Focus on adaptive simulation methods for reliability analysis with OpenTURNS

OpenTURNS Users Day

June 2025

L. Brevault (ONERA), M. Balesdent (ONERA)

Reliability analysis within OpenTURNS

Objective :

- Estimate the probability a limit state function $g: \mathbb{R}^d \to \mathbb{R}$ of crossing a threshold T considering a failure event $D_f = \{\boldsymbol{x} \in \mathbb{R}^d | g(\boldsymbol{x}) \leq T\}$ associated to an aleatory vector $\boldsymbol{X}$ defined by the joint PDF $f_{\boldsymbol{X}}$

$$P_f = \int_{\mathbb{R}^d} \mathbf{1}_{g(\boldsymbol{x}) \leq T} f_{\boldsymbol{X}}(\boldsymbol{x}) \mathrm{d}\boldsymbol{x}$$

Context :

- **Rare failure event** analysis

Focus on adaptive simulation methods for reliability analysis

Reliability analysis methods available in OpenTURNS

Reliability algorithms

- First Order Reliability Method (FORM)
- Second Order Reliability Method (SORM)
- And derivatives : MultiFORM, SystemFORM

Hybrid algorithms

- Post Analytical Importance Sampling

Simulation algorithms

- Monte Carlo sampling
- Directional Sampling
- Subset sampling
- Adaptive Directional Stratification
- Cross Entropy Importance Sampling
- Nonparametric Adaptive Importance Sampling

Unified way to handle reliability problems in OpenTURNS

Generic steps for performing reliability analysis :

1. Create input variable probability distribution
2. Create the limit state function
3. Create a threshold event class that gathers all the information required for reliability analysis
4. Select the reliability algorithm in OpenTURNS on the threshold event
   - Reliability algorithms : FORM, MultiFORM, SystemFORM, SORM, *etc*.
   - Simulation algorithms : Monte-Carlo experiment, Importance Sampling, Subset Simulation, Directional Stratification, *etc*.

OpenTURNS

An Open source initiative for the Treatment of Uncertainties, Risks'N Statistics

OpenTURNS 1.24 documentation » Contents » Examples » Reliability & Sensitivity » Reliability » Create a threshold event

Create a threshold event

Abstract

We present in this example the creation and the use of a `ThresholdEvent` to estimate a simple integral.

```
import openturns as ot
import openturns.viewer as otv
from matplotlib import pylab as plt
```

We consider a standard Gaussian random vector X and build a random vector from this distribution.

```
distX = ot.Normal()
vecX = ot.RandomVector(distX)
```

We consider the simple model $f : x \mapsto |x|$ and consider the output random variable $Y = f(X)$.

```
f = ot.SymbolicFunction(["x1"], ["abs(x1)"])
vecY = ot.CompositeRandomVector(f, vecX)
```

We define a very simple `ThresholdEvent` which happpens whenever $|X| < 1.0$:

```
thresholdEvent = ot.ThresholdEvent(vecY, ot.Less(), 1.0)
```

For the normal distribution, it is a well-known fact that the values lower than one standard deviation (here exactly 1) away from the mean (here 0) account roughly for 68.27% of the set. So the probability of the event is:

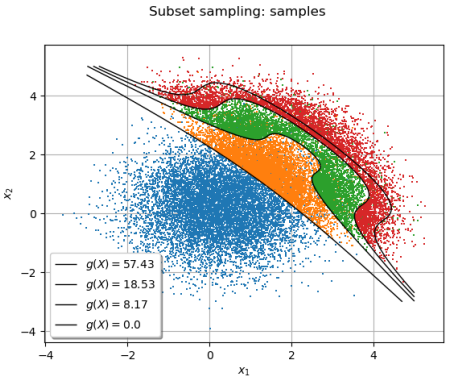
```
print("Probability of the event : %.4f" % 0.6827)
```

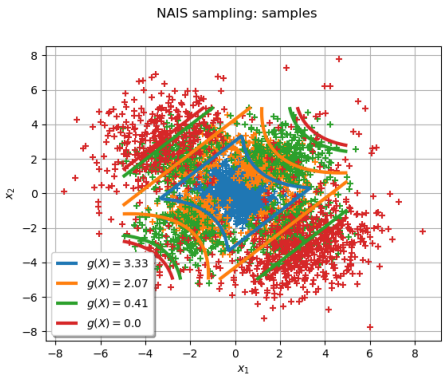
Out: `Probability of the event : 0.6827`

Adaptive simulation methods for reliability analysis

General principle :

- Define a sequential strategy in order to accuratly estimate the probability of failure and to reduce the variance of the estimator
- All methods are based on an *EventSimulation()*





Threshold probability: Simulation algorithms

Simulations methods

`SimulationAlgorithm`(*args)	Base class for simulation algorithms.
`EventSimulation`(*args)	Base class for sampling methods.
`ProbabilitySimulationAlgorithm`(*args)	Iterative sampling methods.
`DirectionalSampling`(*args)	Directional sampling algorithm.
`PostAnalyticalSimulation`(*args)	Post analytical simulation.
`PostAnalyticalControlledImportanceSampling`(*args)	Post analytical controlled importance sampling.
`PostAnalyticalImportanceSampling`(*args)	Post analytical importance sampling.
`SubsetSampling`(*args)	Subset simulation.
`AdaptiveDirectionalStratification`(*args)	Adaptive directional simulation.
`NAIS`(*args)	Nonparametric Adaptive Importance Sampling (NAIS) algorithm.
`CrossEntropyImportanceSampling`(*args)	Cross-Entropy Importance Sampling algorithm.
`PhysicalSpaceCrossEntropyImportanceSampling`(*args)	Physical Space Cross-Entropy Importance Sampling.
`StandardSpaceCrossEntropyImportanceSampling`(*args)	Standard Space Cross-Entropy Importance Sampling.

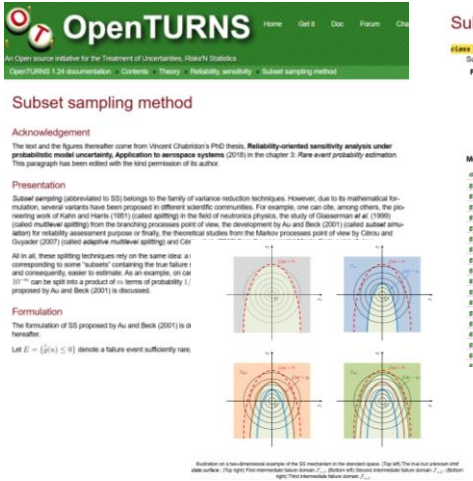
Focus on adaptive simulation methods for reliability analysis

Subset simulation

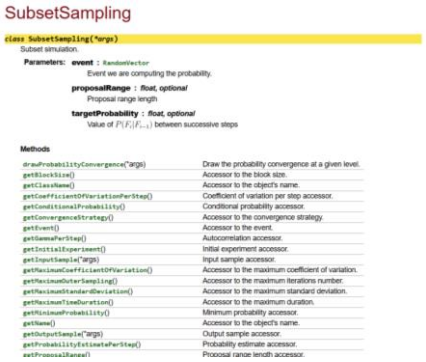
- The principle of Subset Simulation method is to decompose the target probability in a product of conditional probabilities with respect to intermediate thresholds that can be estimated with a reasonable simulation budget (use of Monte Carlo Markov Chain for conditional sampling)

$$P_f = \mathbb{P}(\boldsymbol{X} \in D_f) = \prod_{k=1}^{m} \mathbb{P}(\boldsymbol{X} \in D_k | \boldsymbol{X} \in D_{k-1}) \text{ with } D_0 = \mathbb{R}^d \supset D_1 \supset \cdots \supset D_{m-1} \supset D_m = D_f$$

Theoretical elements



API



Parameters :

- targetProbability → between successive steps

Interesting setters :

- setKeepSample()

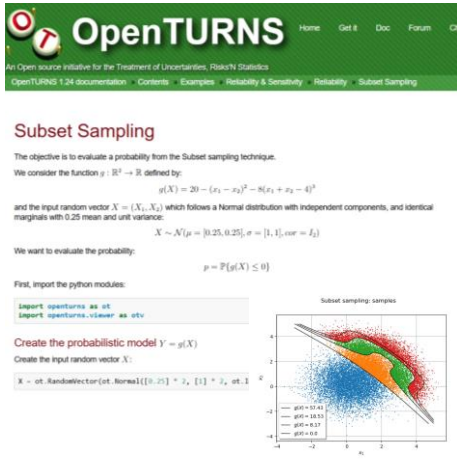
Interesting accessors based on *getResult()*:

- *getProbabilityEstimate()*
- *getStepsNumber()*

- *getProbabilityEstimatePerStep()*
- *getThresholdPerStep()*
- *getCoefficientOfVariationPerStep()*

- *getConfidenceLength()* → *for confidence interval*

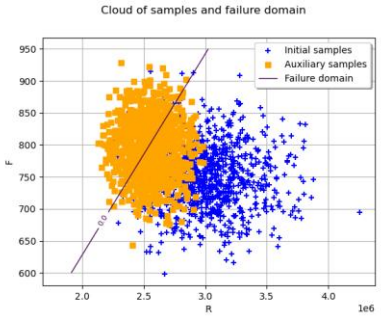
Examples



Cross Entropy Importance Sampling

- The principle of Cross Entropy Importance Sampling is to define an adaptive sequence of **parametric** auxiliary distributions $h_{\boldsymbol{\lambda}}$ with respect to intermediate thresholds

$$P_f = \int_{\mathbb{R}^d} \mathbf{1}_{g(\boldsymbol{x}) \leq T} f_{\boldsymbol{X}}(\boldsymbol{x}) \mathrm{d}\boldsymbol{x} = \int_{\mathbb{R}^d} \mathbf{1}_{g(\boldsymbol{x}) \leq T} \frac{f_{\boldsymbol{X}}(\boldsymbol{x})}{h_{\boldsymbol{\lambda}}(\boldsymbol{x})} h_{\boldsymbol{\lambda}}(x) \mathrm{d}\boldsymbol{x}$$



The « optimal » auxiliary distribution $h_{\boldsymbol{\lambda}_{\mathrm{opt}}}$ is defined iteratively based on a sequence of intermediate thresholds $T_0 > T_1 > \cdots \geq T$ with the determination of the corresponding parameters values $\boldsymbol{\lambda}_0, \boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_{\mathrm{opt}}$

$h_{\boldsymbol{\lambda}}$ is defined in the **physical space**

PhysicalSpaceCrossEntropyImportanceSampling()

$h_{\boldsymbol{\lambda}}$ is defined in the **standard space**

StandardSpaceCrossEntropyImportanceSampling()

Cross Entropy Importance Sampling

Specific to PhysicalSpace algorithm

Theoretical elements

The following explanations are given for a failure event defined as $g(\boldsymbol{X}) < T$ with $\boldsymbol{X}$ a random vector following a joint P
$f_{\boldsymbol{X}}$, T a threshold and g a limit state function, without loss of generality.

The Importance Sampling (IS) probability estimate $\widehat{P}^{\text{IS}}$ is given by:

$$\widehat{P}^{\text{IS}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}_{g(\boldsymbol{x}_i)<T} \frac{f_{\boldsymbol{X}}(\boldsymbol{x}_i)}{h(\boldsymbol{x}_i)},$$

with h the auxiliary PDF of Importance Sampling, N the number of independent samples generated with h and $\mathbf{1}_{g(\boldsymbol{x}_i)<T}$
dicator function of the failure domain.

The optimal density h_{opt} minimizing the variance of the estimator is defined as:

$$h_{\text{opt}} = \frac{\mathbf{1}_{g(\boldsymbol{x})<T} f_{\boldsymbol{X}}}{P},$$

with P the failure probability which is inaccessible in practice since this probability is the quantity of interest and unknow

The Physical Space Cross-Entropy Importance Sampling algorithm [**rubinstein2017**] uses a parametric auxiliary distrib
$h_{\boldsymbol{\lambda}}$ in order to optimize its parameters to compute the probability of interest with accuracy.

It involves an auxiliary optimization problem to find the auxiliary distribution parameters $\boldsymbol{\lambda}$ minimizing the Kullback-Leible
vergence with respect to h_{opt}. The following algorithm is used:

1. $k = 1$, set the quantile level $\rho \in [0, 1]$ and $h_0 = f_{\boldsymbol{X}}$
2. Generate the population $\boldsymbol{x}_1^{(k)}, \ldots, \boldsymbol{x}_N^{(k)}$ according to the PDF h_{k-1}, apply the function g in order to have $y_1^{(k)} = g(\boldsymbol{x}_1^{(k)}), \ldots, y_N^{(k)} = g(\boldsymbol{x}_N^{(k)})$
3. Compute the empirical ρ-quantile $q_k = \max(T, y_{\lfloor \rho N \rfloor}^{(k)})$ using the floor of ρN.
4. If $q_k > T$: go to Step 7
5. Estimate the auxiliary distribution parameters:

$$\boldsymbol{\lambda}_k = \operatorname{argmax}_{\boldsymbol{\lambda}} \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}_{g(\boldsymbol{x}_i^{(k)})\le q_k} \frac{f_{\boldsymbol{X}}(\boldsymbol{x}_i^{(k)})}{h_{\boldsymbol{\lambda}_{k-1}}(\boldsymbol{x}_i^{(k)})} \log(h_{\boldsymbol{\lambda}}(\boldsymbol{x}_i^{(k)}))$$

6. $k \leftarrow k + 1$, go to Step 2
7. Estimate the probability $\widehat{P}^{\text{CE}}(g(\boldsymbol{X}) < T) = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}_{\ldots} \frac{f_{\boldsymbol{X}}(\boldsymbol{x}_i^{(k)})}{\ldots}$

API

PhysicalSpaceCrossEntropyImportanceSampling

class PhysicalSpaceCrossEntropyImportanceSampling(*args)

Physical Space Cross-Entropy Importance Sampling.

Parameters: **event** : ThresholdEvent
Event we are computing the probability of.

activeParameters : *sequence of integers*
List of active parameters indices for the auxiliary distribution.

initialAuxiliaryDistributionParameters : *sequence of floats*
Initial value of active parameters of the auxiliary distribution.

bounds : Interval
Bounds on the active parameters of the auxiliary distribution.

auxiliaryDistribution : Distribution
Auxiliary distribution for the Cross Entropy Importance Sampling algorithm.

quantileLevel : *float* $0 < quantileLevel < 1$
Intermediate quantile level. The default number can be tweaked with the CrossEntropyImportanceSampling-DefaultQuantileLevel key from ResourceMap.

Methods

drawProbabilityConvergence(*args)	Draw the probability convergence at a given level.
getBlockSize()	Accessor to the block size.
getClassName()	Accessor to the object's name.
getConvergenceStrategy()	Accessor to the convergence strategy.
getEvent()	Accessor to the event.
getInputSample(*args)	Input sample accessor.
getMaximumCoefficientOfVariation()	Accessor to the maximum coefficient of variation.
getMaximumOuterSampling()	Accessor to the maximum iterations number.

Parameters :

- activeParameter
- initialAuxiliaryDistributionParameters
- bounds
- auxiliaryDistribution
- quantileLevel

Interesting setters :

- setKeepSample()

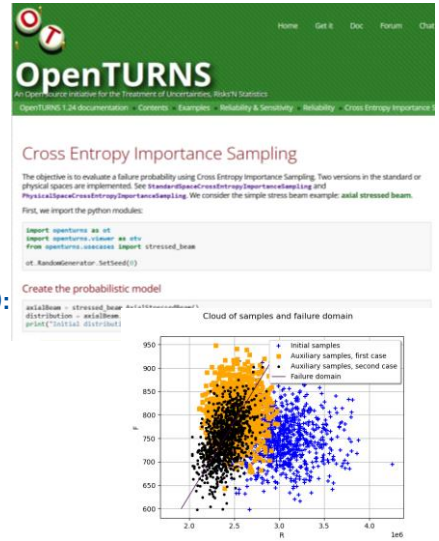
Interesting accessors based on *getResult()*:

- *getProbabilityEstimate()*
- *getStepsNumber()*

- *getProbabilityEstimatePerStep()*
- *getThresholdPerStep()*
- *getCoefficientOfVariationPerStep()*

- *getAuxiliaryDistribution()*
- *getAuxiliaryInputSample()*
- *getAuxiliaryOutputSample()*

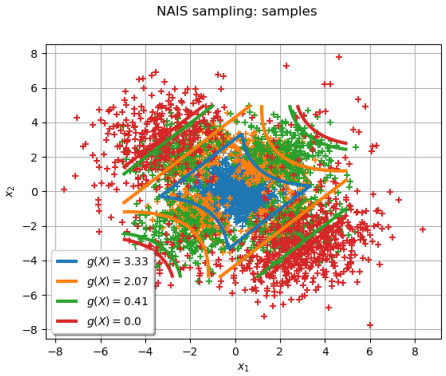
Examples



Nonparametric Adaptive Importance Sampling

- The principle of Nonparametric Adaptive Importance Sampling is to define an adaptive sequence of **nonparametric** auxiliary distributions h_λ with respect to intermediate thresholds

$$P_f = \int_{\mathbb{R}^d} \mathbf{1}_{g(\boldsymbol{x}) \leq T} f_{\boldsymbol{X}}(\boldsymbol{x}) \mathrm{d}\boldsymbol{x} = \int_{\mathbb{R}^d} \mathbf{1}_{g(\boldsymbol{x}) \leq T} \frac{f_{\boldsymbol{X}}(\boldsymbol{x})}{h_{\boldsymbol{\lambda}}(\boldsymbol{x})} h_{\boldsymbol{\lambda}}(x) \mathrm{d}\boldsymbol{x}$$



The « optimal » auxiliary distribution $h_{\boldsymbol{\lambda}_{\mathrm{opt}}}$ is defined iteratively through a sequence of intermediate thresholds $T_0 > T_1 > \cdots \geq T$ and based on **kernel smoothing method**

Nonparametric Adaptive Importance Sampling

Theoretical elements

The following explanations are given for a failure event defined as $g(\mathbf{X}) < T$ with $\mathbf{X}$ a random vector following a joint PDF $f_{\mathbf{X}}$, T a threshold and g a limit state function, without loss of generality.

The Importance Sampling (IS) probability estimate $\widehat{P}^{\mathrm{IS}}$ is given by:

$$\widehat{P}^{\mathrm{IS}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}_{g(\mathbf{x}_i)<T}\frac{h_0(\mathbf{x}_i)}{h(\mathbf{x}_i)},$$

with $h_0 = f_{\mathbf{X}}$ the PDF of $\mathbf{X}$, h the auxiliary PDF of Importance Sampling, N the number of independent samples generated with h and $\mathbf{1}_{g(\mathbf{x}_i)<T}$ the indicator function of the failure domain.

The optimal density minimizing the variance of the estimator h_{opt} is defined as:

$$h_{opt} = \frac{\mathbf{1}_{g(x)<T}h_0}{P},$$

with P the failure probability which is inaccessible in practice since this probability is the quantity of interest and unknown.

The objective of Non parametric Adaptive Importance Sampling (NAIS) [morio2015] is to approximate the IS optimal auxiliary density h_{opt} from the preceding equation with a kernel density function (e.g. Gaussian kernel). Its iterative principle is described by the following steps.

1. $k = 1$ and set the quantile level $\rho \in [0, 1]$
2. Generate the population $\mathbf{x}_1^{(k)}, ..., \mathbf{x}_N^{(k)}$ according to the PDF h_{k-1}, apply the function g in order to have $y_1^{(k)} = g(\mathbf{x}_1^{(k)}), ..., y_N^{(k)} = g(\mathbf{x}_N^{(k)})$
3. Compute the empirical quantile of level ρ $q_k = \max(T, y_{\lceil \rho N \rceil}^{(k)})$
4. Estimate $I_k = \frac{1}{kN}\sum_{j=1}^{k}\sum_{i=1}^{N}\mathbf{1}_{g(\mathbf{x}_i^{(j)})\leq q_k}\frac{h_0(\mathbf{x}_i^{(j)})}{h_{j-1}(\mathbf{x}_i^{(j)})}$
5. Update the Gaussian kernel sampling PDF with:

$$h_k(\mathbf{x}) = \frac{1}{kNI_k\det(B_{k+1})}\sum_{j=1}^{k}\sum_{i=1}^{N} w_j(\mathbf{x}_i^{(j)})K_d\left(B_{k+1}^{-1}\left(\mathbf{x}-\mathbf{x}_i^{(j)}\right)\right)$$

where K_d is the PDF of the standard d-dimensional normal distribution, $B_{k+1} = \mathrm{diag}(b_{k+1}^1, ..., b_{k+1}^d)$ and $w_j = \mathbf{1}_{g(\mathbf{x}_i^{(j)})\leq q_k}\frac{h_0(\mathbf{x}_i^{(j)})}{h_{j-1}(\mathbf{x}_i^{(j)})}$. The coefficients of the matrix B_{k+1} can be approximated (Silverman Rule) or postulated according to the AMISE (Asymptotic Mean Integrated Square Error) criterion for example.

API

NAIS

class NAIS(*args)

Nonparametric Adaptive Importance Sampling (NAIS) algorithm.

Parameters: **event** : RandomVector
Event we are computing the probability of.

quantileLevel : *float* $0 < quantileLevel < 1$
Intermediate quantile level.

Methods

drawProbabilityConvergence(*args)	Draw the probability convergence at a given level.
getBlockSize()	Accessor to the block size.
getClassName()	Accessor to the object's name.
getConvergenceStrategy()	Accessor to the convergence strategy.
getEvent()	Accessor to the event.
getInputSample(*args)	Input sample accessor.
getMaximumCoefficientOfVariation()	Accessor to the maximum coefficient of variation.
getMaximumOuterSampling()	Accessor to the maximum iterations number.
getMaximumStandardDeviation()	Accessor to the maximum standard deviation.
getMaximumTimeDuration()	Accessor to the maximum duration.
getName()	Accessor to the object's name.
getOutputSample(*args)	Output sample accessor.
getQuantileLevel()	Accessor to the intermediate quantile level.
getResult()	Accessor to the results.

Parameters :

- quantileLevel

Interesting setters :

- setKeepSample()

Interesting accessors based on *getResult()*:

- *getProbabilityEstimate()*
- *getStepsNumber()*

- *getProbabilityEstimatePerStep()*
- *getThresholdPerStep()*
- *getCoefficientOfVariationPerStep()*

- *getAuxiliaryDistribution()*
- *getAuxiliaryInputSample()*
- *getAuxiliaryOutputSample()*

Examples

OpenTURNS

Home | Get it | Doc | Forum | Chat

An Open source initiative for the Treatment of Uncertainties, Risk&N Statistics

OpenTURNS 1.24 documentation » Contents » Examples » Reliability & Sensitivity » Reliability »

Non parametric Adaptive Importance Sampling (NAIS)

The objective is to evaluate a probability from the Non parametric Adaptive Importance Sampling (NAIS) technique.

We consider the four-branch function $g : \mathbb{R}^2 \to \mathbb{R}$ defined by:

$$g(\boldsymbol{X}) = \min\begin{pmatrix} 3 + 0.1(x_1 - x_2)^2 - \frac{(x_1+x_2)}{\sqrt{2}} \\ 3 + 0.1(x_1 - x_2)^2 + \frac{(x_1+x_2)}{\sqrt{2}} \\ (x_1 - x_2) + \frac{7}{\sqrt{2}} \\ (x_2 - x_1) + \frac{7}{\sqrt{2}} \end{pmatrix}$$

and the input random vector $\boldsymbol{X} = (X_1, X_2)$ which follows the standard 2-dimensional Normal distribution:

$$\boldsymbol{X} \sim \mathcal{N}(\mu = [0, 0], \sigma = [1, 1], corr = \boldsymbol{I}_2)$$

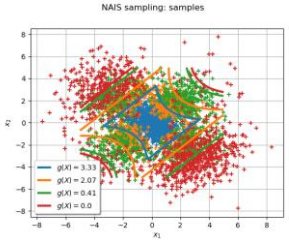
We want to evaluate the probability:

$$p = \mathbb{P}(g(\boldsymbol{X}) \leq 0)$$

First, import the python modules:

```
import openturns as ot
from openturns.viewer im
import math
```

Create the probabilisti



Adaptive Directional Stratification

- The principle of Adaptive Directional Stratification combines stratified and directional sampling concepts.
  - Stratified sampling consists in splitting the support of the random vector $\boldsymbol{X}$ into m mutually exclusive and collectively exhaustive subsets.
  - Directional sampling uses the spheric symmetry of the standard space for estimating the failure probability as the average of conditional probabilities calculated on directions drawn at random in the standard space.

Theoretical elements

Let $\mathcal{D}_f$ denote the failure domain defined as $\mathcal{D}_f = \{\boldsymbol{x} \in \mathbb{R}^{n_X} | g(\boldsymbol{x}) \leq 0\}$, where $\boldsymbol{x}$ are realization of the random vector $\boldsymbol{X}$ and g is the limit-state function as defined elsewhere in the documentation.

The purpose of the ADS-2 algorithm and its variants is to estimate the following probability:

$$P_f = \int_{\mathcal{D}_f} f_{\boldsymbol{X}}(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} = \int_{\mathbb{R}^{n_X}} \mathbb{1}_{\{g(\boldsymbol{x}) \leq 0\}} f_{\boldsymbol{X}}(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} = \mathbb{P}\left(\{g(\boldsymbol{X}) \leq 0\}\right).$$

Principles

The ADS-2 method [munoz2011] combines the stratified and directional sampling concepts. Stratified sampling consists in splitting the support of the random vector $\boldsymbol{x}$ into m mutually exclusive and collectively exhaustive subsets. Here, ADS-2 splits the standard space into $m = 2^d$ quadrants, where d is the dimension of the random vector $\boldsymbol{X}$. Stratified sampling is often run in two steps: (i) a learning step is used for polling the input space and detect the subsets that contribute most to the probability and (ii) an estimation step is used for estimating the probability by weighted sampling (some subsets are more sampled than the others). Directional sampling uses the spheric symmetry of the standard space for estimating the failure probability as the average of conditional probabilities calculated on directions drawn at random in the standard space.

The learning step uses an a priori number of random directions that is uniformly distributed over the quadrants, meaning the weights are as follows:

$$\omega_i^1 = \frac{1}{m}, \quad i = 1, \ldots, m.$$

Directional sampling is used for estimating the failure probability in each quadrant:

$$\hat{P}_i^{DS} = \mathbb{P}\left(\{g(\boldsymbol{X}) \leq 0\} \mid \boldsymbol{X} \in \mathbb{Q}_i\right), \; i = 1, \ldots, m.$$

and the corresponding estimation variances are denoted as $\sigma_i^{DS\,2}$. These probabilities are estimated using the same number N_i^0 of random directions per quadrant as told by the uniform weights distribution.

The probability of interest is then computed as a weighted average of the previously defined conditional probabilities:

$$\hat{P}_f = \sum_{i=1}^{m} \omega_i \hat{P}_i^{DS}$$

API

AdaptiveDirectionalStratification

```
class AdaptiveDirectionalStratification(*args)
```

Adaptive directional simulation.

Parameters: **event** : RandomVector
Event we are computing the probability of.

rootStrategy : RootStrategy, *optional*
Strategy adopted to evaluate the intersections of each direction with the limit state function and take into account the contribution of the direction to the event probability. Set to SafeAndSlow by default.

samplingStrategy : SamplingStrategy, *optional*
Strategy adopted to sample directions. Set to RandomDirection by default.

Methods

drawProbabilityConvergence(*args)	Draw the probability convergence at a given level.
getBlockSize()	Accessor to the block size.
getClassName()	Accessor to the object's name.
getConvergenceStrategy()	Accessor to the convergence strategy.
getEvent()	Accessor to the event.
getGamma()	Gamma accessor.
getMaximumCoefficientOfVariation()	Accessor to the maximum coefficient of variation.
getMaximumOuterSampling()	Accessor to the maximum iterations number.

Examples

Parameters :

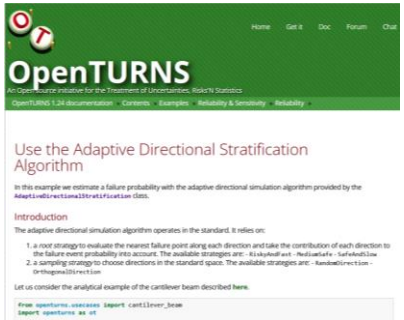
- rootStrategy → evaluate the intersections of each direction with the limit state function
- samplingStrategy → random, orthogonal

Interesting setters :

- setGamma() → computational budget per steps

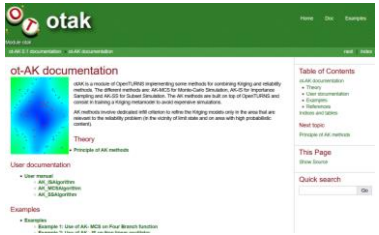
Interesting accessors based on *getResult()*:

- *getProbabilityEstimate()*



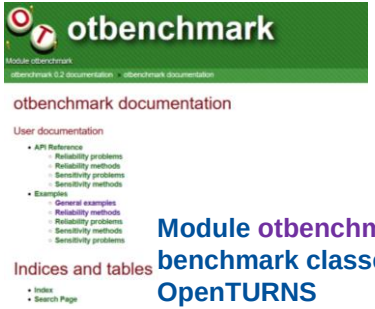
Advantages / drawbacks of each technique

Method	Advantages	Drawbacks
Subset sampling	Multi-failure modes Flexibility	MCMC convergence
Cross Entropy Importance Sampling	Scale with dimension	Multi-failure modes, optimization of parameters
Nonparametric Adaptive Importance Sampling	Multi-failure modes Flexibility	Scale with dimension, kernel smoothing bandwith estimate
Adaptive Directional Stratification	Multi-failure modes	Scale with dimension, regularity and differentiability assumptions



Module otak for surrogate-based reliability analysis

https://m-balesdent.github.io/otak/master/



Module otbenchmark : benchmark classes for OpenTURNS

https://openturns.github.io/otbenchmark/master/

Future module otCEISVAE based on Variational AutoEncoder for high dimensional problems (release expected in 2025)

- Based on PhD. work of J. Demange-Chryst (ONERA)