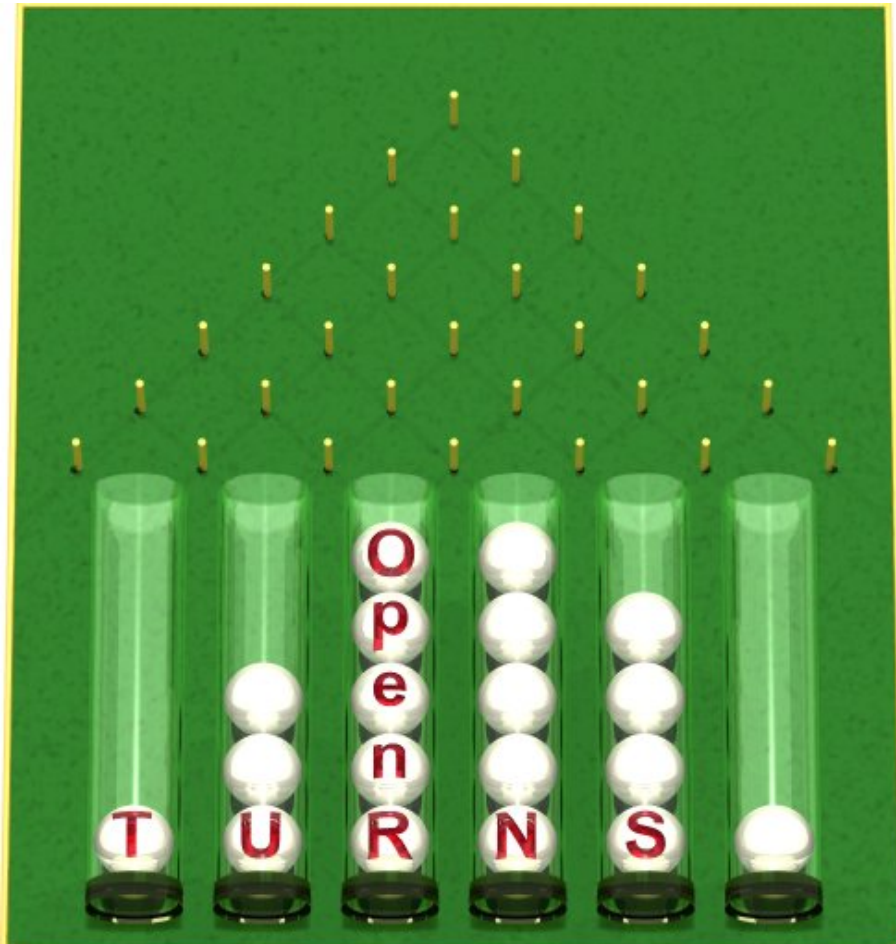


Examples Guide

Open TURNS version 1.0

Documentation built from package openturns-doc-April2012

April 20, 2012



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 design of experiments | 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 | 35 |
| 1.10.1 | Min/Max approach | 35 |
| 1.10.2 | Central tendency approach | 35 |
| 1.10.3 | Threshold exceedance approach | 35 |
| 1.10.4 | Response surface : Polynomial expansion chaos | 36 |
| 2 | Example 2: elastic truss structure | 37 |
| 2.1 | Problem statement | 37 |
| 2.1.1 | Physical model | 37 |
| 2.1.2 | Probabilistic model | 38 |
| 2.2 | Uncertainty and sensitivity analysis based on polynomial chaos expansions | 38 |
| 2.2.1 | Methodology to construct a sparse polynomial chaos approximation | 38 |
| 2.2.2 | Parametric study varying the degree of the metamodel | 39 |
| 2.2.3 | Second moments and sensitivity indices based on chaos coefficients | 40 |
| 2.2.4 | Distribution and higher-order moments analysis | 40 |
| 2.3 | Python scripts | 41 |
| 2.3.1 | Physical model – Physical_model.py | 41 |
| 2.3.2 | Probabilistic model – Proba_model.py | 43 |
| 2.3.3 | Uncertainty analysis based on PC expansions – main_SparsePolyChaos.py | 44 |

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 :

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

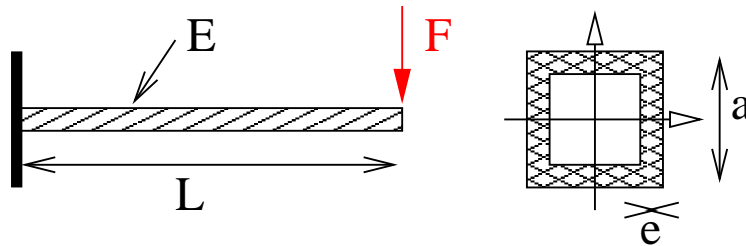


Figure 1: cantilever beam under a punctual bending load.

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 design of experiments ,
- with a random design of experiments ,
- 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 design of experiments

We consider a composite design of experiments , 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 space, 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 space.

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 design of experiments 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 inputDistribution.setDescription( ("E", "F", "L", "I") )
111
112 # Create the input random vector
113 inputRandomVector = RandomVector(inputDistribution)
114
115 # Create the output variable of interest
116 outputVariableOfInterest = RandomVector(deviation, inputRandomVector)
117
118
119 #####
120 ### Min/Max approach Study
121 #####
122
123
124 #####
125 # Min/Max study with deterministic design of experiment
126 #####
127
128 print "#####"

```

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

```

176
177 # Evaluate the range of the output variable of interest on that design of
      # experiments
178 minValue = outputSample.getMin()
179 maxValue = outputSample.getMax()
180
181 print "From a composite design of experiments of size =", pointNumber
182 print "Levels =", levels[0], ", ", levels[1], ", ", levels[2]
183 print "Min Value =", minValue[0]
184 print "Max Value =", maxValue[0]
185 print ""
186
187 #####
188 # Min/Max study by random sampling
189 #####
190
191 print "#####"
192 print " _Min/Max study by random sampling"
193 print "#####"
194
195 pointNumber = 10000
196 print "From random sampling =", pointNumber
197 outputSample2 = outputVariableOfInterest.getNumericalSample(pointNumber)
198
199 minValue2 = outputSample2.getMin()
200 maxValue2 = outputSample2.getMax()
201
202 print "Min Value =", minValue2[0]
203 print "Max Value =", maxValue2[0]
204 print ""
205
206
207
208
209
210 print ""
211 #####
212 ### Random Study : central tendance of
213 ### the output variable of interest
214 #####
215
216 print "#####"
217 print "Random Study : _central _tendance _of"
218 print "the _output _variable _of _interest"
219 print "#####"
220 print ""
221
222 #####

```

```

223 # Taylor variance decomposition
224 #####
225
226 print "#####"
227 print "Taylor_variance_decomposition"
228 print "#####"
229 print ""
230
231 # We create a quadraticCumul algorithm
232 myQuadraticCumul = QuadraticCumul(outputVariableOfInterest)
233
234 # We compute the several elements provided by the quadratic cumul algorithm
235 # and evaluate the number of calculus needed
236 nbBefr = deviation.getEvaluationCallsNumber()
237
238 # Mean first order
239 meanFirstOrder = myQuadraticCumul.getMeanFirstOrder()[0]
240 nbAfter1 = deviation.getEvaluationCallsNumber()
241
242 # Mean second order
243 meanSecondOrder = myQuadraticCumul.getMeanSecondOrder()[0]
244 nbAfter2 = deviation.getEvaluationCallsNumber()
245
246 # Standard deviation
247 stdDeviation = sqrt(myQuadraticCumul.getCovariance()[0,0])
248 nbAfter3 = deviation.getEvaluationCallsNumber()
249
250 print "First_order_mean=", myQuadraticCumul.getMeanFirstOrder()[0]
251 print "Evaluation_calls_number=", nbAfter1 - nbBefr
252 print "Second_order_mean=", myQuadraticCumul.getMeanSecondOrder()[0]
253 print "Evaluation_calls_number=", nbAfter2 - nbAfter1
254 print "Standard_deviation=", sqrt(myQuadraticCumul.getCovariance()[0,0])
255 print "Evaluation_calls_number=", nbAfter3 - nbAfter2
256
257 print "Importance_factors="
258 for i in range(inputRandomVector.getDimension()):
259     print inputDistribution.getDescription()[i], " ", myQuadraticCumul.
        getImportanceFactors()[i]
260 print ""
261
262
263 #####
264 # Random sampling
265 #####
266
267 print "#####"
268 print "Random_sampling"
269 print "#####"

```

```

270
271 size1 = 10000
272 output_Sample1 = outputVariableOfInterest.getNumericalSample(size1)
273 outputMean = output_Sample1.computeMean()
274 outputCovariance = output_Sample1.computeCovariance()
275
276 print "Sample_size=", size1
277 print "Mean_from_sample=", outputMean[0]
278 print "Standard_deviation_from_sample=", sqrt(outputCovariance[0,0])
279 print ""
280
281
282 #####
283 # Kernel Smoothing Fitting
284 #####
285
286
287 print "#####"
288 print "# Kernel Smoothing Fitting"
289 print "#####"
290
291 # We generate a sample of the output variable
292 size = 10000
293 output_sample = outputVariableOfInterest.getNumericalSample(size)
294
295 # We build the kernel smoothing distribution
296 kernel = KernelSmoothing()
297 bw = kernel.computeSilvermanBandwidth(output_sample)
298 smoothed = kernel.build(output_sample, bw)
299 print "Sample_size=", size
300 print "Kernel_bandwidth=", kernel.getBandwidth()[0]
301
302 # We draw the pdf and cdf from kernel smoothing
303 # Evaluate at best the range of the graph
304 mean_sample = output_sample.computeMean()[0]
305 standardDeviation_sample = sqrt(output_sample.computeCovariance()[0,0])
306 xmin = mean_sample - 4*standardDeviation_sample
307 xmax = mean_sample + 4*standardDeviation_sample
308
309 # Draw the PDF
310 smoothedPDF = smoothed.drawPDF(xmin, xmax, 251)
311 # Change the title
312 smoothedPDF.setTitle("Kernel_smoothing_of_the_deviation_-_PDF")
313 # Change the legend
314 smoothedPDF_draw = smoothedPDF.getDrawable(0)
315 title = "PDF_from_Normal_kernel_" + str(size) + "_data"
316 smoothedPDF_draw.setLegendName(title)
317 smoothedPDF.setDrawable(smoothedPDF_draw, 0)

```

```

318 smoothedPDF.draw("smoothedPDF", 640, 480)
319
320 # Draw the CDF
321 smoothedCDF = smoothed.drawCDF(xmin, xmax, 251)
322 # Change the title
323 smoothedCDF.setTitle("Kernel_smoothing_of_the_deviation_CDF")
324 # Change the legend
325 smoothedCDF_draw = smoothedCDF.getDrawable(0)
326 title = "CDF_from_Normal_kernel_" + str(size) + "_data"
327 smoothedCDF_draw.setLegendName(title)
328 smoothedCDF.setDrawable(smoothedCDF_draw, 0)
329 # Change the legend position
330 smoothedCDF.setLegendPosition("bottomright")
331 smoothedCDF.draw("smoothedCDF", 640, 480)
332
333 # In order to see the graph without creating the associated files
334 #Show(smoothedCDF)
335 #Show(smoothedPDF)
336
337 # Mean of the output variable of interest
338 print "Mean_from_kernel_smoothing=", smoothed.getMean()[0]
339 print ""
340
341 # Superposition of the kernel smoothing pdf and the gaussian one
342 # which mean and standard deviation are those of the output_sample
343 normalDist = NormalFactory().build(output_sample)
344 normalDistPDF = normalDist.drawPDF(xmin, xmax, 251)
345 normalDistPDFDrawable = normalDistPDF.getDrawable(0)
346 normalDistPDFDrawable.setColor('blue')
347 smoothedPDF.add(normalDistPDFDrawable)
348 smoothedPDF.draw("smoothedPDF_and_NormalPDF", 640, 480)
349
350 # In order to see the graph without creating the associated files
351 #Show(smoothedPDF)
352
353 #####
354 ### Probabilistic Study : threshold exceedance: deviation > 30cm
355 #####
356
357 print ""
358 print "#####"
359 print " Probabilistic _Study_: _threshold _exceedance:_deviation <-1cm"
360 print "#####"
361 print ""
362
363 #####
364 # FORM
365 #####

```

```

366
367 print "#####"
368 print "FORM"
369 print "#####"
370
371 # We create an Event from this RandomVector
372 # threshold has been defined in the kernel smoothing section
373 threshold = 30
374 myEvent = Event(outputVariableOfInterest , ComparisonOperator(Greater() ,
    threshold)
375 myEvent.setName(" Deviation >_30_cm")
376
377 # We create a NearestPoint algorithm
378 myCobyla = Cobyla()
379 myCobyla.setMaximumIterationsNumber(1000)
380 myCobyla.setMaximumAbsoluteError(1.0e-10)
381 myCobyla.setMaximumRelativeError(1.0e-10)
382 myCobyla.setMaximumResidualError(1.0e-10)
383 myCobyla.setMaximumConstraintError(1.0e-10)
384
385 # We create a FORM algorithm
386 # The first parameter is a NearestPointAlgorithm
387 # The second parameter is an event
388 # The third parameter is a starting point for the design point research
389 meanVector = inputRandomVector.getMean()
390 myAlgoFORM = FORM(NearestPointAlgorithm(myCobyla) , myEvent , meanVector)
391
392 # Get the number of times the limit state function has been evaluated so far
393 deviationCallNumberBeforeFORM = deviation.getEvaluationCallsNumber()
394
395 # Perform the simulation
396 myAlgoFORM.run()
397
398 # Get the number of times the limit state function has been evaluated so far
399 deviationCallNumberAfterFORM = deviation.getEvaluationCallsNumber()
400
401 # Stream out the result
402 resultFORM = myAlgoFORM.getResult()
403 print "FORM_event_probability=" , resultFORM.getEventProbability()
404 print "Number_of_evaluations_of_the_limit_state_function_=" ,
    deviationCallNumberAfterFORM - deviationCallNumberBeforeFORM
405 print " Generalized_reliability_index=" , resultFORM.
    getGeneralisedReliabilityIndex()
406 print " Standard_space_design_point="
407 for i in range(inputRandomVector.getDimension()) :
408     print inputDistribution.getDescription()[i] , " _=" , resultFORM.
        getStandardSpaceDesignPoint()[i]
409 print " Physical_space_design_point="

```

```

410 for i in range(inputRandomVector.getDimension()) :
411     print inputDistribution.getDescription()[i], " = ", resultFORM.
        getPhysicalSpaceDesignPoint()[i]
412
413     print "Importance_factors="
414     for i in range(inputRandomVector.getDimension()) :
415         print inputDistribution.getDescription()[i], " = ", resultFORM.
            getImportanceFactors()[i]
416
417     print "Hasofer_reliability_index=" , resultFORM.getHasoferReliabilityIndex()
418     print ""
419
420 # Graph 1 : Importance Factors graph */
421 importanceFactorsGraph = resultFORM.drawImportanceFactors()
422 title = "FORM_Importance_factors_-" + myEvent.getName()
423 importanceFactorsGraph.setTitle( title )
424 importanceFactorsGraph.draw("ImportanceFactorsDrawingFORM", 640, 480)
425
426 # In order to see the graph without creating the associated files
427 #Show(importanceFactorsGraph)
428
429
430 #####
431 # MC
432 #####
433
434 print "#####"
435 print "Monte_Carlo"
436 print "#####"
437 print ""
438
439
440 maximumOuterSampling = 40000
441 blockSize = 100
442 coefficientOfVariation = 0.10
443
444 # We create a Monte Carlo algorithm
445 myAlgoMonteCarlo = MonteCarlo(myEvent)
446 myAlgoMonteCarlo.setMaximumOuterSampling(maximumOuterSampling)
447 myAlgoMonteCarlo.setBlockSize(blockSize)
448 myAlgoMonteCarlo.setMaximumCoefficientOfVariation(coefficientOfVariation)
449
450 # Define the HistoryStrategy to store the values of the probability estimator
451 # and the variance estimator
452 # used ot draw the convergence graph
453 # Full strategy
454 myAlgoMonteCarlo.setConvergenceStrategy(HistoryStrategy(Full()))
455

```

```

456 # Perform the simulation
457 myAlgoMonteCarlo.run()
458
459 # Display number of iterations and number of evaluations
460 # of the limit state function
461 print "Number_of_evaluations_of_the_limit_state_function = ", myAlgoMonteCarlo.
      getResult().getOuterSampling()* myAlgoMonteCarlo.getResult().getBlockSize()
462
463 # Display the Monte Carlo probability of 'myEvent'
464 print "Monte_Carlo_probability_estimation = ", myAlgoMonteCarlo.getResult().
      getProbabilityEstimate()
465
466 # Display the variance of the Monte Carlo probability estimator
467 print "Variance_of_the_Monte_Carlo_probability_estimator = ", myAlgoMonteCarlo.
      getResult().getVarianceEstimate()
468
469 # Display the confidence interval length centered around
470 # the MonteCarlo probability MCTProb
471 # IC = [MCTProb - 0.5*length, MCTProb + 0.5*length]
472 # level 0.95
473
474 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), "]"
475 print ""
476
477 # Draw the convergence graph and the confidence interval of level alpha
478 alpha = 0.90
479 convergenceGraphMonteCarlo = myAlgoMonteCarlo.drawProbabilityConvergence(alpha)
480 # In order to see the graph without creating the associated files
481 #Show(convergenceGraphMonteCarlo)
482
483 # Create the file .EPS
484 convergenceGraphMonteCarlo.draw("convergenceGrapheMonteCarlo", 640, 480)
485 #Show(convergenceGraphMonteCarlo)
486
487
488 #####
489 # Directional Sampling
490 #####
491
492 print "#####"
493 print "Directional_Sampling"
494 print "#####"
495 print "_"
496

```

```

497 # Directional sampling from an event (slow and safe strategy by default)
498
499 # We create a Directional Sampling algorithm */
500 myAlgoDirectionalSim = DirectionalSampling(myEvent)
501 myAlgoDirectionalSim.setMaximumOuterSampling(maximumOuterSampling * blockSize)
502 myAlgoDirectionalSim.setBlockSize(1)
503 myAlgoDirectionalSim.setMaximumCoefficientOfVariation(coefficientOfVariation)
504
505 # Define the HistoryStrategy to store the values of the probability estimator
506 # and the variance estimator
507 # used to draw the convergence graph
508 # Full strategy
509 myAlgoDirectionalSim.setConvergenceStrategy(HistoryStrategy(Full()))
510
511 # Save the number of calls to the limit state function, its gradient and hessian
    already done
512 deviationCallNumberBefore = deviation.getEvaluationCallsNumber()
513 deviationGradientCallNumberBefore = deviation.getGradientCallsNumber()
514 deviationHessianCallNumberBefore = deviation.getHessianCallsNumber()
515
516 # Perform the simulation */
517 myAlgoDirectionalSim.run()
518
519 # Save the number of calls to the limit state function, its gradient and hessian
    already done
520 deviationCallNumberAfter = deviation.getEvaluationCallsNumber()
521 deviationGradientCallNumberAfter = deviation.getGradientCallsNumber()
522 deviationHessianCallNumberAfter = deviation.getHessianCallsNumber()
523
524 # Display number of iterations and number of evaluations
525 # of the limit state function
526 print "Number_of_evaluations_of_the_limit_state_function_=",
    deviationCallNumberAfter - deviationCallNumberBefore
527
528 # Display the Directional Simulation probability of 'myEvent'
529 print "Directional_Sampling_probability_estimation_=", myAlgoDirectionalSim.
    getResult().getProbabilityEstimate()
530
531 # Display the variance of the Directional Simulation probability estimator
532 print "Variance_of_the_Directional_Sampling_probability_estimator_=",
    myAlgoDirectionalSim.getResult().getVarianceEstimate()
533
534 # Display the confidence interval length centered around
535 # the Directional Simulation probability DSProb
536 # IC = [DSProb - 0.5*length, DSProb + 0.5*length]
537 # level 0.95
538 print "0.95_Confidence_Interval_=",
    getResult().getProbabilityEstimate() - 0.5*myAlgoDirectionalSim.getResult().

```

```

    getConfidenceLength(0.95), ",_," , myAlgoDirectionalSim.getResult().
    getProbabilityEstimate() + 0.5*myAlgoDirectionalSim.getResult().
    getConfidenceLength(0.95), "]"
539 print ""
540
541
542 # Draw the convergence graph and the confidence interval of level alpha
543 alpha = 0.90
544 convergenceGraphDS = myAlgoDirectionalSim.drawProbabilityConvergence(alpha)
545 # In order to see the graph without creating the associated files
546 #Show(convergenceGraphDS)
547
548 # Create the file .EPS
549 convergenceGraphDS.draw("convergenceGrapheDS", 640, 480)
550 #Show(convergenceGraphDS)
551
552
553 #####
554 # Latin HyperCube Sampling
555 #####
556
557 print "#####"
558 print "Latin_HyperCube_Sampling"
559 print "#####"
560 print ""
561
562
563 # We create a LHS algorithm
564 myAlgoLHS = LHS(myEvent)
565 myAlgoLHS.setMaximumOuterSampling(maximumOuterSampling)
566 myAlgoLHS.setBlockSize(blockSize)
567 myAlgoLHS.setMaximumCoefficientOfVariation(coefficientOfVariation)
568
569 # Define the HistoryStrategy to store the values of the probability estimator
570 # and the variance estimator
571 # used to draw the convergence graph
572 # Full strategy
573 myAlgoLHS.setConvergenceStrategy(HistoryStrategy(Full()))
574
575 # Perform the simulation
576 myAlgoLHS.run()
577
578 # Display number of iterations and number of evaluations
579 # of the limit state function
580 print "Number_of_evaluations_of_the_limit_state_function_=_", myAlgoLHS.
    getResult().getOuterSampling()*myAlgoLHS.getResult().getBlockSize()
581
582 # Display the LHS probability of {\itshape myEvent}

```

```

583 print "LHS_probability_estimation_=", myAlgoLHS.getResult().
      getProbabilityEstimate()
584
585 # Display the variance of the LHS probability estimator
586 print "Variance_of_the_LHS_probability_estimator_=", myAlgoLHS.getResult().
      getVarianceEstimate()
587
588 # Display the confidence interval length centered around the LHS probability
      LHSProb
589 # IC = [LHSProb - 0.5*length, LHSProb + 0.5*length]
590 # level 0.95
591 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), "]"
592 print ""
593
594 # Draw the convergence graph and the confidence interval of level alpha
595 alpha = 0.90
596 convergenceGraphLHS = myAlgoLHS.drawProbabilityConvergence(alpha)
597 # In order to see the graph without creating the associated files
598 #Show(convergenceGraphLHS)
599
600 # Create the file .EPS
601 convergenceGraphLHS.draw("convergenceGrapheLHS", 640, 480)
602 #Show(convergenceGraphLHS)
603
604 #####
605 # Importance Sampling
606 #####
607
608
609 print "#####"
610 print "Importance_Sampling"
611 print "#####"
612 print ""
613
614 maximumOuterSampling = 40000
615 blockSize = 1
616 standardSpaceDesignPoint = resultFORM.getStandardSpaceDesignPoint()
617 mean = standardSpaceDesignPoint
618 sigma = NumericalPoint(4, 1.0)
619 importanceDistribution = Normal(mean, sigma, CorrelationMatrix(4))
620
621 myStandardEvent = StandardEvent(myEvent)
622
623 myAlgoImportanceSampling = ImportanceSampling(myStandardEvent, Distribution(
      importanceDistribution))

```

```

624 myAlgoImportanceSampling.setMaximumOuterSampling(maximumOuterSampling)
625 myAlgoImportanceSampling.setBlockSize(blockSize)
626 myAlgoImportanceSampling.setMaximumCoefficientOfVariation(coefficientOfVariation
)
627
628 # Define the HistoryStrategy to store the values of the probability estimator
629 # and the variance estimator
630 # used ot draw the convergence graph
631 # Full strategy
632 myAlgoImportanceSampling.setConvergenceStrategy(HistoryStrategy(Full()))
633
634 # Perform the simulation
635 myAlgoImportanceSampling.run()
636
637 # Display number of iterations and number of evaluations
638 # of the limit state function
639 print "Number_of_evaluations_of_the_limit_state_function = ",
        myAlgoImportanceSampling.getResult().getOuterSampling()*
        myAlgoImportanceSampling.getResult().getBlockSize()
640
641 # Display the Importance Sampling probability of 'myEvent'
642 print "Importance_Sampling_probability_estimation = ", myAlgoImportanceSampling.
        getResult().getProbabilityEstimate()
643
644 # Display the variance of the Importance Sampling probability estimator
645 print "Variance_of_the_Importance_Sampling_probability_estimator = ",
        myAlgoImportanceSampling.getResult().getVarianceEstimate()
646
647 # Display the confidence interval length centered around
648 # the ImportanceSampling probability ISProb
649 # IC = [ISProb - 0.5*length, ISProb + 0.5*length]
650 # level 0.95
651 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), "]"
652
653 # Draw the convergence graph and the confidence intervale of level alpha
654 alpha = 0.90
655 convergenceGraphIS = myAlgoImportanceSampling.drawProbabilityConvergence(alpha)
656 # In order to see the graph whithout creating the associated files
657 #Show(convergenceGraphIS)
658
659 # Create the file .EPS
660 convergenceGraphIS.draw("convergenceGrapheIS", 640, 480)
661 #Show(convergenceGraphIS)
662

```

```

663
664
665
666 #####
667 # Response surface : Polynomial expansion chaos
668 #####
669
670 print "#####"
671 print "Polynomial_expansion_chaos"
672 print "#####"
673 print "_"
674
675 #####
676 # STEP 1 : Construction of the multivariate orthonormal basis
677
678 # Dimension of the input random vector
679 dim = 4
680
681 # Create the univariate polynomial family collection
682 # which regroups the polynomial families for each direction
683 polyColl = PolynomialFamilyCollection(dim)
684
685 # Variable E
686 #Jacobi(alpha, beta) <=> Beta(\beta + 1, \alpha + \beta + 2, -1, 1)
687 alphaJ = 1.27
688 betaJ = -0.07
689 jacobiFamily = JacobiFactory(alphaJ, betaJ)
690 polyColl[0] = OrthogonalUniVariatePolynomialFamily(jacobiFamily)
691
692
693 # Variable F
694 # Laguerre(k) <=> Gamma(k+1,1,0) (parametrage ppal)
695 kLaguerre = 1.78
696 laguerreFamily = LaguerreFactory(kLaguerre)
697 polyColl[1] = OrthogonalUniVariatePolynomialFamily(laguerreFamily)
698
699 # Variable L
700 # Legendre <=> Unif(-1,1)
701 legendreFamily = LegendreFactory()
702 polyColl[2] = OrthogonalUniVariatePolynomialFamily(legendreFamily)
703
704 # Variable E
705 # Jacobi(alpha, beta) <=> Beta(\beta + 1, \alpha + \beta + 2, -1, 1)
706 alphaJ2 = 0.5
707 betaJ2 = 1.5
708 jacobiFamily2 = JacobiFactory(alphaJ2, betaJ2)
709 polyColl[3] = OrthogonalUniVariatePolynomialFamily(jacobiFamily2)
710

```

```
711
712 # Create the multivariate orthonormal basis
713 # which is the the cartesian product of the univariate basis
714 multivariateBasis = OrthogonalProductPolynomialFactory(polyColl,
    LinearEnumerateFunction(dim))
715
716 # Build a term of the basis as a NumericalMathFunction
717 # Generally, we do not need to construct manually any term,
718 # all terms are constructed automatically by a strategy of construction of the
    basis
719 i=5
720 Psi_i = multivariateBasis.build(i)
721
722 # Get the measure mu associated to the multivariate basis
723 distributionMu = multivariateBasis.getMeasure()
724
725 #####
726 # STEP 2 : Truncature strategy of the multivariate orthonormal basis
727
728 # CleaningStrategy :
729 # among the maximumConsideredTerms = 500 first polynoms,
730 # those which have the mostSignificant = 50 most significant contributions
731 # with significance criterion significanceFactor = 10(-4)
732 # The True boolean indicates if we are interested
733 # in the online monitoring of the current basis update
734 # (removed or added coefficients)
735 maximumConsideredTerms = 500
736 mostSignificant = 50
737 significanceFactor = 1.0e-4
738 truncatureBasisStrategy = CleaningStrategy(OrthogonalBasis(multivariateBasis),
    maximumConsideredTerms, mostSignificant, significanceFactor, True)
739 truncatureBasisStrategy = FixedStrategy(OrthogonalBasis(multivariateBasis),481)
740
741 #####
742 # STEP 3 : Evaluation strategy of the approximation coefficients
743
744 # The technique proposed is the Least Squares technique
745 # where the points come from an design of experiments
746 # Here : the Monte Carlo sampling technique
747 sampleSize = 10000
748 evaluationCoeffStrategy = LeastSquaresStrategy(MonteCarloExperiment(sampleSize))
749
750 # STEP 4 : Creation of the Functional Chaos Algorithm
751
752 # FunctionalChaosAlgorithm :
753 # combination of the model : limitStateFunction
754 # the distribution of the input random vector : Xdistribution
755 # the truncature strategy of the multivariate basis
```

```

756 # and the evaluation strategy of the coefficients
757 polynomialChaosAlgorithm = FunctionalChaosAlgorithm(deviation , Distribution(
    inputDistribution), AdaptiveStrategy(truncatureBasisStrategy),
    ProjectionStrategy(evaluationCoeffStrategy))
758
759 #####
760 # Run and results exploitation
761
762 # Perform the simulation
763 polynomialChaosAlgorithm.run()
764
765 # Stream out the result
766 polynomialChaosResult = polynomialChaosAlgorithm.getResult()
767
768 # Get the polynomial chaos coefficients
769 coefficients = polynomialChaosResult.getCoefficients()
770
771 # Get the meta model as a NumericalMathFunction
772 metaModel = polynomialChaosResult.getMetaModel()
773
774 # Get the indices of the selected polynomials : K
775 subsetK = polynomialChaosResult.getIndices()
776
777 # Get the composition of the polynomials
778 # of the truncated multivariate basis
779 for i in range(subsetK.getSize()) :
780     print "Polynomial_number_", i, "_in_truncated_basis_<->_polynomial_number_",
        subsetK[i], "_=_", LinearEnumerateFunction(dim)(subsetK[i]), "_in_complete_
        basis"
781 print ""
782
783 # Get the multivariate basis
784 # as a collection of NumericalMathFunction
785 multivariateBasisCollection = polynomialChaosResult.getReducedBasis()
786
787 # Get the distribution of variables Z
788 mu = polynomialChaosResult.getDistribution()
789 print "Distribution_in_the_transformed_variables_=_", mu
790 print ""
791
792 # Get the composed model which is the model of the reduced variables Z
793 composedModel = polynomialChaosResult.getComposedModel()
794
795 # Define the new random vector
796 newOutputVariableOfInterest = RandomVector(polynomialChaosResult)
797
798 # Get the mean and variance of the meta model
799

```

```

800 print "Mean_=", newOutputVariableOfInterest.getMean()
801 print "Standard_deviation_=", sqrt(newOutputVariableOfInterest.getCovariance()
      [0,0])
802 print ""
803
804
805
806 #####
807 # Graphs validation
808
809
810 # Graph 1 : cloud
811
812 # Generate a NumericalSample of the input random vector
813 # Evaluate the meta model and the real model
814 # draw the coulds (metamodel, real model)
815 # Verify points are on the first diagonal
816 sizeX = 500
817 Xsample = inputDistribution.getNumericalSample(sizeX)
818
819 modelSample = deviation(Xsample)
820 metaModelSample = metaModel(Xsample)
821
822 sampleMixed = NumericalSample(sizeX,2)
823 for i in range(sizeX) :
824     sampleMixed[i][0] = modelSample[i][0]
825     sampleMixed[i][1] = metaModelSample[i][0]
826
827 legend = str(sizeX) + "_realizations"
828 comparisonCloud = Cloud(sampleMixed, "blue", "fsquare", legend)
829 graphCloud = Graph("Polynomial_chaos_expansion", "model", "meta_model", True, "
      topleft")
830 graphCloud.add(comparisonCloud)
831
832 #Show(graphCloud)
833 graphCloud.draw("PCE_comparisonModels")
834
835
836 # Graph 2 : polynoms family graphs
837
838 degreeMax = 5
839 pointNumber = 101
840 colorList = Drawable.GetValidColors()
841
842 # Jacobi for E
843 xMinJacobi = -1
844 xMaxJacobi = 1
845 titleJacobi = "Jacobi(" + str(alphaJ) + ",_" + str(betaJ) + ")_polynomials"

```

```

846 graphJacobi = Graph(titleJacobi , "z" , "polynomial_values" , True , "topleft")
847 for i in range(degreeMax) :
848     graphJacobi_temp = jacobiFamily.build(i).draw(xMinJacobi , xMaxJacobi ,
849         pointNumber)
849     graphJacobi_temp_draw = graphJacobi_temp.getDrawable(0)
850     legend = "degree_" + str(i)
851     graphJacobi_temp_draw.setLegendName(legend)
852     graphJacobi_temp_draw.setColor(colorList[i])
853     graphJacobi.add(graphJacobi_temp_draw)
854 #Show(graphJacobi)
855 graphJacobi.draw("PCE_JacobiPolynomials_VariableE")
856
857 # Laguerre for F
858 xMinLaguerre = 0
859 xMaxLaguerre = 10
860 titleLaguerre = "Laguerre(" + str(kLaguerre) + ")_polynomials"
861 graphLaguerre = Graph(titleLaguerre , "z" , "polynomial_values" , True , "topleft")
862 for i in range(degreeMax) :
863     graphLaguerre_temp = laguerreFamily.build(i).draw(xMinLaguerre , xMaxLaguerre ,
864         pointNumber)
864     graphLaguerre_temp_draw = graphLaguerre_temp.getDrawable(0)
865     legend = "degree_" + str(i)
866     graphLaguerre_temp_draw.setLegendName(legend)
867     graphLaguerre_temp_draw.setColor(colorList[i])
868     graphLaguerre.add(graphLaguerre_temp_draw)
869 #Show(graphLaguerre)
870 graphLaguerre.draw("PCE_LaguerrePolynomials_VariableF")
871
872 # Legendre for L
873 xMinLegendre = -1
874 xMaxLegendre = 1
875 titleLegendre = "Legendre_polynomials"
876 graphLegendre = Graph(titleLegendre , "z" , "polynomial_values" , True , "topright")
877 for i in range(degreeMax) :
878     graphLegendre_temp = laguerreFamily.build(i).draw(xMinLegendre , xMaxLegendre ,
879         pointNumber)
879     graphLegendre_temp_draw = graphLegendre_temp.getDrawable(0)
880     legend = "degree_" + str(i)
881     graphLegendre_temp_draw.setLegendName(legend)
882     graphLegendre_temp_draw.setColor(colorList[i])
883     graphLegendre.add(graphLegendre_temp_draw)
884 #Show(graphLegendre)
885 graphLegendre.draw("PCE_LegendrePolynomials_VariableL")
886
887 # Jacobi for I
888 xMinJacobi2 = -1
889 xMaxJacobi2 = 1
890 titleJacobi2 = "Jacobi(" + str(alphaJ2) + ",_" + str(betaJ2) + ")_polynomials"

```

```

891 graphJacobi2 = Graph(titleJacobi2 , "z" , "polynomial_values" , True , "topright")
892 for i in range(degreeMax) :
893     graphJacobi2_temp = jacobiFamily2.build(i).draw(xMinJacobi2 , xMaxJacobi2 ,
894         pointNumber)
895     graphJacobi2_temp_draw = graphJacobi2_temp.getDrawable(0)
896     legend = "degree_" + str(i)
897     graphJacobi2_temp_draw.setLegendName(legend)
898     graphJacobi2_temp_draw.setColor(colorList[i])
899     graphJacobi2.add(graphJacobi2_temp_draw)
900 #Show(graphJacobi2)
graphJacobi2.draw("PCE_JacobiPolynomials_VariableI")

```

1.8 Output of the Python script

```

1 #####
2 Min/Max study with deterministic design of experiments
3 #####
4 From a composite design of experiments 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 <-lcm
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 | \leftrightarrow | polynomial number 6 | = | [1,1,0,0] |
| | | in complete basis | | | | |
| 125 | Polynomial number 7 | in truncated basis | \leftrightarrow | polynomial number 7 | = | [1,0,1,0] |
| | | in complete basis | | | | |
| 126 | Polynomial number 8 | in truncated basis | \leftrightarrow | polynomial number 8 | = | [1,0,0,1] |
| | | in complete basis | | | | |
| 127 | Polynomial number 9 | in truncated basis | \leftrightarrow | polynomial number 9 | = | [0,2,0,0] |
| | | in complete basis | | | | |
| 128 | Polynomial number 10 | in truncated basis | \leftrightarrow | polynomial number 10 | = | [0,1,1,0] |
| | | in complete basis | | | | |
| 129 | Polynomial number 11 | in truncated basis | \leftrightarrow | polynomial number 11 | = | [0,1,0,1] |
| | | in complete basis | | | | |
| 130 | Polynomial number 12 | in truncated basis | \leftrightarrow | polynomial number 12 | = | [0,0,2,0] |
| | | in complete basis | | | | |
| 131 | Polynomial number 13 | in truncated basis | \leftrightarrow | polynomial number 13 | = | [0,0,1,1] |
| | | in complete basis | | | | |
| 132 | Polynomial number 14 | in truncated basis | \leftrightarrow | polynomial number 14 | = | [0,0,0,2] |
| | | in complete basis | | | | |
| 133 | Polynomial number 15 | in truncated basis | \leftrightarrow | polynomial number 15 | = | [3,0,0,0] |
| | | in complete basis | | | | |
| 134 | Polynomial number 16 | in truncated basis | \leftrightarrow | polynomial number 16 | = | [2,1,0,0] |
| | | in complete basis | | | | |
| 135 | Polynomial number 17 | in truncated basis | \leftrightarrow | 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 <=> polynomial number 105 =
    [0,5,0,0] in complete basis
149 Polynomial number 31 in truncated basis <=> polynomial number 111 =
    [0,2,3,0] in complete basis
150 Polynomial number 32 in truncated basis <=> polynomial number 120 =
    [0,0,5,0] in complete basis
151 Polynomial number 33 in truncated basis <=> polynomial number 122 =
    [0,0,3,2] in complete basis
152 Polynomial number 34 in truncated basis <=> polynomial number 131 =
    [4,1,1,0] in complete basis
153 Polynomial number 35 in truncated basis <=> polynomial number 147 =
    [2,3,1,0] in complete basis
154 Polynomial number 36 in truncated basis <=> polynomial number 150 =
    [2,2,1,1] in complete basis
155 Polynomial number 37 in truncated basis <=> polynomial number 163 =
    [1,4,0,1] in complete basis
156 Polynomial number 38 in truncated basis <=> polynomial number 238 =
    [3,1,1,2] in complete basis
157 Polynomial number 39 in truncated basis <=> polynomial number 248 =
    [2,3,2,0] in complete basis
158 Polynomial number 40 in truncated basis <=> polynomial number 249 =
    [2,3,1,1] in complete basis
159 Polynomial number 41 in truncated basis <=> 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.

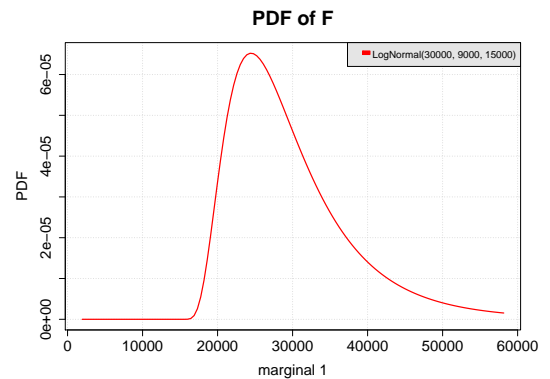
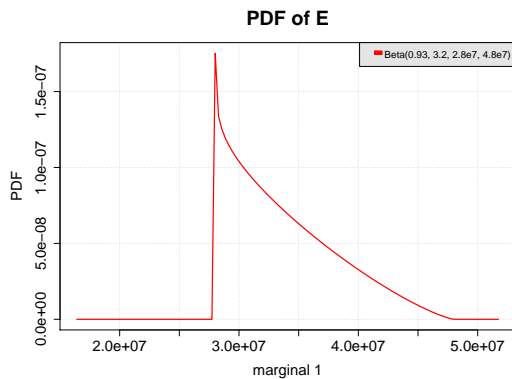


Figure 2: Probability density function of the parameter E Figure 3: Probability density function of the parameter F

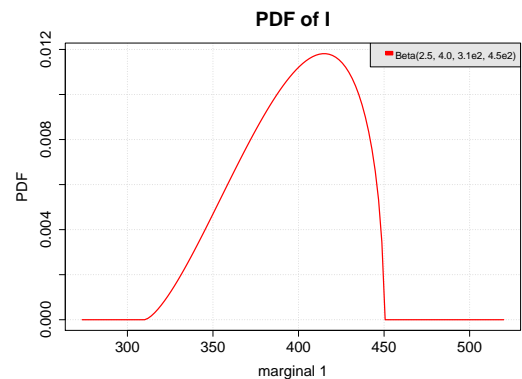
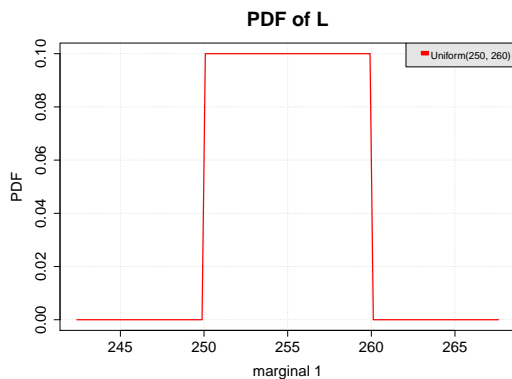


Figure 4: PDF of the parameter L

Figure 5: PDF of the parameter I

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.

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.

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

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 first five 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.

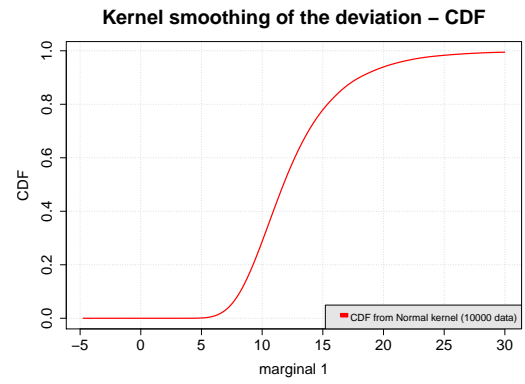
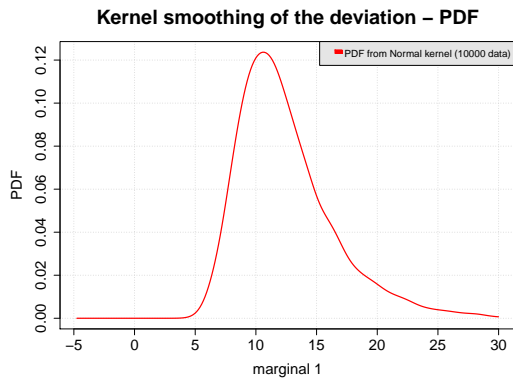


Figure 6: PDF of the deviation with the kernel smoothing method. Figure 7: CDF of the deviation with the kernel smoothing method.

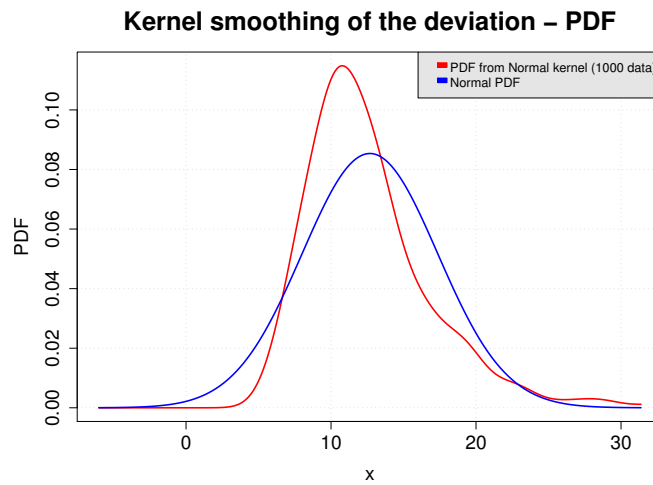


Figure 8: Superposition of the kernel smoothed density function and the normal fitted from the same sample.

FORM Importance factors – Deviation > 30 cm

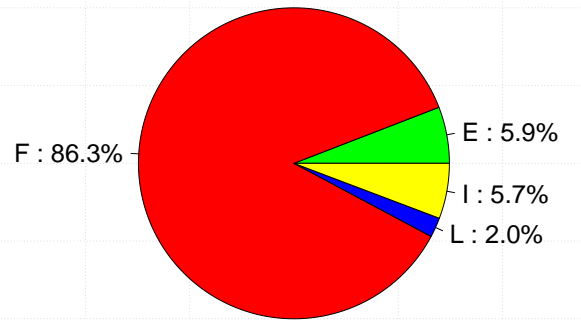


Figure 9: FORM importance factors of the event : deviation \geq 30 cm.

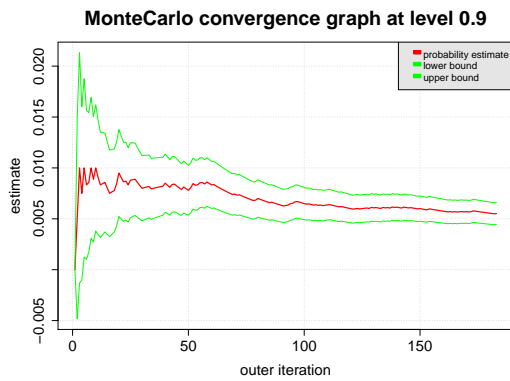


Figure 10: Monte Carlo convergence graph.

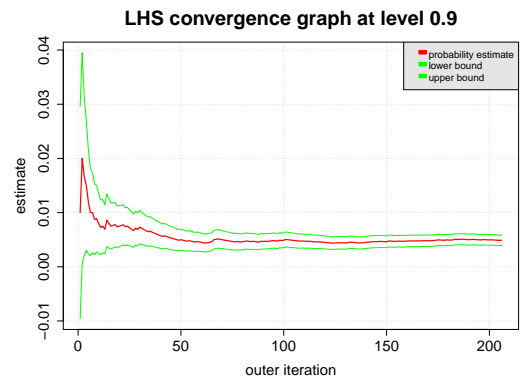


Figure 11: LHS convergence graph.

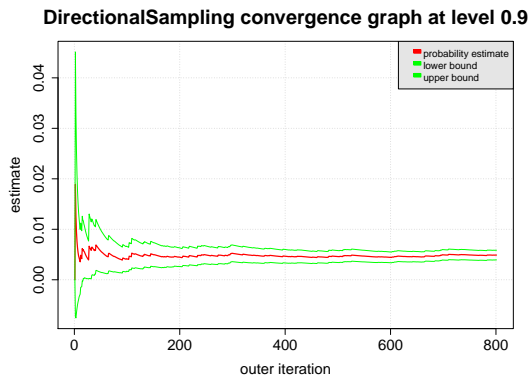


Figure 12: Directional Sampling convergence graph.

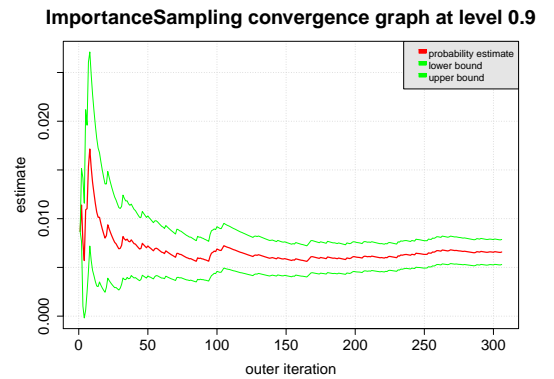


Figure 13: LHS convergence graph.

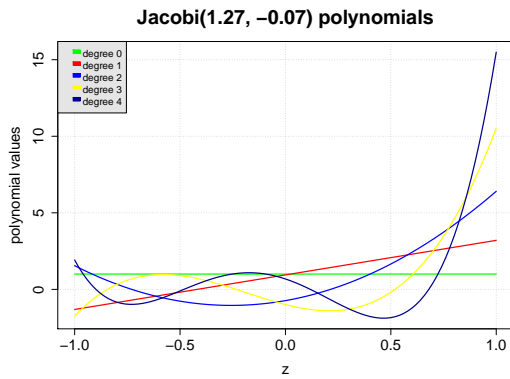


Figure 14: The 5-th first polynomials of the Jacobi family associated to the variable E .

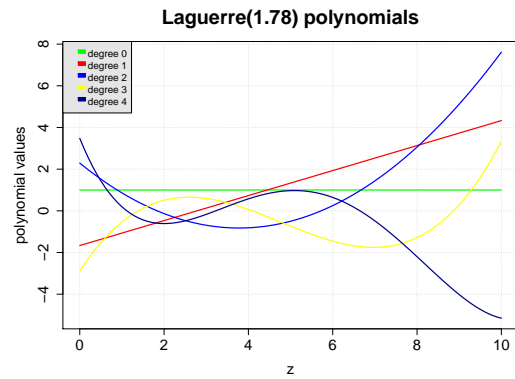


Figure 15: The 5-th first polynomials of the Laguerre family associated to the variable F .

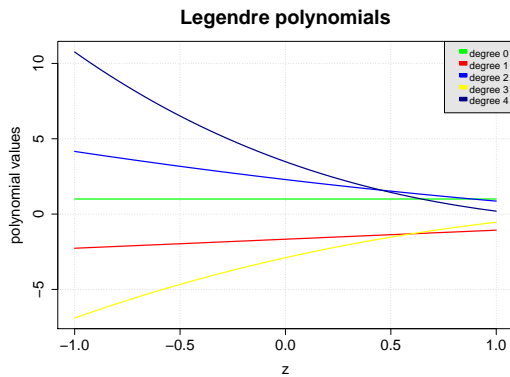


Figure 16: The 5-th first polynomials of the Legendre associated to the variable L .

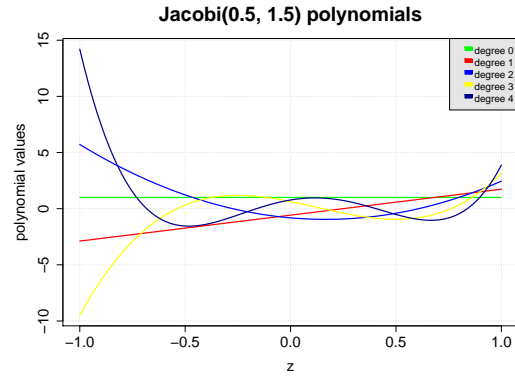


Figure 17: The 5-th first polynomials of the Jacobi family associated to the variable I .

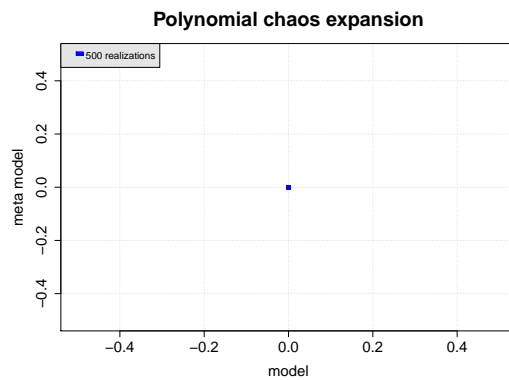


Figure 18: Comparison of values from the model and the polynomial chaos meta model.

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 design of experiments 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 design of experiments , whereas the random sampling technique did not manage to give a good evaluation of them.

Note that the composite design of experiments 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 realization 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.

2 Example 2: elastic truss structure

2.1 Problem statement

2.1.1 Physical model

Let us consider the elastic truss structure represented in Figure 19. The structure comprises three bars (labelled 1, 2 and 3) with specific cross-section areas and Young's moduli (denoted by S_1, S_2, S_3 and E_1, E_2, E_3 , respectively). The angle between bar #1 and bar #3 (resp. bar #2 and bar #3) is denoted by α_1 (resp. α_2). The system is subjected to an oblique point load P such that the angle between P and bar #3 is equal to θ .

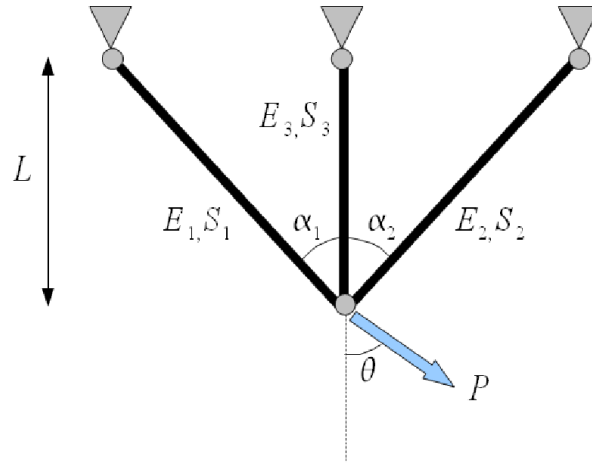


Figure 19: Elastic three-bar truss structure

The model response of interest is the norm δ of the displacement vector at the bottom node. It may be derived analytically as follows. The traction forces in the vertical bar (bar #1) due to the vertical load $P_v = P \cos \theta$ and the horizontal load $P_h = P \sin \theta$ are respectively equal to:

$$N_1^v = \frac{\sin(\alpha_2 + \alpha_1)}{\sin(\alpha_2)} \left[\frac{E_3 S_3}{\cos(\alpha_1) E_1 S_1} + \frac{E_3 S_3}{\cos(\alpha_2) E_2 S_2} \left(\frac{\sin(\alpha_1)}{\sin(\alpha_2)} \right)^2 + \left(\frac{\sin(\alpha_1 + \alpha_2)}{\sin(\alpha_2)} \right)^2 \right]^{-1} P_v = C_v P_v \quad (1)$$

$$N_1^h = \frac{\frac{\sin(\alpha_1)}{\sin(\alpha_2)^2 \cos(\alpha_2) E_2 S_2} + \frac{\sin(\alpha_1 + \alpha_2) \cos(\alpha_2)}{\sin(\alpha_2) E_3 S_3}}{\frac{1}{\cos(\alpha_1) E_1 S_1} + \frac{1}{\cos(\alpha_2) E_2 S_2} \left(\frac{\sin(\alpha_1)}{\sin(\alpha_2)} \right)^2 + \left(\frac{\sin(\alpha_1 + \alpha_2)}{\sin(\alpha_2)} \right)^2 \frac{1}{E_3 S_3}} P_h = C_h P_h \quad (2)$$

Upon applying the Castigliano's theorem, the vertical and the horizontal displacements are respectively given by:

$$\delta_v = \left[\frac{\alpha_v^2}{E_1 S_1 \cos(\alpha_1)} + \frac{\alpha_v^2 \sin^2(\alpha_1)}{E_2 S_2 \cos(\alpha_2) \sin^2(\alpha_2)} + \frac{\left(1 - \frac{\sin(\alpha_1 + \alpha_2)}{\sin(\alpha_2)} \right)^2}{E_3 S_3} \right] P_v L \quad (3)$$

$$\delta_h = \left[\frac{\alpha_v^2}{E_1 S_1 \cos(\alpha_1)} + \frac{\left(\alpha_h \frac{\sin(\alpha_1)}{\sin(\alpha_2)} \right)^2}{E_2 S_2 \cos(\alpha_2)} + \frac{\left(\alpha_h \frac{\sin(\alpha_1 + \alpha_2)}{\sin(\alpha_2)} - \frac{\cos(\alpha_2)}{\sin(\alpha_2)} \right)^2}{E_3 S_3} \right] P_h L \quad (4)$$

where L denotes the length of bar #3. Eventually the model response of interest δ is the Euclidean norm of the displacement, that is:

$$\delta = \sqrt{\delta_h^2 + \delta_v^2} \quad (5)$$

2.1.2 Probabilistic model

The cross-section areas, the Young's moduli, the point load and the angles are assumed to be uncertain and are modelled by independent random variables. Hence an input random vector of dimension $M = 10$ and which reads:

$$\underline{X} = \{E_1, E_2, E_3, S_1, S_2, S_3, P, \alpha_1, \alpha_2, \theta\}^T \quad (6)$$

The distributions and the parameters of the random variables are reported in Table 1.

| Variable | Distribution | Mean | Coef. of variation |
|------------|--------------|---------------------------------|--------------------|
| E_i | Lognormal | 210 GPa | 10% |
| S_i | Normal | $1.5 \cdot 10^{-3} \text{ m}^2$ | 5% |
| P | Gumbel | $2.5 \cdot 10^5 \text{ N}$ | 20% |
| α_j | Normal | 45° | 3% |
| θ | Normal | 45° | 3% |

Table 1: Three-bar truss example – Input random variables

Due to uncertainty propagation through the model \mathcal{M} , the displacement (i.e. the model response) is also a random variable denoted by $Y = \mathcal{M}(\underline{X})$. The characterization of the distribution of Y and of some of its properties (e.g. moments, sensitivity indices) are of interest in the sequel.

2.2 Uncertainty and sensitivity analysis based on polynomial chaos expansions

2.2.1 Methodology to construct a sparse polynomial chaos approximation

In order to perform uncertainty and sensitivity analysis at a low computational cost, we aim at constructing a polynomial chaos (PC) approximation of the model response. In this purpose, the input parameters X_i are first scaled according to the following isoprobabilistic transform:

$$\xi_i = \Phi^{-1}(F_{X_i}(X_i)) \quad , \quad i = 1, \dots, 10 \quad (7)$$

where Φ and F_{X_i} denote the cumulative density function of the standard normal distribution and the variable X_i , respectively. This makes it possible to recast the random model response (denoted by Y) in terms of independent standard normal random variables $\underline{\xi} = \{\xi_1, \dots, \xi_{10}\}$ as follows:

$$Y = \mathcal{M}(\underline{\xi}) \quad (8)$$

We want to approximate the model response Y by a PC expansion made of normalized Hermite polynomials. Such a representation reads:

$$Y \simeq Y^{\text{PC}} = \sum_{\alpha \in \Lambda} a_{\alpha} \psi_{\alpha}(\underline{\xi}) \quad (9)$$

In the above equation, $\psi_{\alpha}(\underline{\xi})$ is a product of normalized Hermite polynomials, that is:

$$\psi_{\alpha}(\underline{\xi}) = \prod_{i=1}^M H_{\alpha_i}(\xi_i) \quad (10)$$

where H_{α_i} is the normalized Hermite polynomial of degree α_i . Λ is a non empty and finite subset of \mathbb{N}^M . The a_{α} 's are the PC coefficients that have to be estimated.

It is assumed that the maximum number of allowed simulations (i.e. the simulation budget) is equal to $N = 200$. We want to make the best use of these model evaluations to construct a PC approximation of the response. To this end, several PC expansions are built up for various total degrees p ranging from 1 to 4 (thus the so-called *fixed strategy* is used to truncate all of these metamodels). It is recalled that the number of terms in these approximations is given by the formula:

$$P = \frac{(M + p)!}{M!p!} \quad (11)$$

It has to be noted that $P > N$ when the degree p is greater than or equal to 3 (P is equal to 286 if $p = 3$ and to 1,001 if $p = 4$). As a consequence, it is not possible to evaluate the PC coefficients by *ordinary least squares* in this case since the problem would be ill-posed. As an alternative, the coefficients are computed using a *sparse least squares* approach based on *least angle regression* (LAR). In this purpose, the N simulations are carried out in such a way that the corresponding N realizations of the input random vector \underline{X} form a *latin hypercube design*.

2.2.2 Parametric study varying the degree of the metamodel

The LAR approach is used to compute the coefficients of the PC approximations of degrees p varying from 1 to 4. Then these metamodels are assessed by evaluating their *corrected leave-one-out errors* (these quantity are calculated when running LAR and are hence available as a by-product of this algorithm). The errors are plotted in Figure 20.

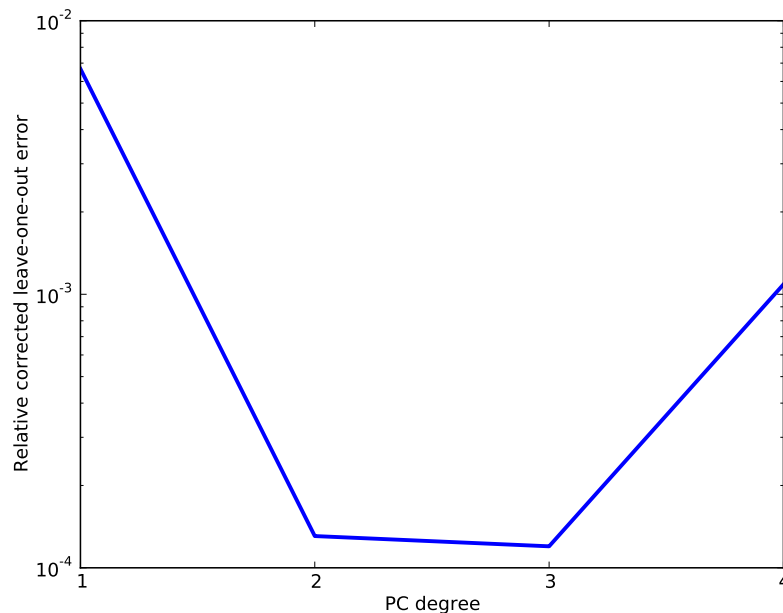


Figure 20: Corrected leave-one-out errors associated with the various polynomial chaos approximations

It appears that the PC of degree $p = 3$ is the most accurate, with an error estimate equal to $1.2 \cdot 10^{-4}$. It contains $P' = 97$ non zero coefficients, that is a “sparsity ratio” equal to $97/286 \approx 34\%$ compared to the total number of coefficients.

2.2.3 Second moments and sensitivity indices based on chaos coefficients

From now on, we only consider the sparse PC expansion of degree $p = 3$. It is straight forward to estimate the mean and the standard deviation of the random displacement Y from the PC coefficients:

$$\mu_{Y^{PC}} \approx 4.3 \text{ mm} \quad , \quad \sigma_{Y^{PC}} \approx 0.9 \text{ mm} \quad (12)$$

hence a coefficient of variation equal to 21%.

The sensitivity indices of Y to each input random variable X_i can also be computed easily from the coefficients. The *single-effect* indices are plotted together with the *total* ones in Figure 21.

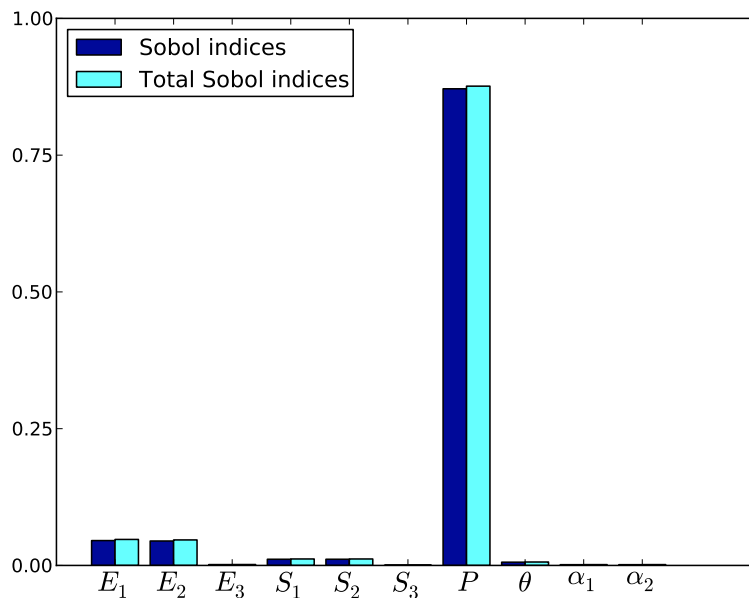


Figure 21: Single-effect and total sensitivity indices of the truss displacement to the various input parameters

It is observed that both kinds of indices strongly coincide, which reveals that there is almost no interaction effect. The variability of Y is mainly explained by the one of the point load P . Indeed, the associated sensitivity indices are both nearly equal to 90%. On the contrary, all the input variables except the Young’s moduli E_1 and E_2 have a negligible influence on the variance of Y , with sensitivity indices less than 2%. Therefore, it would be relevant in further investigation to fix all the insignificant variables to their nominal values, and to focus on a fine characterization of the distribution of the load P .

2.2.4 Distribution and higher-order moments analysis

Of interest is the estimation of the probability density function (PDF) of Y by exploiting its PC approximation Y^{PC} , which is very fast to evaluate. Note that in this example the original model is itself fast to simulate,

for it is formulated in terms of a closed-form equation. Hence the computational gain related to the use of a metamodel is not really significant. Nonetheless, such a gain would be considerable in presence of a more complicated model, e.g. a finite element model with many degrees of freedom. This is why we nevertheless decided to use the metamodel-based methodology in the following.

First, a large sample of size $\mathcal{N} = 100,000$ is drawn according to the joint distribution of the input random vector \underline{X} . Then the PC metamodel is evaluated at each input realization, leading to a \mathcal{N} -sample of responses. The histogram of this sample is represented in Figure 22. From visual inspection, the PDF of Y appears to be slightly positively skewed.

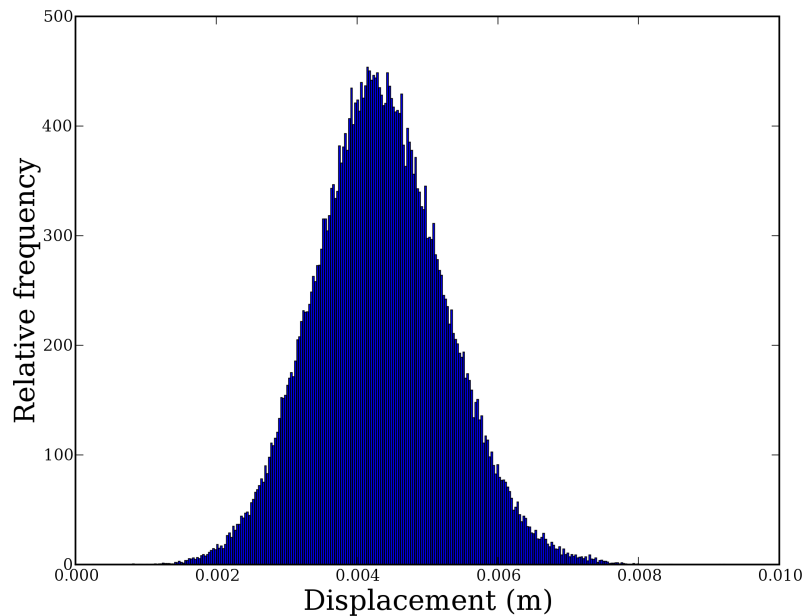


Figure 22: Histogram of the random displacement

The sample standardized moments of order 3 and 4, namely the *skewness* and *kurtosis* coefficients, are given by:

$$\gamma_{1,Y^{PC}} \approx 0.15 \quad , \quad \gamma_{2,Y^{PC}} \approx 3.09 \quad (13)$$

Broadly speaking, the random variable Y is not so far from a Gaussian distribution, for which the skewness and the kurtosis would be equal to 0 and 3, respectively. This could be expected since even a PC of degree $p = 1$ had an approximation error of only $7 \cdot 10^{-3}$ (Figure 20). In other words, a linear combination of Gaussian variables ξ_i , which is itself a Gaussian variable, revealed a fair approximation of the model response.

2.3 Python scripts

The Python files include two modules corresponding to the physical and the probabilistic models, and one main program which performs the uncertainty analysis based on polynomial chaos expansions.

2.3.1 Physical model – `Physical_model.py`

```

1 from openturns import *
2 import numpy as np
3 #
4 #-----
5 #
6 # Number of input random variables
7 dim=10
8 #
9 # Definition of the physical model
10 class myfunction(OpenTURNPythonFunction):
11     def __init__(self):
12         OpenTURNPythonFunction.__init__(self, dim, 1)
13     def _exec(self, X):
14
15         # Length of bar 3 (deterministic)
16         L = 5.    # in m
17
18         # Extract input variables from X
19         E1,E2,E3,S1,S2,S3,P,Theta,alpha1,alpha2 = X
20
21         # Degrees to radians
22         theta_rad=Theta*np.pi/180.0
23         alpha1_rad=alpha1*np.pi/180.0
24         alpha2_rad=alpha2*np.pi/180.0
25
26         # Vertical and horizontal loads
27         Pv=-1.*P*np.cos(theta_rad)
28         Ph=P*np.sin(theta_rad)
29
30         # Intermediate variables
31         alphav=(np.sin(alpha2_rad+alpha1_rad)/np.sin(alpha2_rad))*(1.0/((E3*S3
32             /(np.cos(alpha1_rad)*E1*S1)+(E3*S3/(np.cos(alpha2_rad)*E2*S2))*(np
33             .sin(alpha1_rad)/np.sin(alpha2_rad))**2+(np.sin(alpha2_rad+
34             alpha1_rad)/np.sin(alpha2_rad))**2))
35
36         alphah=(np.sin(alpha1_rad)/(np.sin(alpha2_rad)**2*np.cos(alpha2_rad)*
37             E2*S2)+(np.sin(alpha1_rad+alpha2_rad)*np.cos(alpha2_rad))/(np.sin(
38             alpha2_rad)**2*E3*S3))/(1.0/(np.cos(alpha1_rad)*E1*S1)+((np.sin(
39             alpha1_rad)/np.sin(alpha2_rad))**2)*(1.0/(np.cos(alpha2_rad)*E2*S2)
40             )+((np.sin(alpha1_rad+alpha2_rad)/np.sin(alpha2_rad))**2)*(1.0/(E3*
41             S3)))
42
43         # Horizontal and vertical displacements
44         deltah=(alphah**2/(E1*S1*np.cos(alpha1_rad))+((alphah*np.sin(
45             alpha1_rad)/np.sin(alpha2_rad)-1.0/np.sin(alpha2_rad))**2)/(E2*S2*
46             np.cos(alpha2_rad))+((alphah*np.sin(alpha1_rad+alpha2_rad)/np.sin(
47             alpha2_rad)-np.cos(alpha2_rad)/np.sin(alpha2_rad))**2)/(E3*S3))*Ph*

```

```

37         L
           deltav=(alphav**2/(E1*S1*np.cos(alpha1_rad)))+(alphav**2*np.sin(
           alpha1_rad)**2)/(E2*S2*np.cos(alpha2_rad)*np.sin(alpha2_rad)**2)
           +((1.0-alphav*np.sin(alpha2_rad+alpha1_rad)/np.sin(alpha2_rad))**2)
           /(E3*S3))*Pv*L
38
39     # L2-norm of the displacement vector
40     Y = np.sqrt(deltah**2+deltav**2)
41
42     return [Y]
43 #
44 #=====
45 #
46 # For test: evaluate the model at the nominal input parameters
47 #
48 if __name__ == "__main__":
49     #
50     truss_model = NumericalMathFunction(myfunction())
51     #
52     Xnom = NumericalPoint(([2100.e6,2100.e6,2100.e6
53         ,0.0015,0.0015,0.0015,2500.,45.,45.,45.]))
54     Ynom = truss_model(Xnom)
55     #
56     print "" ; print "Value_of_the_displacement_(m):", Ynom[0] ; print ""

```

2.3.2 Probabilistic model – Proba_model.py

```

1 from openturns import *
2 #
3 # Number of input random variables
4 dim=10
5 #
6 #=====
7 #                               Marginal PDFs
8 #=====
9 #
10 # Young's moduli
11 distE1 = LogNormal(210.e9,0.10,0.,LogNormal.MU.SIGMAOVERMU) # Mean in Pa
12 distE2 = LogNormal(210.e9,0.10,0.,LogNormal.MU.SIGMAOVERMU) # Mean in Pa
13 distE3 = LogNormal(210.e9,0.10,0.,LogNormal.MU.SIGMAOVERMU) # Mean in Pa
14 # Cross-section areas
15 distS1 = Normal(0.0015,0.0015*0.05) # Mean in m**2
16 distS2 = Normal(0.0015,0.0015*0.05) # Mean in m**2
17 distS3 = Normal(0.0015,0.0015*0.05) # Mean in m**2
18 # Point load
19 distP = Normal(250000.,250000.*0.20) # Mean in N
20 #Load direction

```

```

21 distTheta = Normal(45.,45.*0.03)      # Mean in degrees
22 # Angle (bar1-bar3)
23 distalpha1 = Normal(45.,45.*0.03)    # Mean in degrees
24 # Angle (bar2-bar3)
25 distalpha2 = Normal(45.,45.*0.03)    # Mean in degrees
26 #
27 #=====
28 #                               Input random vector
29 #=====
30 #
31 myCollection = DistributionCollection(dim)
32 myCollection[0] = Distribution(distE1)
33 myCollection[1] = Distribution(distE2)
34 myCollection[2] = Distribution(distE3)
35 myCollection[3] = Distribution(distS1)
36 myCollection[4] = Distribution(distS2)
37 myCollection[5] = Distribution(distS3)
38 myCollection[6] = Distribution(distP)
39 myCollection[7] = Distribution(distTheta)
40 myCollection[8] = Distribution(distalpha1)
41 myCollection[9] = Distribution(distalpha2)
42 myDistribution = ComposedDistribution(myCollection)
43 vectX = RandomVector(Distribution(myDistribution))

```

2.3.3 Uncertainty analysis based on PC expansions – main_SparsePolyChaos.py

```

1 from openturns import *
2 from Physical_Model import *
3 from Proba_Model import *
4 from numpy import empty, argmin, array, arange, floor, sqrt, linspace
5 from pylab import ion, figure, semilogy, xlabel, ylabel, bar, legend, xticks,
  yticks, hist
6 #
7 # Model function
8 #
9 truss_model = NumericalMathFunction(myfunction())
10 #
11 # Output random vector
12 #
13 vectY = RandomVector(truss_model, vectX)
14 #
15 # Basis of the polynomial chaos (PC) expansion (Hermite polynomials are selected
  )
16 #
17 polyColl = PolynomialFamilyCollection(dim)
18 for i in range(dim):
19     polyColl[i] = OrthogonalUniVariatePolynomialFamily(HermiteFactory())

```

```

20 #
21 enumerateFunction = EnumerateFunction(dim)
22 multivariateBasis = OrthogonalProductPolynomialFactory(polyColl ,
    enumerateFunction)
23 #
24 # Definition of the approx. algo.: Least Angle Regression (LAR)
25 #
26 basisSequenceFactory = LAR()
27 fittingAlgorithm = CorrectedLeaveOneOut()
28 approximationAlgorithm = LeastSquaresMetaModelSelectionFactory(
    basisSequenceFactory , fittingAlgorithm)
29 #
30 # Number of simulations (i.e. simulation budget)
31 #
32 N = 200
33 #
34 # Initialization of the seed of the random generator
35 RandomGenerator.SetSeed(77)
36 #
37 evalStrategy = LeastSquaresStrategy(LHSExperiment(N) , approximationAlgorithm)
38 #
39 # Parametric study varying the PC degree
40 #
41 pmax = 4
42 Results_tuple = ()
43 Relative_errors = empty(pmax)
44 p_list = range(1 , pmax+1)
45 #
46 for p in p_list:
47     #
48     P = enumerateFunction.getStrataCumulatedCardinal(p)
49     truncatureBasisStrategy = FixedStrategy(OrthogonalBasis(multivariateBasis) , P)
50     #
51     polynomialChaosAlgorithm = FunctionalChaosAlgorithm (truss_model , \
52         Distribution(myDistribution) , AdaptiveStrategy(truncatureBasisStrategy) , \
53         ProjectionStrategy(evalStrategy))
54     polynomialChaosAlgorithm.run()
55     #
56     new_result = polynomialChaosAlgorithm.getResult()
57     Results_tuple = Results_tuple + (new_result ,)
58     #
59     Relative_errors[p-1] = array(new_result.getRelativeErrors())[0]
60 #
61 # Plot the obtained relative errors
62 ion()
63 lw = 2.5
64 figure(1)
65 semilogy(p_list , Relative_errors , linewidth=lw)

```

```

66 xticks(list(arange(pmax)+1))
67 xlabel('PC_degree') ; ylabel('Relative_corrected_leave-one-out_error')
68 #
69 # Identify the most accurate PC expansion
70 p_optim = argmin(Relative_errors) + 1
71 The_Result = Results_tuple[p_optim-1]
72 #
73 print "" ; print "Optimal_degree:", p_optim
74 print "" ; print "Relative_corrected_LOO_error:", Relative_errors[p_optim-1]
75 print "" ; print "Number_of_nonzero_coefficients:", len(The_Result.getIndices())
76 #
77 # Post-processing of the optimal PC expansion
78 #
79 ChaosRV = FunctionalChaosRandomVector(The_Result)
80 #
81 Mean = ChaosRV.getMean()[0]
82 StD = sqrt(ChaosRV.getCovariance()[0,0])
83 print "" ; print "Response_mean:", Mean ; print ""
84 print "" ; print "Response_standard_deviation:", StD ; print ""
85 #
86 SU = empty(dim) ; SUT = empty(dim)
87 for i in range(dim):
88     SU[i] = ChaosRV.getSobolIndex(i)
89     SUT[i] = ChaosRV.getSobolTotalIndex(i)
90 #
91 # Plot the sensitivity indices
92 w = 0.4
93 figure(2)
94 b1 = bar((arange(dim)+1)-w,SU, width=w, color='#000999')
95 b2 = bar((arange(dim)+1),SUT, width=w, color='#66FFFF')
96 legend((b1[0],b2[0]),('Sobol_indices','Total_Sobol_indices'),'upper_left')
97 xticks(list(arange(dim)+1),(r'$E_1$',r'$E_2$',r'$E_3$',r'$S_1$',r'$S_2$',r'$S_3$
    ',r'$P$',r'$\theta$',r'$\alpha_1$',r'$\alpha_2$'),size=18)
98 yticks(list(linspace(0.,1.,5)))
99 #
100 #Plot histogram
101 truss_model_PC = The_Result.getMetaModel()
102 #
103 samplesize = 100000
104 sample_X = vectX.getNumericalSample(samplesize)
105 sample_YPC = truss_model_PC(sample_X)
106 asampleYPC = array(sample_YPC).flatten()
107 #
108 figure(3)
109 hist(sample_YPC,normed=True,bins=floor(sqrt(samplesize)))
110 #
111 print "" ; print "Skewness:", sample_YPC.computeSkewnessPerComponent()[0] ;
    print ""

```

```
112 print "" ; print "Kurtosis:", sample_YPC.computeKurtosisPerComponent()[0] ;  
    print ""
```