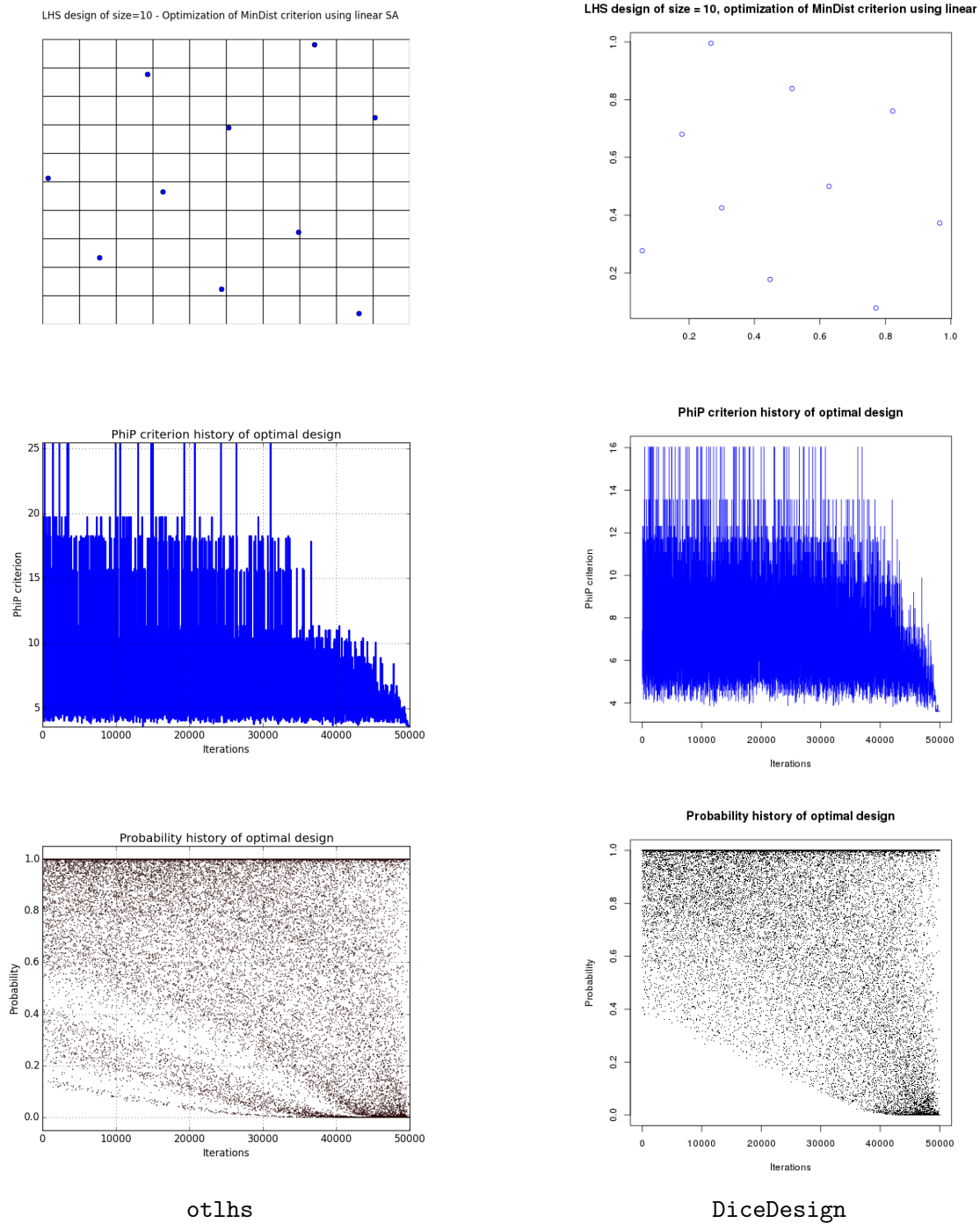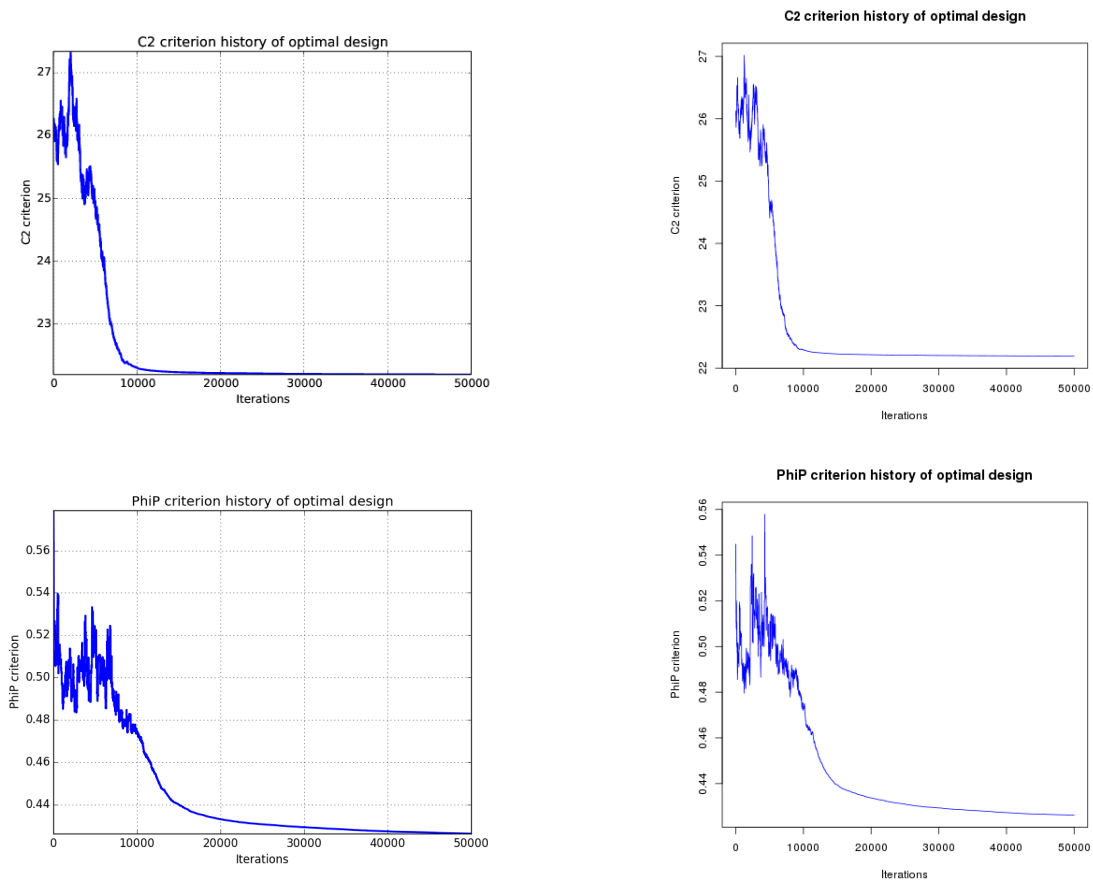
Figure 10: Simulated annealing results - Test id 2

Figure 11: Simulated annealing criterion results - Test id 3 and 4