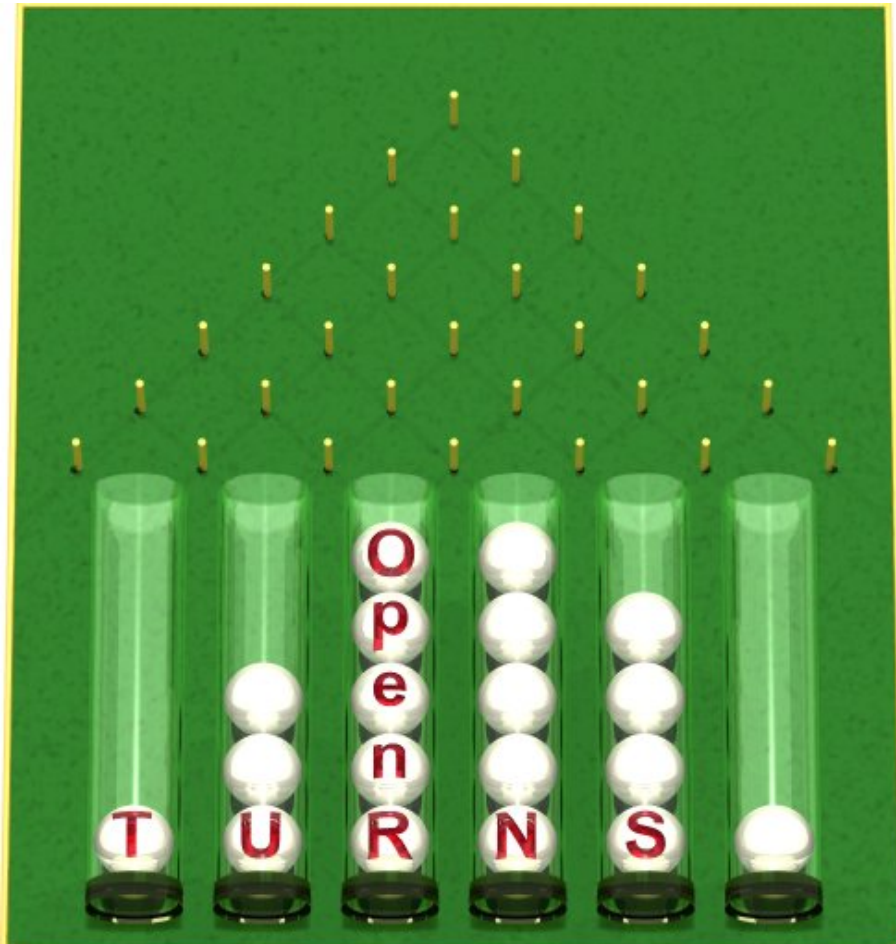


Examples Guide

Open TURNS version 0.14.0

Documentation built from package openturns-doc-June2011

June 30, 2011



Contents

1	Example 1 : deviation of a cantilever beam	2
1.1	Presentation of the study case	2
1.2	Probabilistic modelisation	3
1.2.1	Marginal distributions	3
1.2.2	Dependence structure	3
1.3	Min/Max approach	4
1.3.1	Deterministic experiment plane	4
1.3.2	Random sampling	4
1.4	Central tendency approach	4
1.4.1	Taylor variance decomposition	4
1.4.2	Random sampling	4
1.4.3	Kernel smoothing	4
1.5	Threshold exceedance approach	4
1.5.1	FORM	4
1.5.2	Monte Carlo simulation method	5
1.5.3	Directional Sampling method	5
1.5.4	Latin Hyper Cube Sampling method	5
1.5.5	Importance Sampling method	5
1.6	Response surface by polynomial chaos expansion	5
1.7	The Python script	6
1.8	Output of the Python script	26
1.9	Figures	31
1.10	Results comments	36
1.10.1	Min/Max approach	36
1.10.2	Central tendency approach	36
1.10.3	Threshold exceedance approach	36
1.10.4	Response surface : Polynomial expansion chaos	37

1 Example 1 : deviation of a cantilever beam

1.1 Presentation of the study case

This Example Guide regroups several Use Cases described in the Use Cases Guide in order to show one example of a complete study.

This example has been presented in the ESREL 2007 conference in the paper : *Open TURNS, an Open source initiative to Treat Uncertainties, Risks'N Statistics in a structured industrial approach*, from A. Dutfoy(EDF R&D), I. Dutka-Malen(EDF R&D), R. Lebrun (EADS innovation Works) *et al.*

Let's consider the following analytical example of a cantilever beam, of Young's modulus E , length L , section modulus I . One end is built in a wall and we apply a concentrated bending load at the other end of the beam. The deviation (vertical displacement) y of the free end is equal to :

$$y(E, F, L, I) = \frac{FL^3}{3EI}$$

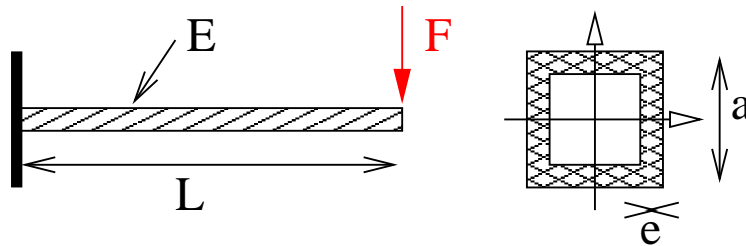


Figure 1: cantilever beam under a punctual bending load.

The objective of this study is to evaluate the influence of uncertainties of the input data (E, F, L, I) on the deviation y .

We consider a steel beam with a hollow square section of length a and of thickness e . Thus, the flexion section inertie of the beam is equal to $I = \frac{a^4 - (a - e)^4}{12}$. The beam length is L . The Young's modulus is E . The charge applied is F .

The values used for the deterministic studies are :

$$\begin{cases} E = 3.0e9Pa \\ F = 300N \\ L = 2.5m \\ I = 4.0e - 6m^4. \end{cases}$$

which corresponds to the point (3.0e7, 30000, 250, 400) when the length L is given in unit cm et noo in the standard unit m .

This example treats the following points of the methodology :

- Min/Max approach : evaluation of the range of the output variable of interest (deviation)

- with a deterministic experiment plane,
- with a random experiment plane,
- Central tendency approach : evaluation of the central indicators of the output variable of interest (deviation)
 - Taylor variance decomposition,
 - Random sampling,
 - Kernel smoothing of the distribution of the output variable of interest,
- Threshold exceedance approach : evaluation of the probability that the output variable of interest (deviation) $30 \geq 30cm$
 - FORM,
 - Monte Carlo simulation method,
 - Directional Sampling method,
 - Importance Sampling method.

1.2 Probabilistic modelisation

1.2.1 Marginal distributions

The random modelisation of the input data is the following one :

- $E = \text{Beta}(\ast)$ where $r = 0.93, t = 3.2, a = 2.8e7, b = 4.8e7$,
- $F = \text{LogNormal}$, where the mean value is $E[F] = 30000$, the standard deviation is $\sqrt{\text{Var}[F]} = 9000$ and the min value is $\min(E) = 15000$,
- $L = \text{Uniform on } [250; 260]$,
- $I = \text{Beta}(\ast)$ where $r = 2.5, t = 4.0, a = 3.1e2, b = 4.5e2$.

(*) We recall here the expression of the probability density function of the Beta distribution :

$$p(x) = \frac{(x-a)^{(r-1)}(b-x)^{(t-r-1)}}{(b-a)^{(t-1)}B(r, t-r)} \mathbf{1}_{[a,b]}(x)$$

where $r > 0, t > r$ and $a < b$.

1.2.2 Dependence structure

We suppose that the probabilistic variables L and I are dependent. This dependence may be explained by the manufacturing process of the beam : the thinner the beam has been laminated, the longer it is.

We modelise the dependence structure by a Normal copula, parameterized from the Spearman correlation coefficient of both correlated variables : $\rho_S = -0.2$.

Then, the Spearman correlation matrix of the input random vector (E, F, L, I) is :

$$R_S = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -0.2 \\ 0 & 0 & -0.2 & 1 \end{pmatrix}$$

1.3 Min/Max approach

1.3.1 Deterministic experiment plane

We consider a composite experiment plane, where :

- the levels of the centered and reduced grid are ± 0.5 , $\pm 1.$, $\pm 3.$,
- the unit per dimension (scaling factor) is given by the standard deviation of the marginal distribution of the corresponding variable,
- the center is the mean point of the input random vector distribution.

1.3.2 Random sampling

We evaluate the range of the deviation from a random sample of size 10^4 .

1.4 Central tendency approach

1.4.1 Taylor variance decomposition

We evaluate the mean and the standard deviation of the deviation thanks to the Taylor variance decomposition method. The importance factors of that method rank the influence of the input uncertainties on the mean of the deviation.

1.4.2 Random sampling

We evaluate the mean and standard deviation of the deviation from a random sample of size 10^4 .

1.4.3 Kernel smoothing

We fit the distribution of the deviation with a Normal kernel, which bandwidth is evaluated from the Scott rule, from a random sample of size 10^4 .

We superpose then the kernel smoothing pdf and the normal one which mean and standard deviation are those of the random sample of the output variable of interest in order to graphically check if the Normal model fits to the deviation distribution.

1.5 Threshold exceedance approach

We consider the event where the deviation exceeds $30cm$.

1.5.1 FORM

We use the Cobyala algorithm to research the design point, which requires no evaluation of the gradient of the limit state function. We parameterize the Cobyala algorithm with the following parameters :

- Maximum Iterations Number = 10^3 ,
- Maximum Absolute Error = 10^{-10} ,
- Maximum Relative Error = 10^{-10} ,
- Maximum Residual Error = 10^{-10} ,
- Maximum Constraint Error = 10^{-10} .

1.5.2 Monte Carlo simulation method

We evaluate the probability with the Monte Carlo method, parameterized as follows :

- Maximum Outer Sampling = $4 \cdot 10^4$,
- Block Size = 10^2 ,
- Maximum Coefficient of Variation = 10^{-1} .

We evaluate the confidence interval of level 0.95 and we draw the convergence graph of the Monte Carlo estimator with its confidence interval of level 0.90.

1.5.3 Directional Sampling method

We evaluate the probability with the Directional Sampling method, with its default parameters :

- 'Slow and Safe' for the root strategy,
- 'Random direction' for the sampling strategy

We evaluate the confidence interval of level 0.95 and we draw the convergence graph of the Directional Sampling estimator with its confidence interval of level 0.90.

1.5.4 Latin Hyper Cube Sampling method

We evaluate the probability with the Latin Hyper Cube Sampling method with the same parameters as the Monte Carlo method and we draw the convergence graph of the LHS estimator with its confidence interval of level 0.90.

1.5.5 Importance Sampling method

We evaluate the probability with the Importance Sampling method in the standard sapce, with the same parameters as the Monte carlo method. The importance distribution is the normal one, centered on the standard design point and which standard deviation is 4. The importance sampling is performed in the standard sapce.

We fix the BlockSize is fixed to 1 and the MaximumOuterIteration to $4 \cdot 10^4$.

We draw the convergence graph of the Importance Sampling estimator with its confidence interval of level 0.90.

1.6 Response surface by polynomial chaos expansion

We evaluate the meta model determined thanks to the polynomial chaos expansion technique.

We took the following 1D polynomial families, which parameters have been determined in order to be adapted to the marginal distributions of the input random vector :

- E : Jacobi($\alpha = 1.3$, $\beta = -0.1$),
- F : Laguerre($k = 1.78$),
- L : Legendre,
- I : Jacobi($\alpha = 0.5$, $\beta = 1.5$).

The truncature strategy of the multivariate orthonormal basis is the Cleaning Strategy where we considered within the 500 first terms of the multivariate basis, among the 50 most significant ones, those which contribution wre significant (which means superior to 10^{-4}).

The evaluation strategy of the approximation coefficients is the least square strategy based on a experiment plane determined with the Monte Carlo sampling technique of size 100.

Figures (14) to (18) draw the following graphs :

- the drawings of some members of the 1D polynomial family,
- the cloud of points making the comparison between the model values and the meta model ones : if the adequation is perfect, points must be on the first diagonal.

1.7 The Python script

```

1 from openturns import *
2
3 from math import *
4
5 #####
6 ### Function 'deviation'
7 #####
8 # Create here the python lines to define the implementation of the function
9
10 # In order to be able to use that function with the openturns library,
11 # it is necessary to define a class which derives from OpenTURNPythonFunction
12
13 class modelePYTHON(OpenTURNPythonFunction) :
14     # that following method defines the input size (4) and the output size (1)
15     def __init__(self) :
16         OpenTURNPythonFunction.__init__(self,4,1)
17
18     # that following method gives the implementation of modelePYTHON
19     def f(self,x) :
20         E=x[0]
21         F=x[1]
22         L=x[2]
23         I=x[3]
24         return [(F*L*L*L)/(3.*E*I)]
25
26 # Use that function defined in the script python
27 # with the openturns library
28 # Create a NumericalMathFunction from modelePYTHON
29 deviation = NumericalMathFunction(modelePYTHON())
30
31
32 #####
33 ### Input random vector

```

```
34 #####
35
36 # Create the marginal distributions of the input random vector
37 distributionE = Beta(0.93, 3.2, 2.8e7, 4.8e7)
38 distributionF = LogNormal(30000, 9000, 15000, LogNormal.MUSIGMA)
39 distributionL = Uniform(250, 260)
40 distributionI = Beta(2.5, 4.0, 3.1e2, 4.5e2)
41
42 # Visualize the probability density functions
43
44 pdfLoiE = distributionE.drawPDF()
45 # Change the legend
46 draw_E = pdfLoiE.getDrawable(0)
47 draw_E.setLegendName("Beta(0.93, 3.2, 2.8e7, 4.8e7)")
48 pdfLoiE.setDrawable(draw_E,0)
49 # Change the title
50 pdfLoiE.setTitle("PDF_of_E")
51
52 pdfLoiE.draw("distributionE_pdf", 640, 480)
53 #Show(pdfLoiE)
54
55 pdfLoiF = distributionF.drawPDF()
56 # Change the legend
57 draw_F = pdfLoiF.getDrawable(0)
58 draw_F.setLegendName("LogNormal(30000, 9000, 15000)")
59 pdfLoiF.setDrawable(draw_F,0)
60 # Change the title
61 pdfLoiF.setTitle("PDF_of_F")
62
63 pdfLoiF.draw("distributionF_pdf", 640, 480)
64 #Show(pdfLoiF)
65
66 pdfLoiL = distributionL.drawPDF()
67 # Change the legend
68 draw_L = pdfLoiL.getDrawable(0)
69 draw_L.setLegendName("Uniform(250, 260)")
70 pdfLoiL.setDrawable(draw_L,0)
71 # Change the title
72 pdfLoiL.setTitle("PDF_of_L")
73
74 pdfLoiL.draw("distributionL_pdf", 640, 480)
75 #Show(pdfLoiL)
76
77
78 pdfLoiI = distributionI.drawPDF()
79 # Change the legend
80 draw_I = pdfLoiI.getDrawable(0)
81 draw_I.setLegendName("Beta(2.5, 4.0, 3.1e2, 4.5e2)")
```

```
82 pdfLoiI.setDrawable(draw_I,0)
83 # Change the title
84 pdfLoiI.setTitle("PDF_of_I")
85
86 pdfLoiI.draw("distributionI_pdf", 640, 480)
87 #Show(pdfLoiI)
88
89 # Create the Spearman correlation matrix of the input random vector
90 RS = CorrelationMatrix(4)
91 RS[2,3] = -0.2
92
93 # Evaluate the correlation matrix of the Normal copula from RS
94 R = NormalCopula.GetCorrelationFromSpearmanCorrelation(RS)
95
96 # Create the Normal copula parametrized by R
97 copuleNormal = NormalCopula(R)
98
99 # Create a collection of the marginal distributions
100 collectionMarginals = DistributionCollection(4)
101 collectionMarginals[0] = Distribution(distributionE)
102 collectionMarginals[1] = Distribution(distributionF)
103 collectionMarginals[2] = Distribution(distributionL)
104 collectionMarginals[3] = Distribution(distributionI)
105
106 # Create the input probability distribution of dimension 4
107 inputDistribution = ComposedDistribution(collectionMarginals, Copula(
    copuleNormal))
108
109 # Give a description of each component of the input distribution
110 inputDescription = Description(4)
111 inputDescription[0] = "E"
112 inputDescription[1] = "F"
113 inputDescription[2] = "L"
114 inputDescription[3] = "I"
115 inputDistribution.setDescription(inputDescription)
116
117 # Create the input random vector
118 inputRandomVector = RandomVector(inputDistribution)
119
120 # Create the output variable of interest
121 outputVariableOfInterest = RandomVector(deviation, inputRandomVector)
122
123
124 #####
125 ### Min/Max approach Study
126 #####
127
128
```

```
129 #####
130 # Min/Max study with deterministic experiment plane
131 #####
132
133 print "#####"
134 print " _Min/Max_study_with_deterministic_experiment_plane"
135 print "#####"
136
137
138 dim = deviation.getInputDimension()
139
140 # Create the structure of the experiment plane : Composite type
141
142 # On each direction separately, several levels are evaluated
143 # here, 3 levels : +/-0.5, +/-1., +/-3. from the center
144 levelsNumber = 3
145 levels = NumericalPoint(levelsNumber, 0.0)
146 levels[0] = 0.5
147 levels[1] = 1.0
148 levels[2] = 3.0
149 # Creation of the composite plane
150 myPlane = Composite(dim, levels)
151
152 # Generation of points according to the structure of the experiment plane
153 # (in a reduced centered space)
154 inputSample = myPlane.generate()
155
156 # Scaling of the structure of the experiment plane
157 # scaling vector for each dimension of the levels of the structure
158 # to take into account the dimension of each component
159 # for example : the standard deviation of each component of 'inputRandomVector'
160 # in case of a RandomVector
161 scaling = NumericalPoint(dim)
162 scaling[0] = sqrt(inputRandomVector.getCovariance()[0,0])
163 scaling[1] = sqrt(inputRandomVector.getCovariance()[1,1])
164 scaling[2] = sqrt(inputRandomVector.getCovariance()[2,2])
165 scaling[3] = sqrt(inputRandomVector.getCovariance()[3,3])
166
167 inputSample.scale(scaling)
168
169
170 # Translation of the nonReducedSample onto the center of the experiment plane
171 # center = mean point of the inputRandomVector distribution
172 center = inputRandomVector.getMean()
173 inputSample.translate(center)
174
175 # Get the number of points in the experiment plane
176 pointNumber = inputSample.getSize()
```

```

177
178 # Evaluate the output variable of interest on the experiment plane
179 outputSample = deviation(inputSample)
180
181
182 # Evaluate the range of the output variable of interest on that experiment plane
183 minValue = outputSample.getMin()
184 maxValue = outputSample.getMax()
185
186 print "From_a_composite_experiment_plane_of_size_", pointNumber
187 print "Levels_", levels[0], ", ", levels[1], ", ", levels[2]
188 print "Min_Value_", minValue[0]
189 print "Max_Value_", maxValue[0]
190 print ""
191
192 #####
193 # Min/Max study by random sampling
194 #####
195
196 print "#####"
197 print " _Min/Max_study_by_random_sampling"
198 print "#####"
199
200 pointNumber = 10000
201 print "From_random_sampling_", pointNumber
202 outputSample2 = outputVariableOfInterest.getNumericalSample(pointNumber)
203
204 minValue2 = outputSample2.getMin()
205 maxValue2 = outputSample2.getMax()
206
207 print "Min_Value_", minValue2[0]
208 print "Max_Value_", maxValue2[0]
209 print ""
210
211
212
213
214
215 print ""
216 #####
217 ### Random Study : central tendance of
218 ### the output variable of interest
219 #####
220
221 print "#####"
222 print "Random_Study_: _central_tendance_of"
223 print "the_output_variable_of_interest"
224 print "#####"

```

```

225 print ""
226
227 #####
228 # Taylor variance decomposition
229 #####
230
231 print "#####"
232 print "Taylor_variance_decomposition"
233 print "#####"
234 print ""
235
236 # We create a quadraticCumul algorithm
237 myQuadraticCumul = QuadraticCumul(outputVariableOfInterest)
238
239 # We compute the several elements provided by the quadratic cumul algorithm
240 # and evaluate the number of calculus needed
241 nbBefr = deviation.getEvaluationCallsNumber()
242
243 # Mean first order
244 meanFirstOrder = myQuadraticCumul.getMeanFirstOrder()[0]
245 nbAfter1 = deviation.getEvaluationCallsNumber()
246
247 # Mean second order
248 meanSecondOrder = myQuadraticCumul.getMeanSecondOrder()[0]
249 nbAfter2 = deviation.getEvaluationCallsNumber()
250
251 # Standard deviation
252 stdDeviation = sqrt(myQuadraticCumul.getCovariance()[0,0])
253 nbAfter3 = deviation.getEvaluationCallsNumber()
254
255 print "First_order_mean=", myQuadraticCumul.getMeanFirstOrder()[0]
256 print "Evaluation_calls_number=", nbAfter1 - nbBefr
257 print "Second_order_mean=", myQuadraticCumul.getMeanSecondOrder()[0]
258 print "Evaluation_calls_number=", nbAfter2 - nbAfter1
259 print "Standard_deviation=", sqrt(myQuadraticCumul.getCovariance()[0,0])
260 print "Evaluation_calls_number=", nbAfter3 - nbAfter2
261
262 print "Importance_factors="
263 for i in range(inputRandomVector.getDimension()):
264     print inputDistribution.getDescription()[i], "=", myQuadraticCumul.
        getImportanceFactors()[i]
265 print ""
266
267
268 #####
269 # Random sampling
270 #####
271

```

```

272 print "#####"
273 print "Random_sampling"
274 print "#####"
275
276 size1 = 10000
277 output_Sample1 = outputVariableOfInterest.getNumericalSample(size1)
278 outputMean = output_Sample1.computeMean()
279 outputCovariance = output_Sample1.computeCovariance()
280
281 print "Sample_size=", size1
282 print "Mean_from_sample=", outputMean[0]
283 print "Standard_deviation_from_sample=", sqrt(outputCovariance[0,0])
284 print ""
285
286
287 #####
288 # Kernel Smoothing Fitting
289 #####
290
291
292 print "#####"
293 print "#_Kernel_Smoothing_Fitting"
294 print "#####"
295
296 # We generate a sample of the output variable
297 size = 10000
298 output_sample = outputVariableOfInterest.getNumericalSample(size)
299
300 # We build the kernel smoothing distribution
301 kernel = KernelSmoothing()
302 bw = kernel.computeSilvermanBandwidth(output_sample)
303 smoothed = kernel.build(output_sample, bw)
304 print "Sample_size=", size
305 print "Kernel_bandwidth=", kernel.getBandwidth()[0]
306
307 # We draw the pdf and cdf from kernel smoothing
308 # Evaluate at best the range of the graph
309 mean_sample = output_sample.computeMean()[0]
310 standardDeviation_sample = sqrt(output_sample.computeCovariance()[0,0])
311 xmin = mean_sample - 4*standardDeviation_sample
312 xmax = mean_sample + 4*standardDeviation_sample
313
314 # Draw the PDF
315 smoothedPDF = smoothed.drawPDF(xmin, xmax, 251)
316 # Change the title
317 smoothedPDF.setTitle("Kernel_smoothing_of_the_deviation_-_PDF")
318 # Change the legend
319 smoothedPDF_draw = smoothedPDF.getDrawable(0)

```

```

320 title = "PDF_from_Normal_kernel_" + str(size) + "_data)"
321 smoothedPDF_draw.setLegendName(title)
322 smoothedPDF.setDrawable(smoothedPDF_draw,0)
323 smoothedPDF.draw("smoothedPDF", 640, 480)
324
325 # Draw the CDF
326 smoothedCDF = smoothed.drawCDF(xmin, xmax, 251)
327 # Change the title
328 smoothedCDF.setTitle("Kernel_smoothing_of_the_deviation_-_CDF")
329 # Change the legend
330 smoothedCDF_draw = smoothedCDF.getDrawable(0)
331 title = "CDF_from_Normal_kernel_" + str(size) + "_data)"
332 smoothedCDF_draw.setLegendName(title)
333 smoothedCDF.setDrawable(smoothedCDF_draw,0)
334 # Change the legend position
335 smoothedCDF.setLegendPosition("bottomright")
336 smoothedCDF.draw("smoothedCDF", 640, 480)
337
338 # In order to see the graph without creating the associated files
339 #Show(smoothedCDF)
340 #Show(smoothedPDF)
341
342 # Mean of the output variable of interest
343 print "Mean_from_kernel_smoothing_=", smoothed.getMean()[0]
344 print ""
345
346 # Superposition of the kernel smoothing pdf and the gaussian one
347 # which mean and standard deviation are those of the output_sample
348 normalDist = NormalFactory().build(output_sample)
349 normalDistPDF = normalDist.drawPDF(xmin, xmax, 251)
350 normalDistPDFDrawable = normalDistPDF.getDrawable(0)
351 normalDistPDFDrawable.setColor('blue')
352 smoothedPDF.add(normalDistPDFDrawable)
353 smoothedPDF.draw("smoothedPDF_and_NormalPDF", 640, 480)
354
355 # In order to see the graph without creating the associated files
356 #Show(smoothedPDF)
357
358 #####
359 ### Probabilistic Study : threshold exceedance: deviation > 30cm
360 #####
361
362 print ""
363 print "#####"
364 print "Probabilistic_Study:_threshold_exceedance:_deviation_<-1cm"
365 print "#####"
366 print ""
367

```

```

368 #####
369 # FORM
370 #####
371
372 print "#####"
373 print "FORM"
374 print "#####"
375
376 # We create an Event from this RandomVector
377 # threshold has been defined in the kernel smoothing section
378 threshold = 30
379 myEvent = Event(outputVariableOfInterest , ComparisonOperator( Greater() ) ,
380                 threshold)
381
382 # We create a NearestPoint algorithm
383 myCobyla = Cobyla()
384 myCobyla.setMaximumIterationsNumber(1000)
385 myCobyla.setMaximumAbsoluteError(1.0e-10)
386 myCobyla.setMaximumRelativeError(1.0e-10)
387 myCobyla.setMaximumResidualError(1.0e-10)
388 myCobyla.setMaximumConstraintError(1.0e-10)
389
390 # We create a FORM algorithm
391 # The first parameter is a NearestPointAlgorithm
392 # The second parameter is an event
393 # The third parameter is a starting point for the design point research
394 meanVector = inputRandomVector.getMean()
395 myAlgoFORM = FORM(NearestPointAlgorithm(myCobyla) , myEvent , meanVector)
396
397 # Get the number of times the limit state function has been evaluated so far
398 deviationCallNumberBeforeFORM = deviation.getEvaluationCallsNumber()
399
400 # Perform the simulation
401 myAlgoFORM.run()
402
403 # Get the number of times the limit state function has been evaluated so far
404 deviationCallNumberAfterFORM = deviation.getEvaluationCallsNumber()
405
406 # Stream out the result
407 resultFORM = myAlgoFORM.getResult()
408 print "FORM_event_probability=" , resultFORM.getEventProbability()
409 print "Number_of_evaluations_of_the_limit_state_function_=",
410       deviationCallNumberAfterFORM - deviationCallNumberBeforeFORM
411 print "Generalized_reliability_index=" , resultFORM.
412       getGeneralisedReliabilityIndex()
413 print "Standard_space_design_point="
414 for i in range(inputRandomVector.getDimension()) :

```

```

413     print inputDistribution.getDescription()[i], " _=_", resultFORM.
        getStandardSpaceDesignPoint()[i]
414 print "Physical_space_design_point="
415 for i in range(inputRandomVector.getDimension()):
416     print inputDistribution.getDescription()[i], " _=_", resultFORM.
        getPhysicalSpaceDesignPoint()[i]
417
418 print "Importance_factors="
419 for i in range(inputRandomVector.getDimension()):
420     print inputDistribution.getDescription()[i], " _=_", resultFORM.
        getImportanceFactors()[i]
421
422 print "Hasofer_reliability_index=" , resultFORM.getHasoferReliabilityIndex()
423 print ""
424
425 # Graph 1 : Importance Factors graph */
426 importanceFactorsGraph = resultFORM.drawImportanceFactors()
427 title = "FORM_Importance_factors_=_"+ myEvent.getName()
428 importanceFactorsGraph.setTitle( title )
429 importanceFactorsGraph.draw("ImportanceFactorsDrawingFORM", 640, 480)
430
431 # In order to see the graph without creating the associated files
432 #Show(importanceFactorsGraph)
433
434
435 #####
436 # MC
437 #####
438
439 print "#####"
440 print "Monte_Carlo"
441 print "#####"
442 print ""
443
444
445 maximumOuterSampling = 40000
446 blockSize = 100
447 coefficientOfVariation = 0.10
448
449 # We create a Monte Carlo algorithm
450 myAlgoMonteCarlo = MonteCarlo(myEvent)
451 myAlgoMonteCarlo.setMaximumOuterSampling(maximumOuterSampling)
452 myAlgoMonteCarlo.setBlockSize(blockSize)
453 myAlgoMonteCarlo.setMaximumCoefficientOfVariation(coefficientOfVariation)
454
455 # Define the HistoryStrategy to store the numerical samples generated
456 # both for the input random vector and the output random vector
457 # Full strategy

```

```

458 myAlgoMonteCarlo.setInputOutputStrategy( HistoryStrategy( Full() ) )
459
460 # Define the HistoryStrategy to store the values of the probability estimator
461 # and the variance estimator
462 # used ot draw the convergence graph
463 # Full strategy
464 myAlgoMonteCarlo.setConvergenceStrategy( HistoryStrategy( Full() ) )
465
466 # Perform the simulation
467 myAlgoMonteCarlo.run()
468
469 # Display number of iterations and number of evaluations
470 # of the limit state function
471 print "Number_of_evaluations_of_the_limit_state_function = ", myAlgoMonteCarlo.
    getResult().getOuterSampling() * myAlgoMonteCarlo.getResult().getBlockSize()
472
473 # Display the Monte Carlo probability of 'myEvent'
474 print "Monte_Carlo_probability_estimation = ", myAlgoMonteCarlo.getResult().
    getProbabilityEstimate()
475
476 # Display the variance of the Monte Carlo probability estimator
477 print "Variance_of_the_Monte_Carlo_probability_estimator = ", myAlgoMonteCarlo.
    getResult().getVarianceEstimate()
478
479 # Display the confidence interval length centered around
480 # the MonteCarlo probability MCProb
481 # IC = [MCProb - 0.5*length, MCProb + 0.5*length]
482 # level 0.95
483
484 print "0.95_Confidence_Interval = [" , myAlgoMonteCarlo.getResult().
    getProbabilityEstimate() - 0.5*myAlgoMonteCarlo.getResult().
    getConfidenceLength(0.95), ", " , myAlgoMonteCarlo.getResult().
    getProbabilityEstimate() + 0.5*myAlgoMonteCarlo.getResult().
    getConfidenceLength(0.95), "]"
485 print ""
486
487 # Draw the convergence graph and the confidence intervalle of level alpha
488 alpha = 0.90
489 convergenceGraphMonteCarlo = myAlgoMonteCarlo.drawProbabilityConvergence(alpha)
490 # In order to see the graph whithout creating the associated files
491 #Show(convergenceGraphMonteCarlo)
492
493 # Create the file .EPS
494 convergenceGraphMonteCarlo.draw("convergenceGrapheMonteCarlo", 640, 480)
495 #Show(convergenceGraphMonteCarlo)
496
497
498 #####

```

```

499 # Directional Sampling
500 #####
501
502 print "#####"
503 print "Directional_Sampling"
504 print "#####"
505 print "_"
506
507 # Directional sampling from an event (slow and safe strategy by default)
508
509 # We create a Directional Sampling algorithm */
510 myAlgoDirectionalSim = DirectionalSampling(myEvent)
511 myAlgoDirectionalSim.setMaximumOuterSampling(maximumOuterSampling * blockSize)
512 myAlgoDirectionalSim.setBlockSize(1)
513 myAlgoDirectionalSim.setMaximumCoefficientOfVariation(coefficientOfVariation)
514
515 # Define the HistoryStrategy to store the numerical samples generated
516 # both for the input random vector and the output random vector
517 # Full strategy
518 myAlgoDirectionalSim.setInputOutputStrategy(HistoryStrategy(Full()))
519
520 # Define the HistoryStrategy to store the values of the probability estimator
521 # and the variance estimator
522 # used ot draw the convergence graph
523 # Full strategy
524 myAlgoDirectionalSim.setConvergenceStrategy(HistoryStrategy(Full()))
525
526 # Save the number of calls to the limit state fucntion, its gradient and hessian
    already done
527 deviationCallNumberBefore = deviation.getEvaluationCallsNumber()
528 deviationGradientCallNumberBefore = deviation.getGradientCallsNumber()
529 deviationHessianCallNumberBefore = deviation.getHessianCallsNumber()
530
531 # Perform the simulation */
532 myAlgoDirectionalSim.run()
533
534 # Save the number of calls to the limit state fucntion, its gradient and hessian
    already done
535 deviationCallNumberAfter = deviation.getEvaluationCallsNumber()
536 deviationGradientCallNumberAfter = deviation.getGradientCallsNumber()
537 deviationHessianCallNumberAfter = deviation.getHessianCallsNumber()
538
539 # Display number of iterations and number of evaluations
540 # of the limit state function
541 print "Number_of_evaluations_of_the_limit_state_function =_",
    deviationCallNumberAfter - deviationCallNumberBefore
542
543 # Display the Directional Simumation probability of 'myEvent'

```

```

544 print "Directional_Sampling_probability_estimation_=", myAlgoDirectionalSim.
      getResult().getProbabilityEstimate()
545
546 # Display the variance of the Directional Simulation probability estimator
547 print "Variance_of_the_Directional_Sampling_probability_estimator_=",
      myAlgoDirectionalSim.getResult().getVarianceEstimate()
548
549 # Display the confidence interval length centered around
550 # the Directional Simulation probability DSProb
551 # IC = [DSProb - 0.5*length, DSProb + 0.5*length]
552 # level 0.95
553 print "0.95_Confidence_Interval_="[, myAlgoDirectionalSim.getResult().
      getProbabilityEstimate() - 0.5*myAlgoDirectionalSim.getResult().
      getConfidenceLength(0.95), ", ", myAlgoDirectionalSim.getResult().
      getProbabilityEstimate() + 0.5*myAlgoDirectionalSim.getResult().
      getConfidenceLength(0.95), "]"
554 print ""
555
556
557 # Draw the convergence graph and the confidence interval of level alpha
558 alpha = 0.90
559 convergenceGraphDS = myAlgoDirectionalSim.drawProbabilityConvergence(alpha)
560 # In order to see the graph without creating the associated files
561 #Show(convergenceGraphDS)
562
563 # Create the file .EPS
564 convergenceGraphDS.draw("convergenceGrapheDS", 640, 480)
565 #Show(convergenceGraphDS)
566
567
568 #####
569 # Latin HyperCube Sampling
570 #####
571
572 print "#####"
573 print "Latin_HyperCube_Sampling"
574 print "#####"
575 print ""
576
577
578 # We create a LHS algorithm
579 myAlgoLHS = LHS(myEvent)
580 myAlgoLHS.setMaximumOuterSampling(maximumOuterSampling)
581 myAlgoLHS.setBlockSize(blockSize)
582 myAlgoLHS.setMaximumCoefficientOfVariation(coefficientOfVariation)
583
584 # Define the HistoryStrategy to store the numerical samples generated
585 # both for the input random vector and the output random vector

```

```

586 # Full strategy
587 myAlgoLHS.setInputOutputStrategy( HistoryStrategy( Full() ) )
588
589 # Define the HistoryStrategy to store the values of the probability estimator
590 # and the variance estimator
591 # used to draw the convergence graph
592 # Full strategy
593 myAlgoLHS.setConvergenceStrategy( HistoryStrategy( Full() ) )
594
595 # Perform the simulation
596 myAlgoLHS.run()
597
598 # Display number of iterations and number of evaluations
599 # of the limit state function
600 print "Number_of_evaluations_of_the_limit_state_function = ", myAlgoLHS.
    getResult().getOuterSampling()*myAlgoLHS.getResult().getBlockSize()
601
602 # Display the LHS probability of {\itshape myEvent}
603 print "LHS_probability_estimation = ", myAlgoLHS.getResult().
    getProbabilityEstimate()
604
605 # Display the variance of the LHS probability estimator
606 print "Variance_of_the_LHS_probability_estimator = ", myAlgoLHS.getResult().
    getVarianceEstimate()
607
608 # Display the confidence interval length centered around the LHS probability
    LHSProb
609 # IC = [LHSProb - 0.5*length, LHSProb + 0.5*length]
610 # level 0.95
611 print "0.95_Confidence_Interval = [" , myAlgoLHS.getResult().
    getProbabilityEstimate() - 0.5*myAlgoLHS.getResult().getConfidenceLength
    (0.95), ", " , myAlgoLHS.getResult().getProbabilityEstimate() + 0.5*myAlgoLHS.
    getResult().getConfidenceLength(0.95), "]"
612 print ""
613
614 # Draw the convergence graph and the confidence interval of level alpha
615 alpha = 0.90
616 convergenceGraphLHS = myAlgoLHS.drawProbabilityConvergence(alpha)
617 # In order to see the graph without creating the associated files
618 #Show(convergenceGraphLHS)
619
620 # Create the file .EPS
621 convergenceGraphLHS.draw("convergenceGrapheLHS", 640, 480)
622 #Show(convergenceGraphLHS)
623
624 #####
625 # Importance Sampling
626 #####

```

```

627
628
629 print "#####"
630 print "Importance_Sampling"
631 print "#####"
632 print ""
633
634 maximumOuterSampling = 40000
635 blockSize = 1
636 standardSpaceDesignPoint = resultFORM.getStandardSpaceDesignPoint()
637 mean = standardSpaceDesignPoint
638 sigma = NumericalPoint(4, 1.0)
639 importanceDistribution = Normal(mean, sigma, CorrelationMatrix(4))
640
641 myStandardEvent = StandardEvent(myEvent)
642
643 myAlgoImportanceSampling = ImportanceSampling(myStandardEvent, Distribution(
        importanceDistribution))
644 myAlgoImportanceSampling.setMaximumOuterSampling(maximumOuterSampling)
645 myAlgoImportanceSampling.setBlockSize(blockSize)
646 myAlgoImportanceSampling.setMaximumCoefficientOfVariation(coefficientOfVariation
        )
647
648 # Define the HistoryStrategy to store the numerical samples generated
649 # both for the input random vector and the output random vector
650 # Full strategy
651 myAlgoImportanceSampling.setInputOutputStrategy(HistoryStrategy(Full()))
652
653 # Define the HistoryStrategy to store the values of the probability estimator
654 # and the variance estimator
655 # used ot draw the convergence graph
656 # Full strategy
657 myAlgoImportanceSampling.setConvergenceStrategy(HistoryStrategy(Full()))
658
659 # Perform the simulation
660 myAlgoImportanceSampling.run()
661
662 # Display number of iterations and number of evaluations
663 # of the limit state function
664 print "Number_of_evaluations_of_the_limit_state_function_=_",
        myAlgoImportanceSampling.getResult().getOuterSampling()*
        myAlgoImportanceSampling.getResult().getBlockSize()
665
666 # Display the Importance Sampling probability of 'myEvent'
667 print "Importance_Sampling_probability_estimation_=_", myAlgoImportanceSampling.
        getResult().getProbabilityEstimate()
668
669 # Display the variance of the Importance Sampling probability estimator

```

```

670 print "Variance_of_the_Importance_Sampling_probability_estimator_=",
      myAlgoImportanceSampling.getResult().getVarianceEstimate()
671
672 # Display the confidence interval length centered around
673 # the ImportanceSampling probability ISProb
674 # IC = [ISProb - 0.5*length, ISProb + 0.5*length]
675 # level 0.95
676 print "0.95_Confidence_Interval_=", [", myAlgoImportanceSampling.getResult().
      getProbabilityEstimate() - 0.5*myAlgoImportanceSampling.getResult().
      getConfidenceLength(0.95), ",_", myAlgoImportanceSampling.getResult().
      getProbabilityEstimate() + 0.5*myAlgoImportanceSampling.getResult().
      getConfidenceLength(0.95), "]"
677
678 # Draw the convergence graph and the confidence interval of level alpha
679 alpha = 0.90
680 convergenceGraphIS = myAlgoImportanceSampling.drawProbabilityConvergence(alpha)
681 # In order to see the graph without creating the associated files
682 #Show(convergenceGraphIS)
683
684 # Create the file .EPS
685 convergenceGraphIS.draw("convergenceGrapheIS", 640, 480)
686 #Show(convergenceGraphIS)
687
688
689
690
691 #####
692 # Response surface : Polynomial expansion chaos
693 #####
694
695 print "#####"
696 print "Polynomial_expansion_chaos"
697 print "#####"
698 print "_"
699
700 #####
701 # STEP 1 : Construction of the multivariate orthonormal basis
702
703 # Dimension of the input random vector
704 dim = 4
705
706 # Create the univariate polynomial family collection
707 # which regroups the polynomial families for each direction
708 polyColl = PolynomialFamilyCollection(dim)
709
710 # Variable E
711 #Jacobi(alpha, beta) <=> Beta(\beta + 1, \alpha + \beta + 2, -1, 1)
712 alphaJ = 1.27

```

```

713 betaJ = -0.07
714 jacobiFamily = JacobiFactory(alphaJ, betaJ)
715 polyColl[0] = OrthogonalUniVariatePolynomialFamily(jacobiFamily)
716
717
718 # Variable F
719 # Laguerre(k) <=> Gamma(k+1,1,0) (parametrage ppal)
720 kLaguerre = 1.78
721 laguerreFamily = LaguerreFactory(kLaguerre)
722 polyColl[1] = OrthogonalUniVariatePolynomialFamily(laguerreFamily)
723
724 # Variable L
725 # Legendre <=> Unif(-1,1)
726 legendreFamily = LegendreFactory()
727 polyColl[2] = OrthogonalUniVariatePolynomialFamily(legendreFamily)
728
729 # Variable E
730 # Jacobi(alpha, beta) <=> Beta(\beta + 1, \alpha + \beta + 2, -1, 1)
731 alphaJ2 = 0.5
732 betaJ2 = 1.5
733 jacobiFamily2 = JacobiFactory(alphaJ2, betaJ2)
734 polyColl[3] = OrthogonalUniVariatePolynomialFamily(jacobiFamily2)
735
736
737 # Create the multivariate orthonormal basis
738 # which is the the cartesian product of the univariate basis
739 multivariateBasis = OrthogonalProductPolynomialFactory(polyColl,
    LinearEnumerateFunction(dim))
740
741 # Build a term of the basis as a NumericalMathFunction
742 # Generally, we do not need to construct manually any term,
743 # all terms are constructed automatically by a strategy of construction of the
    basis
744 i=5
745 Psi_i = multivariateBasis.build(i)
746
747 # Get the measure mu associated to the multivariate basis
748 distributionMu = multivariateBasis.getMeasure()
749
750 #####
751 # STEP 2 : Truncature strategy of the multivariate orthonormal basis
752
753 # CleaningStrategy :
754 # among the maximumConsideredTerms = 500 first polynoms,
755 # those which have the mostSignificant = 50 most significant contributions
756 # with significance criterion significanceFactor = 10^(-4)
757 # The True boolean indicates if we are interested
758 # in the online monitoring of the current basis update

```

```
759 # (removed or added coefficients)
760 maximumConsideredTerms = 500
761 mostSignificant = 50
762 significanceFactor = 1.0e-4
763 truncatureBasisStrategy = CleaningStrategy(OrthogonalBasis(multivariateBasis),
      maximumConsideredTerms, mostSignificant, significanceFactor, True)
764 truncatureBasisStrategy = FixedStrategy(OrthogonalBasis(multivariateBasis),481)
765
766 #####
767 # STEP 3 : Evaluation strategy of the approximation coefficients
768
769 # The technique proposed is the Least Squares technique
770 # where the points come from an experiment plane
771 # Here : the Monte Carlo sampling technique
772 sampleSize = 10000
773 evaluationCoeffStrategy = LeastSquaresStrategy(MonteCarloExperiment(sampleSize))
774
775 # STEP 4 : Creation of the Functional Chaos Algorithm
776
777 # FunctionalChaosAlgorithm :
778 # combination of the model : limitStateFunction
779 # the distribution of the input random vector : Xdistribution
780 # the truncature strategy of the multivariate basis
781 # and the evaluation strategy of the coefficients
782 polynomialChaosAlgorithm = FunctionalChaosAlgorithm(deviation, Distribution(
      inputDistribution), AdaptiveStrategy(truncatureBasisStrategy),
      ProjectionStrategy(evaluationCoeffStrategy))
783
784 #####
785 # Run and results exploitation
786
787 # Perform the simulation
788 polynomialChaosAlgorithm.run()
789
790 # Stream out the result
791 polynomialChaosResult = polynomialChaosAlgorithm.getResult()
792
793 # Get the polynomial chaos coefficients
794 coefficients = polynomialChaosResult.getCoefficients()
795
796 # Get the meta model as a NumericalMathFunction
797 metaModel = polynomialChaosResult.getMetaModel()
798
799 # Get the indices of the selected polynomials : K
800 subsetK = polynomialChaosResult.getIndices()
801
802 # Get the composition of the polynomials
803 # of the truncated multivariate basis
```

```

804 for i in range(subsetK.getSize()) :
805     print "Polynomial_number", i, "in_truncated_basis<->polynomial_number",
           subsetK[i], "=", LinearEnumerateFunction(dim)(subsetK[i]), "in_complete_
           basis"
806     print ""
807
808     # Get the multivariate basis
809     # as a collection of NumericalMathFunction
810     multivariateBasisCollection = polynomialChaosResult.getReducedBasis()
811
812     # Get the distribution of variables Z
813     mu = polynomialChaosResult.getDistribution()
814     print "Distribution_in_the_transformed_variables =", mu
815     print ""
816
817     # Get the composed model which is the model of the reduced variables Z
818     composedModel = polynomialChaosResult.getComposedModel()
819
820     # Define the new random vector
821     newOutputVariableOfInterest = RandomVector(polynomialChaosResult)
822
823     # Get the mean and variance of the meta model
824
825     print "Mean =", newOutputVariableOfInterest.getMean()
826     print "Standard_deviation =", sqrt(newOutputVariableOfInterest.getCovariance()
           [0,0])
827     print ""
828
829
830
831     #####
832     # Graphs validation
833
834
835     # Graph 1 : cloud
836
837     # Generate a NumericalSample of the input random vector
838     # Evaluate the meta model and the real model
839     # draw the clouds (metamodel, real model)
840     # Verify points are on the first diagonal
841     sizeX = 500
842     Xsample = inputDistribution.getNumericalSample(sizeX)
843
844     modelSample = deviation(Xsample)
845     metaModelSample = metaModel(Xsample)
846
847     sampleMixed = NumericalSample(sizeX,2)
848     for i in range(sizeX) :

```

```

849     sampleMixed[i][0] = modelSample[i][0]
850     sampleMixed[i][1] = metaModelSample[i][0]
851
852     legend = str(sizeX) + "_realisations"
853     comparisonCloud = Cloud(sampleMixed, "blue", "fsquare", legend)
854     graphCloud = Graph("Polynomial_chaos_expansion", "model", "meta_model", True, "
        topleft")
855     graphCloud.add(comparisonCloud)
856
857     #Show(graphCloud)
858     graphCloud.draw("PCE_comparisonModels")
859
860
861     # Graph 2 : polynoms family graphs
862
863     degreeMax = 5
864     pointNumber = 101
865     colorList = Drawable.GetValidColors()
866
867     # Jacobi for E
868     xMinJacobi = -1
869     xMaxJacobi = 1
870     titleJacobi = "Jacobi(" + str(alphaJ) + ",_" + str(betaJ) + ")_polynomials"
871     graphJacobi = Graph(titleJacobi, "z", "polynomial_values", True, "topleft")
872     for i in range(degreeMax) :
873         graphJacobi_temp = jacobiFamily.build(i).draw(xMinJacobi, xMaxJacobi,
            pointNumber)
874         graphJacobi_temp_draw = graphJacobi_temp.getDrawable(0)
875         legend = "degree_" + str(i)
876         graphJacobi_temp_draw.setLegendName(legend)
877         graphJacobi_temp_draw.setColor(colorList[i])
878         graphJacobi.add(graphJacobi_temp_draw)
879     #Show(graphJacobi)
880     graphJacobi.draw("PCE_JacobiPolynomials_VariableE")
881
882     # Laguerre for F
883     xMinLaguerre = 0
884     xMaxLaguerre = 10
885     titleLaguerre = "Laguerre(" + str(kLaguerre) + ")_polynomials"
886     graphLaguerre = Graph(titleLaguerre, "z", "polynomial_values", True, "topleft")
887     for i in range(degreeMax) :
888         graphLaguerre_temp = laguerreFamily.build(i).draw(xMinLaguerre, xMaxLaguerre,
            pointNumber)
889         graphLaguerre_temp_draw = graphLaguerre_temp.getDrawable(0)
890         legend = "degree_" + str(i)
891         graphLaguerre_temp_draw.setLegendName(legend)
892         graphLaguerre_temp_draw.setColor(colorList[i])
893         graphLaguerre.add(graphLaguerre_temp_draw)

```

```

894 #Show(graphLaguerre)
895 graphLaguerre.draw("PCE_LaguerrePolynomials_VariableF")
896
897 # Legendre for L
898 xMinLegendre = -1
899 xMaxLegendre = 1
900 titleLegendre = "Legendre_polynomials"
901 graphLegendre = Graph(titleLegendre, "z", "polynomial_values", True, "topright")
902 for i in range(degreeMax) :
903     graphLegendre_temp = laguerreFamily.build(i).draw(xMinLegendre, xMaxLegendre,
904         pointNumber)
905     graphLegendre_temp_draw = graphLegendre_temp.getDrawable(0)
906     legend = "degree_" + str(i)
907     graphLegendre_temp_draw.setLegendName(legend)
908     graphLegendre_temp_draw.setColor(colorList[i])
909     graphLegendre.add(graphLegendre_temp_draw)
910 #Show(graphLegendre)
911 graphLegendre.draw("PCE_LegendrePolynomials_VariableL")
912
913 # Jacobi for I
914 xMinJacobi2 = -1
915 xMaxJacobi2 = 1
916 titleJacobi2 = "Jacobi(" + str(alphaJ2) + ",_" + str(betaJ2) + ")_polynomials"
917 graphJacobi2 = Graph(titleJacobi2, "z", "polynomial_values", True, "topright")
918 for i in range(degreeMax) :
919     graphJacobi2_temp = jacobiFamily2.build(i).draw(xMinJacobi2, xMaxJacobi2,
920         pointNumber)
921     graphJacobi2_temp_draw = graphJacobi2_temp.getDrawable(0)
922     legend = "degree_" + str(i)
923     graphJacobi2_temp_draw.setLegendName(legend)
924     graphJacobi2_temp_draw.setColor(colorList[i])
925     graphJacobi2.add(graphJacobi2_temp_draw)
926 #Show(graphJacobi2)
927 graphJacobi2.draw("PCE_JacobiPolynomials_VariableI")

```

1.8 Output of the Python script

```

1 #####
2 Min/Max study with deterministic experiment plane
3 #####
4 From a composite experiment plane of size = 73
5 Levels = 0.5 , 1.0 , 3.0
6 Min Value = 0.649717975365
7 Max Value = 55.3605185131
8
9 #####
10 Min/Max study by random sampling

```

```
11 #####
12 From random sampling = 10000
13 Min Value = 5.38682360207
14 Max Value = 52.7553885642
15
16
17 #####
18 Random Study : central tendance of
19 the output variable of interest
20 #####
21
22 #####
23 Taylor variance decomposition
24 #####
25
26 First order mean= 12.3369023123
27 Evaluation calls number = 1
28 Second order mean= 12.4198129769
29 Evaluation calls number = 33
30 Standard deviation= 4.18703072295
31 Evaluation calls number = 8
32 Importance factors=
33 E = 0.149096880954
34 F = 0.781344650861
35 L = 0.0145457110918
36 I = 0.0550127570932
37
38 #####
39 Random sampling
40 #####
41 Sample size = 10000
42 Mean from sample = 12.609042535
43 Standard deviation from sample = 4.35926575014
44
45 #####
46 # Kernel Smoothing Fitting
47 #####
48 Sample size = 10000
49 Kernel bandwidth= 0.688228112153
50 Mean from kernel smoothing = 12.619538853
51
52
53 #####
54 Probabilistic Study : threshold exceedance: deviation <-1cm
55 #####
56
57 #####
58 FORM
```

```
59 #####
60 FORM event probability= 0.00670980421088
61 Number of evaluations of the limit state function = 176
62 Generalized reliability index= 2.47243508163
63 Standard space design point=
64 E = -0.602386403812
65 F = 2.31055515463
66 L = 0.355793665542
67 I = -0.533677429099
68 Physical space design point=
69 E = 30327158.555
70 F = 61318.4694411
71 L = 256.390024534
72 I = 378.634729684
73 Importance factors=
74 E = 0.058682003115
75 F = 0.863350794277
76 L = 0.0204715646194
77 I = 0.0574956379882
78 Hasofer reliability index= 2.47243508163
79
80 #####
81 Monte Carlo
82 #####
83
84 Number of evaluations of the limit state function = 18300
85 Monte Carlo probability estimation = 0.00551912568306
86 Variance of the Monte Carlo probability estimator = 3.02926673715e-07
87 0.95 Confidence Interval = [ 0.00444038551844 , 0.00659786584768 ]
88
89 #####
90 Directional Sampling
91 #####
92 Number of evaluations of the limit state function = 18258
93 Directional Sampling probability estimation = 0.0048902194515
94 Variance of the Directional Sampling probability estimator = 2.39120163922e-07
95 0.95 Confidence Interval = [ 0.0039317987386 , 0.00584864016439 ]
96
97 #####
98 Latin HyperCube Sampling
99 #####
100 Number of evaluations of the limit state function = 20600
101 LHS probability estimation = 0.00490291262136
102 Variance of the LHS probability estimator = 2.39206700255e-07
103 0.95 Confidence Interval = [ 0.00394431850044 , 0.00586150674228 ]
104
105 #####
106 Importance Sampling
```

```

107 #####
108 Number of evaluations of the limit state function = 306
109 Importance Sampling probability estimation = 0.00658159419011
110 Variance of the Importance Sampling probability estimator = 4.29506441412e-07
111 0.95 Confidence Interval = [ 0.00529709767088 , 0.00786609070933 ]
112
113
114 #####
115 Polynomial expansion chaos
116 #####
117
118 Polynomial number 0 in truncated basis <=> polynomial number 0 = [0,0,0,0]
    in complete basis
119 Polynomial number 1 in truncated basis <=> polynomial number 1 = [1,0,0,0]
    in complete basis
120 Polynomial number 2 in truncated basis <=> polynomial number 2 = [0,1,0,0]
    in complete basis
121 Polynomial number 3 in truncated basis <=> polynomial number 3 = [0,0,1,0]
    in complete basis
122 Polynomial number 4 in truncated basis <=> polynomial number 4 = [0,0,0,1]
    in complete basis
123 Polynomial number 5 in truncated basis <=> polynomial number 5 = [2,0,0,0]
    in complete basis
124 Polynomial number 6 in truncated basis <=> polynomial number 6 = [1,1,0,0]
    in complete basis
125 Polynomial number 7 in truncated basis <=> polynomial number 7 = [1,0,1,0]
    in complete basis
126 Polynomial number 8 in truncated basis <=> polynomial number 8 = [1,0,0,1]
    in complete basis
127 Polynomial number 9 in truncated basis <=> polynomial number 9 = [0,2,0,0]
    in complete basis
128 Polynomial number 10 in truncated basis <=> polynomial number 10 =
    [0,1,1,0] in complete basis
129 Polynomial number 11 in truncated basis <=> polynomial number 11 =
    [0,1,0,1] in complete basis
130 Polynomial number 12 in truncated basis <=> polynomial number 12 =
    [0,0,2,0] in complete basis
131 Polynomial number 13 in truncated basis <=> polynomial number 13 =
    [0,0,1,1] in complete basis
132 Polynomial number 14 in truncated basis <=> polynomial number 14 =
    [0,0,0,2] in complete basis
133 Polynomial number 15 in truncated basis <=> polynomial number 15 =
    [3,0,0,0] in complete basis
134 Polynomial number 16 in truncated basis <=> polynomial number 16 =
    [2,1,0,0] in complete basis
135 Polynomial number 17 in truncated basis <=> polynomial number 18 =
    [2,0,0,1] in complete basis

```

136	Polynomial number 18 in truncated basis \leftrightarrow polynomial number 19 = [1,2,0,0] in complete basis
137	Polynomial number 19 in truncated basis \leftrightarrow polynomial number 20 = [1,1,1,0] in complete basis
138	Polynomial number 20 in truncated basis \leftrightarrow polynomial number 26 = [0,2,1,0] in complete basis
139	Polynomial number 21 in truncated basis \leftrightarrow polynomial number 29 = [0,1,1,1] in complete basis
140	Polynomial number 22 in truncated basis \leftrightarrow polynomial number 30 = [0,1,0,2] in complete basis
141	Polynomial number 23 in truncated basis \leftrightarrow polynomial number 31 = [0,0,3,0] in complete basis
142	Polynomial number 24 in truncated basis \leftrightarrow polynomial number 33 = [0,0,1,2] in complete basis
143	Polynomial number 25 in truncated basis \leftrightarrow polynomial number 50 = [1,1,0,2] in complete basis
144	Polynomial number 26 in truncated basis \leftrightarrow polynomial number 56 = [0,3,1,0] in complete basis
145	Polynomial number 27 in truncated basis \leftrightarrow polynomial number 62 = [0,1,2,1] in complete basis
146	Polynomial number 28 in truncated basis \leftrightarrow polynomial number 64 = [0,1,0,3] in complete basis
147	Polynomial number 29 in truncated basis \leftrightarrow polynomial number 65 = [0,0,4,0] in complete basis
148	Polynomial number 30 in truncated basis \leftrightarrow polynomial number 105 = [0,5,0,0] in complete basis
149	Polynomial number 31 in truncated basis \leftrightarrow polynomial number 111 = [0,2,3,0] in complete basis
150	Polynomial number 32 in truncated basis \leftrightarrow polynomial number 120 = [0,0,5,0] in complete basis
151	Polynomial number 33 in truncated basis \leftrightarrow polynomial number 122 = [0,0,3,2] in complete basis
152	Polynomial number 34 in truncated basis \leftrightarrow polynomial number 131 = [4,1,1,0] in complete basis
153	Polynomial number 35 in truncated basis \leftrightarrow polynomial number 147 = [2,3,1,0] in complete basis
154	Polynomial number 36 in truncated basis \leftrightarrow polynomial number 150 = [2,2,1,1] in complete basis
155	Polynomial number 37 in truncated basis \leftrightarrow polynomial number 163 = [1,4,0,1] in complete basis
156	Polynomial number 38 in truncated basis \leftrightarrow polynomial number 238 = [3,1,1,2] in complete basis
157	Polynomial number 39 in truncated basis \leftrightarrow polynomial number 248 = [2,3,2,0] in complete basis
158	Polynomial number 40 in truncated basis \leftrightarrow polynomial number 249 = [2,3,1,1] in complete basis
159	Polynomial number 41 in truncated basis \leftrightarrow polynomial number 266 = [1,6,0,0] in complete basis

```

160 Polynomial number 42 in truncated basis <-> polynomial number 278 =
    [1,2,2,2] in complete basis
161 Polynomial number 43 in truncated basis <-> polynomial number 294 =
    [0,7,0,0] in complete basis
162 Polynomial number 44 in truncated basis <-> polynomial number 345 =
    [5,1,0,2] in complete basis
163 Polynomial number 45 in truncated basis <-> polynomial number 367 =
    [3,4,0,1] in complete basis
164 Polynomial number 46 in truncated basis <-> polynomial number 399 =
    [2,2,1,3] in complete basis
165 Polynomial number 47 in truncated basis <-> polynomial number 437 =
    [1,1,4,2] in complete basis
166 Polynomial number 48 in truncated basis <-> polynomial number 450 =
    [0,8,0,0] in complete basis
167 Polynomial number 49 in truncated basis <-> polynomial number 481 =
    [0,1,4,3] in complete basis
168
169 Distribution in the transformed variables = class=ComposedDistribution name=
    ComposedDistribution dimension=4 copula=class=IndependentCopula name=
    IndependentCopula dimension=4 marginal[0]=class=Beta name=Beta dimension=1 r
    =0.93 t=3.2 a=-1 b=1 marginal[1]=class=Gamma name=Gamma dimension=1 k=2.78
    lambda=1 gamma=0 marginal[2]=class=Uniform name=Uniform dimension=1 a=-1 b=1
    marginal[3]=class=Beta name=Beta dimension=1 r=2.5 t=4 a=-1 b=1
170
171 Mean = class=NumericalPoint name=Unnamed dimension=1 implementation=class=
    NumericalPointImplementation name=Unnamed dimension=1 values=[12.6138]
172 Standard deviation = 4.23422289923

```

1.9 Figures

The probability density function (PDF) of each marginal is given in Figures 2 to 5.

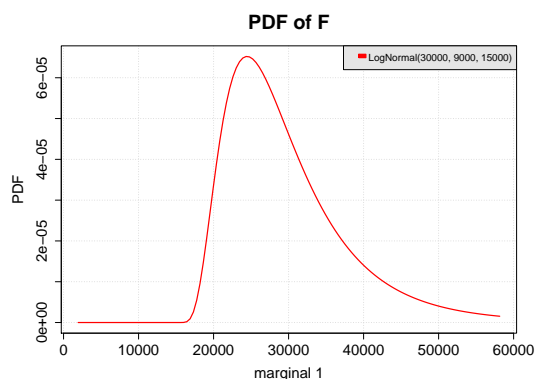
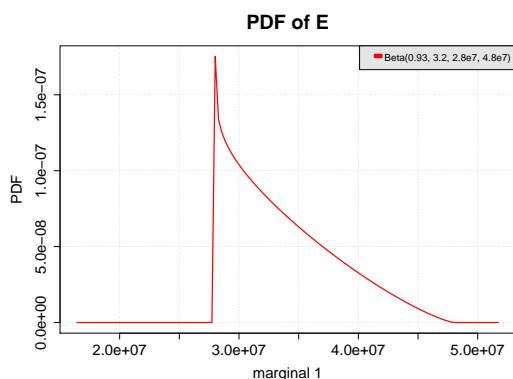


Figure 2: Probability density function of the parameter E Figure 3: Probability density function of the parameter F

The probability density function (PDF) and the cumulative density function (CDF) of the deviation fitted with the kernel smoothing method are drawn in Figures 6 and 7.

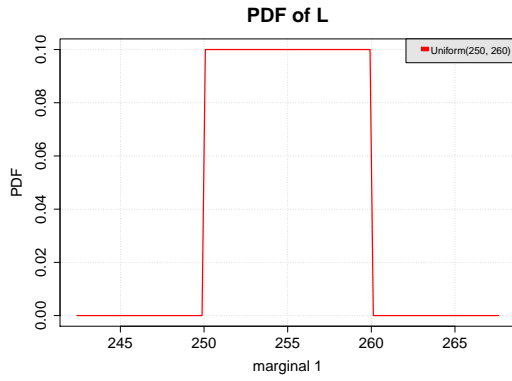


Figure 4: PDF of the parameter L

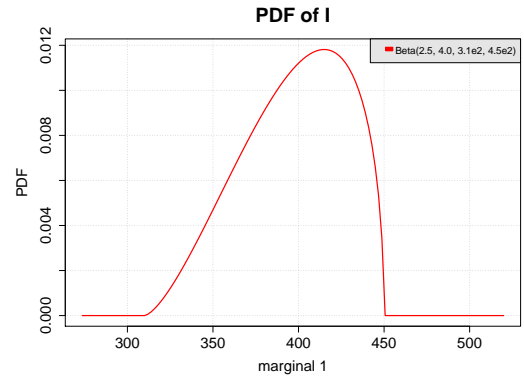


Figure 5: PDF of the parameter I

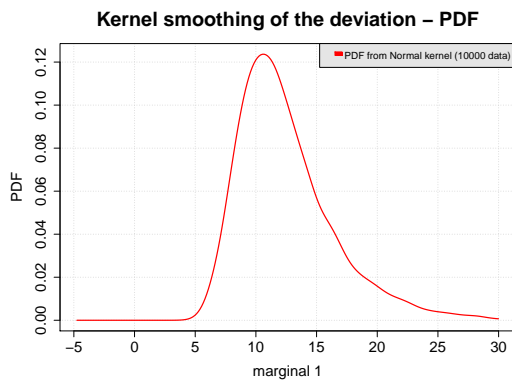


Figure 6: PDF of the deviation with the kernel smoothing method.

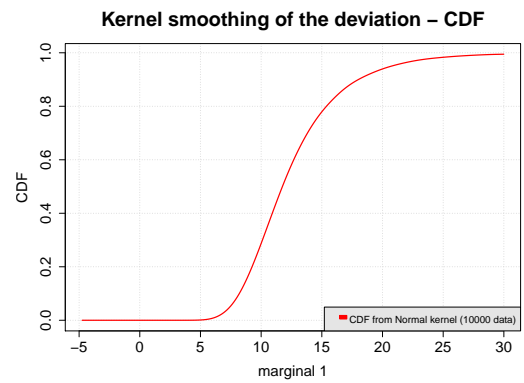


Figure 7: CDF of the deviation with the kernel smoothing method.

The superposition of the kernel smoothed density function and the normal fitted from the same sample with the maximum likelihood method is drawn in Figure 8.

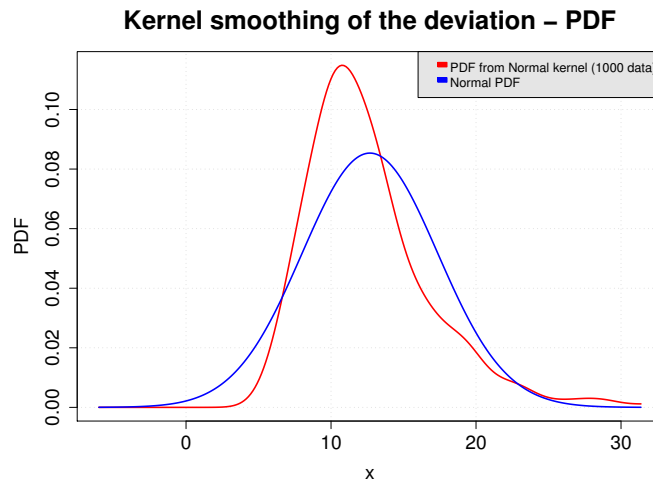


Figure 8: Superposition of the kernel smoothed density function and the normal fitted from the same sample.

The importance factors from the FORM method are given in Figure 9.

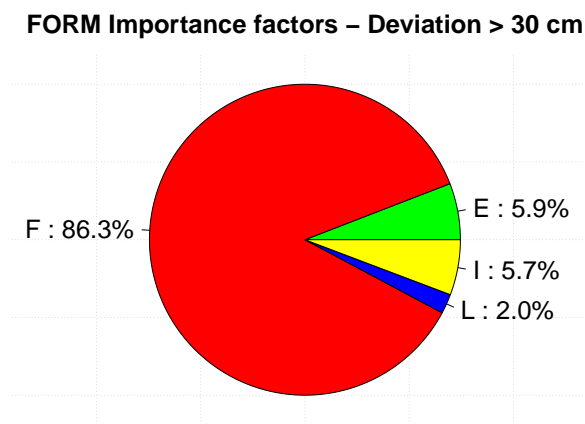


Figure 9: FORM importance factors of the event : deviation $>$ 30 cm.

The convergence graphs of the simulation methods are given in Figures 10 to 13.

Figures (14) to (18) contain the graphs :

- Graph 1 : the drawings of the fifth first members of the 1D polynomial family,
- Graph 2 : the cloud of points making the comparison between the model values and the meta model ones : if the adequation is perfect, points must be on the first diagonal.

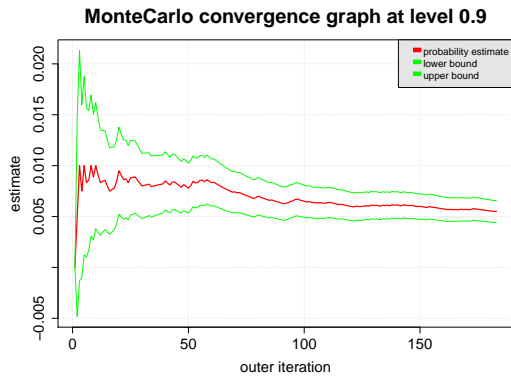


Figure 10: Monte Carlo convergence graph.

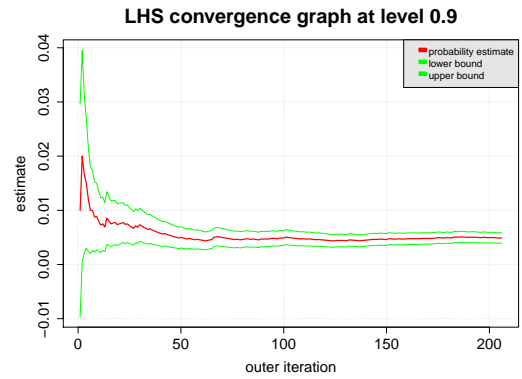


Figure 11: LHS convergence graph.

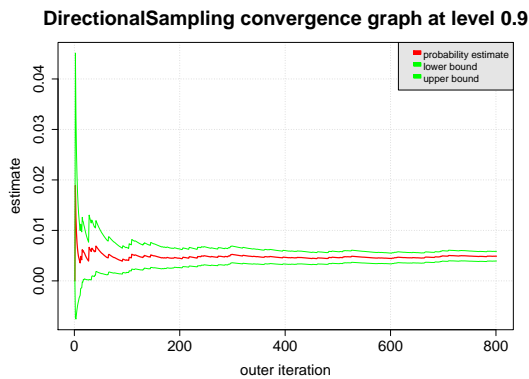


Figure 12: Directional Sampling convergence graph.

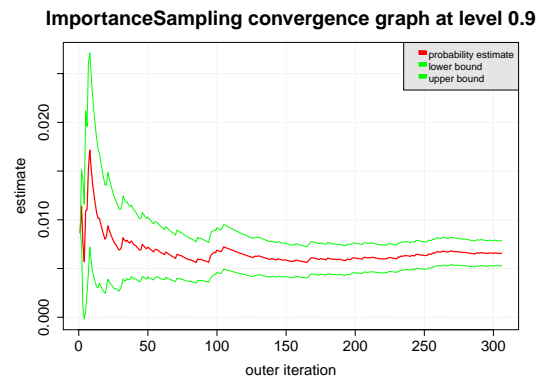


Figure 13: LHS convergence graph.

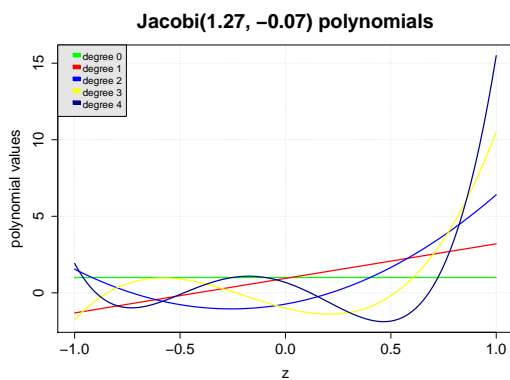


Figure 14: The 5-th first polynomials of the Jacobi family associated to the variable E.

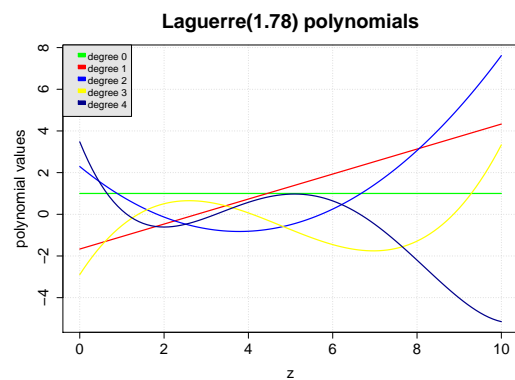


Figure 15: The 5-th first polynomials of the Laguerre family associated to the variable F.

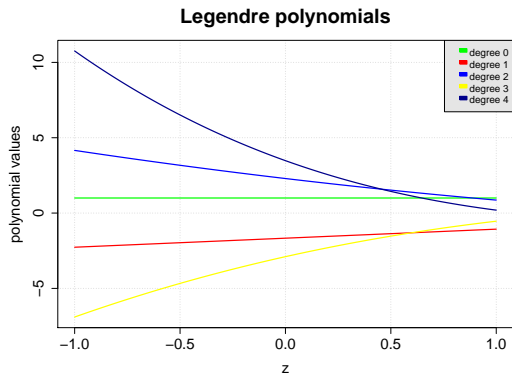


Figure 16: The 5-th first polynomials of the Legendre associated to the variable I.

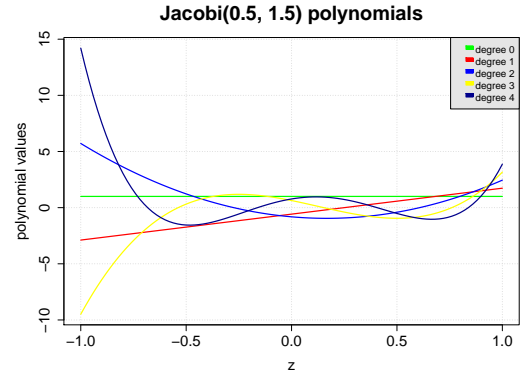


Figure 17: The 5-th first polynomials of the Jacobi family associated to the variable I.

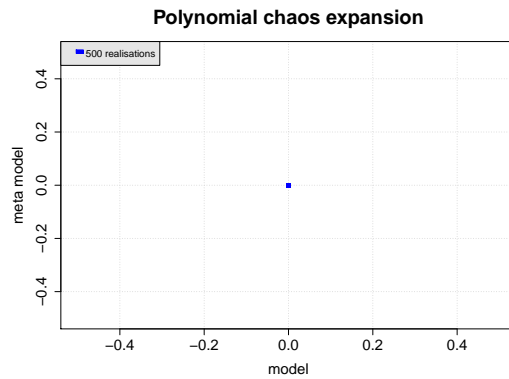


Figure 18: Comparison of values from the model and the polynomial chaos meta model.

1.10 Results comments

1.10.1 Min/Max approach

The Min/Max approach enables to evaluate the range of the deviation.

We note that the use of an experiment plane may be beneficial with regard the random sampling technique as we can catch more easily (which means with less evaluations of the limit state function) the extrem values of the output variable of interest : here, we have managed to catch both extrem bounds of the deviation with the composite experiment plane, whereas the random sampling technique did not manage to give a good evaluation of them.

Note that the composite experiment plane has 73 points, where as the random sampling technique has been effected with 10^4 points.

1.10.2 Central tendency approach

The Taylor variance decomposition has given a good approximation of the mean value of the deviation : the value is comparable to the one obtained with the random technique. Furthermore, note that the Taylor variance decomposition required only 1 evaluation of the limit state function, whereas the random sampling technique required 10^4 evaluations.

The second order evaluation of the mean by the Taylor variance decomposition method adds no information, which probably means that around the mean point of the input random vector, the limit state function is well approximated by its tangent plane.

The importance factors indicate that the mean of the deviation is mostly influenced by the uncertainty of the variable F.

The kernel smoothing technique enables to have a look on the distribution shape and another approximation of the mean value of the deviation.

Note that the normal fitting on the sample is not adapted.

1.10.3 Threshold exceedance approach

The whole event probabilities evaluated from the simulation methods are equivalent and confirm the event probability evaluated with FORM.

Note that the FORM probability required only 176 evaluations of the limit state function whereas the Monte Carlo probability required 17300 evaluations, the Directional Sampling one 17297 evaluations and the LHS one 20300 evaluations.

The Importance Sampling is a simulation method but the importance density has been centered around the design point, where the threshold exceedance is concentrated. That's why the succession of the FORM technique and the Importance sampling one where the importance density is a normal distribution centered around the design point, performed in the standard space, seems to be the better compromise between the limit state evaluation calls number and the probability evaluation precision.

The simulation methods give a confidence interval, which is not possible with FORM.

FORM ranks the influence of the input uncertainties on the realisation of the threshold exceedance event : the variable F is largely the more influent. Thus, if the threshold exceedance probability is judged too high, it is recommended to decrease the variability of the variable F first.

1.10.4 Response surface : Polynomial expansion chaos

The polynomial expansion chaos has defined a meta model thanks to 100 points, which gives very satisfactory results compared to those obtained with other methods.