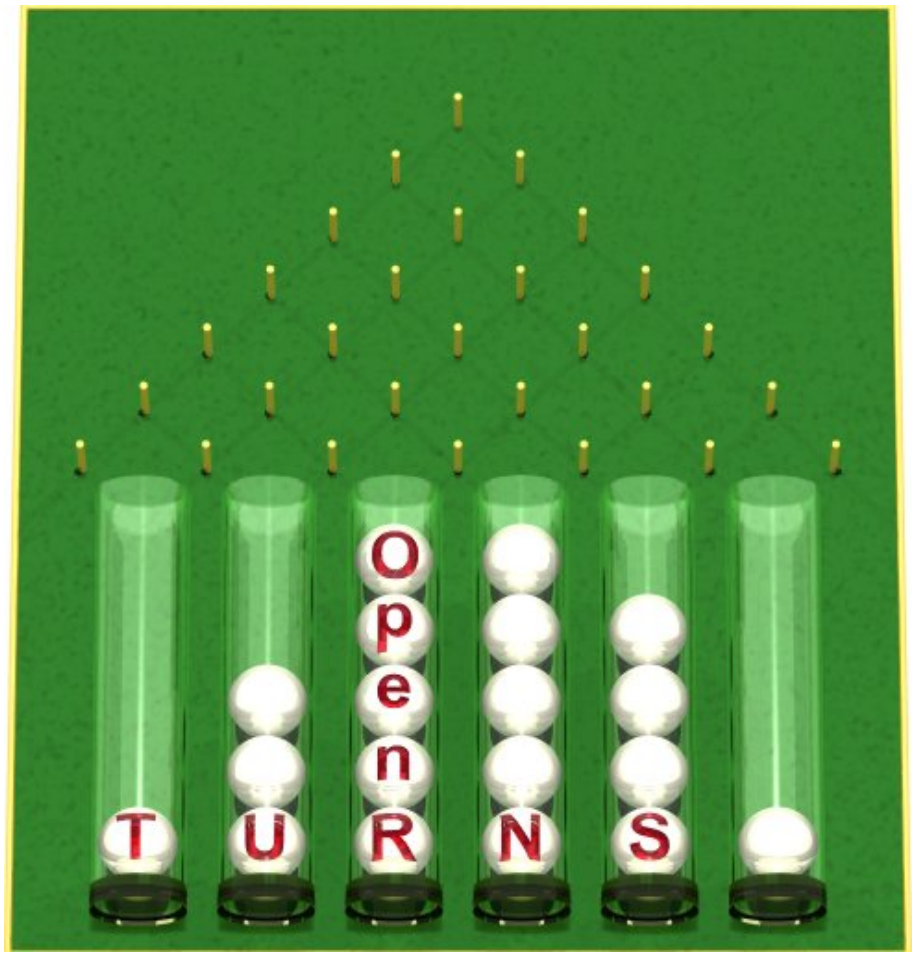
Examples Guide

OpenTURNS 1.7

Documentation built from package openturns-doc-16.03

March 17, 2016



Contents

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 : *OpenTURNS, 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 :

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

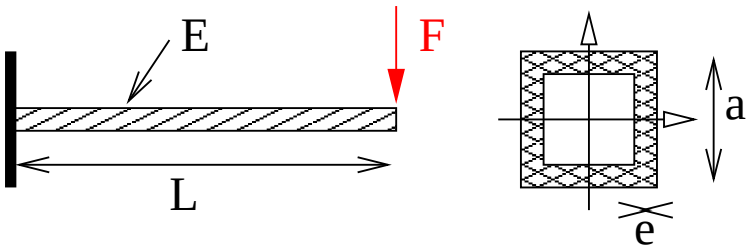


Figure 1: cantilever beam under a ponctual bending load.

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 tendancy 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 = Beta(∗) where $r = O.93, t = 3.2, a = 2.8e7, b = 4.8e7$,
- F = LogNormal, where the mean value is $E[F] = 30000$, the standard deviation is $\sqrt{\mathrm{Var}\,[F]} = 9000$ and the min value is $min(E) = 15000$,
- L = Uniform on $[250; 260]$,
- I = Beta(∗) 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 probabilstic variables L and I are dependent. This dependence may be explained by the manufacturing process of the beam : the thiner 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 reducted grid are +/-0.5, +/-1., +/-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 tendancy 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 Cobyla algorithm to research the design point, which requires no evaluation of the gradient of the limit state function. We parameterize the Cobyla algorithmwit hte 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 = $4\,10^4$,
- 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 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 $4\,10^4$.

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

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 (13) to (17) 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

```
from openturns import *

from math import sqrt

#########################
### Function 'deviation'
#########################
# Create here the python lines to define the implementation of the function

# In order to be able to use that function with the openturns library,
# it is necessary to define a class which derives from OpenTURNSPythonFunction

class modelePYTHON(OpenTURNSPythonFunction) :
  # that following method defines the input size (4) and the output size (1)
  def __init__(self) :
    OpenTURNSPythonFunction.__init__(self,4,1)

  # that following method gives the implementation of modelePYTHON
  def _exec(self,x) :
    E=x[0]
    F=x[1]
    L=x[2]
    I=x[3]
    return [(F*L*L*L)/(3.*E*I)]

# Use that function defined in the script python
# with the openturns library
# Create a NumericalMathFunction from modelePYTHON
deviation = NumericalMathFunction(modelePYTHON())


##############################################
### Input random vector
##############################################

# Create the marginal distriutions of the input random vector
distributionE = Beta(0.93, 3.2, 2.8e7, 4.8e7)
distributionF = LogNormal(30000, 9000, 15000, LogNormal.MUSIGMA)
distributionL = Uniform(250, 260)
distributionI = Beta(2.5, 4.0, 3.1e2, 4.5e2)

```

```
42 # Visualize the probability density functions
43
44 pdfLoiE = distributionE.drawPDF()
45 # Change the legend
46 draw_E = pdfLoiE.getDrawable(0)
47 draw_E.setLegend("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 #View(pdfLoiE).show()
54
55 pdfLoiF = distributionF.drawPDF()
56 # Change the legend
57 draw_F = pdfLoiF.getDrawable(0)
58 draw_F.setLegend("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 #View(pdfLoiF).show()
65
66 pdfLoiL = distributionL.drawPDF()
67 # Change the legend
68 draw_L = pdfLoiL.getDrawable(0)
69 draw_L.setLegend("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 #View(pdfLoiL).show()
76
77
78 pdfLoiI = distributionI.drawPDF()
79 # Change the legend
80 draw_I = pdfLoiI.getDrawable(0)
81 draw_I.setLegend("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 #View(pdfLoiI).show()
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 = [distributionE, distributionF, distributionL,
        distributionI]
101
102 # Create the input probability distribution of dimension 4
103 inputDistribution = ComposedDistribution(collectionMarginals, copuleNormal)
104
105 # Give a description of each component of the input distribution
106 inputDistribution.setDescription( ("E","F","L","I") )
107
108 # Create the input random vector
109 inputRandomVector = RandomVector(inputDistribution)
110 inputRandomVector.setDescription( ("E","F","L","I") )
111
112 # Create the output variable of interest
113 outputVariableOfInterest =  RandomVector(deviation, inputRandomVector)
114
115
116 ############################
117 ### Min/Max approach Study
118 ############################
119
120
121 ########################################################
122 # Min/Max study with deterministic design of experiment
123 ########################################################
124
125 print "########################################################"
126 print " Min/Max study with deterministic design of experiments "
127 print "########################################################"
128
129
130 dim = deviation.getInputDimension()
131
132 # Create the structure of the design of experiments : Composite type
133
134 # On each direction separately, several levels are evaluated
135 # here,  3 levels : +/-0.5, +/-1., +/-3. from the center
136 levelsNumber = 3
```

```
137 levels = NumericalPoint(levelsNumber, 0.0)
138 levels[0] = 0.5
139 levels[1] = 1.0
140 levels[2] = 3.0
141 # Creation of the composite design of experiments
142 myDesign = Composite(dim, levels)
143
144 # Generation of points according to the structure of the design of experiments
145 # (in a reduced centered space)
146 inputSample = myDesign.generate()
147
148 # Scaling of the structure of the design of experiments
149 # scaling vector for each dimension of the levels of the structure
150 # to take into account the dimension of each component
151 # for example : the standard deviation of each component of 'inputRandomVector'
152 # in case of a RandomVector
153 scaling = NumericalPoint(dim)
154 scaling[0] = sqrt(inputRandomVector.getCovariance()[0,0])
155 scaling[1] = sqrt(inputRandomVector.getCovariance()[1,1])
156 scaling[2] = sqrt(inputRandomVector.getCovariance()[2,2])
157 scaling[3] = sqrt(inputRandomVector.getCovariance()[3,3])
158
159 inputSample *= scaling
160
161
162 # Translation of the nonReducedSample onto the center of the design of
       experiments
163 # center = mean point of the inputRandomVector distribution
164 center = inputRandomVector.getMean()
165 inputSample += center
166
167 # Get the number of points in the design of experiments
168 pointNumber = inputSample.getSize()
169
170 # Evaluate the ouput variable of interest on the design of experiments
171 outputSample = deviation(inputSample)
172
173
174 # Evaluate the range of the output variable of interest on that design of
       experiments
175 minValue = outputSample.getMin()
176 maxValue = outputSample.getMax()
177
178 print "From a composite  design of experiments of size = ", pointNumber
179 print "Levels = ", levels[0], ", ", levels[1], ", ", levels[2]
180 print "Min Value = ", minValue[0]
181 print "Max Value = ", maxValue[0]
182 print ""
```

```
183
184 ############################################################
185 # Min/Max study by random sampling
186 ############################################################
187
188 print "#####################################"
189 print " Min/Max study by random sampling"
190 print "#####################################"
191
192 pointNumber = 10000
193 print "From random sampling = ", pointNumber
194 outputSample2 = outputVariableOfInterest.getSample(pointNumber)
195
196 minValue2 = outputSample2.getMin()
197 maxValue2 = outputSample2.getMax()
198
199 print "Min Value = ", minValue2[0]
200 print "Max Value = ", maxValue2[0]
201 print ""
202
203
204
205
206
207 print ""
208 ###################################################
209 ### Random Study : central tendance of
210 ### the output variable of interest
211 ###################################################
212
213 print "################################################"
214 print "Random Study : central tendance of"
215 print "the output variable of interest"
216 print "################################################"
217 print ""
218
219 ######################################
220 # Taylor variance decomposition
221 ######################################
222
223 print "#################################"
224 print "Taylor variance decomposition"
225 print "#################################"
226 print ""
227
228 # We create a quadraticCumul algorithm
229 myQuadraticCumul = QuadraticCumul(outputVariableOfInterest)
230
```

```
231 # We compute the several elements provided by the quadratic cumul algorithm
232 # and evaluate the number of calculus needed
233 nbBefr = deviation.getEvaluationCallsNumber()
234
235 # Mean first order
236 meanFirstOrder = myQuadraticCumul.getMeanFirstOrder()[0]
237 nbAfter1 = deviation.getEvaluationCallsNumber()
238
239 # Mean second order
240 meanSecondOrder = myQuadraticCumul.getMeanSecondOrder()[0]
241 nbAfter2 = deviation.getEvaluationCallsNumber()
242
243 # Standard deviation
244 stdDeviation = sqrt(myQuadraticCumul.getCovariance()[0,0])
245 nbAfter3 = deviation.getEvaluationCallsNumber()
246
247 print "First order mean=", myQuadraticCumul.getMeanFirstOrder()[0]
248 print "Evaluation calls number = ", nbAfter1 - nbBefr
249 print "Second order mean=", myQuadraticCumul.getMeanSecondOrder()[0]
250 print "Evaluation calls number = ", nbAfter2 - nbAfter1
251 print "Standard deviation=", sqrt(myQuadraticCumul.getCovariance()[0,0])
252 print "Evaluation calls number = ", nbAfter3 - nbAfter2
253
254 print  "Importance factors="
255 for i in range(inputRandomVector.getDimension()) :
256   print inputDistribution.getDescription()[i], " = ", myQuadraticCumul.
        getImportanceFactors()[i]
257
258
259 ##############################
260 # Random sampling
261 ##############################
262
263 print "##########################"
264 print "Random sampling"
265 print "##########################"
266
267 size1 = 10000
268 output_Sample1 = outputVariableOfInterest.getSample(size1)
269 outputMean = output_Sample1.computeMean()
270 outputCovariance = output_Sample1.computeCovariance()
271
272 print "Sample size = ", size1
273 print "Mean from sample = ", outputMean[0]
274 print "Standard deviation from sample = ", sqrt(outputCovariance[0,0])
275 print ""
276
277
```

```
278 ############################
279 # Kernel Smoothing Fitting
280 ############################
281
282
283 print "############################"
284 print "# Kernel Smoothing Fitting"
285 print "############################"
286
287 # We generate a sample of the output variable
288 size = 10000
289 output_sample = outputVariableOfInterest.getSample(size)
290
291 # We build the kernel smoothing distribution
292 kernel = KernelSmoothing()
293 bw = kernel.computeSilvermanBandwidth(output_sample)
294 smoothed = kernel.build(output_sample, bw)
295 print "Sample size = ", size
296 print  "Kernel bandwidth=" , kernel.getBandwidth()[0]
297
298 # We draw the pdf and cdf from kernel smoothing
299 # Evaluate at best the range of the graph
300 mean_sample = output_sample.computeMean()[0]
301 standardDeviation_sample = sqrt(output_sample.computeCovariance()[0,0])
302 xmin = mean_sample - 4*standardDeviation_sample
303 xmax = mean_sample + 4*standardDeviation_sample
304
305 # Draw the PDF
306 smoothedPDF = smoothed.drawPDF(xmin, xmax, 251)
307 # Change the title
308 smoothedPDF.setTitle("Kernel smoothing of the deviation - PDF")
309 # Change the legend
310 smoothedPDF_draw = smoothedPDF.getDrawable(0)
311 title = "PDF from Normal kernel (" + str(size) + " data)"
312 smoothedPDF_draw.setLegend(title)
313 smoothedPDF.setDrawable(smoothedPDF_draw,0)
314 smoothedPDF.draw("smoothedPDF", 640, 480)
315
316 # Draw the CDF
317 smoothedCDF = smoothed.drawCDF(xmin, xmax, 251)
318 # Change the title
319 smoothedCDF.setTitle("Kernel smoothing of the deviation - CDF")
320 # Change the legend
321 smoothedCDF_draw = smoothedCDF.getDrawable(0)
322 title = "CDF from Normal kernel (" + str(size) + " data)"
323 smoothedCDF_draw.setLegend(title)
324 smoothedCDF.setDrawable(smoothedCDF_draw,0)
325 # Change the legend position
```

```
326 smoothedCDF.setLegendPosition("bottomright")
327 smoothedCDF.draw("smoothedCDF", 640, 480)
328
329 # In order to see the graph whithout creating the associated files
330 #View(smoothedCDF).show()
331 #View(smoothedPDF).show()
332
333 # Mean of the output variable of interest
334 print "Mean from kernel smoothing = ", smoothed.getMean()[0]
335 print ""
336
337 # Superposition of the kernel smoothing pdf and the gaussian one
338 # which mean and standard deviation are those of the output_sample
339 normalDist = NormalFactory().build(output_sample)
340 normalDistPDF = normalDist.drawPDF(xmin, xmax, 251)
341 normalDistPDFDrawable = normalDistPDF.getDrawable(0)
342 normalDistPDFDrawable.setColor('blue')
343 smoothedPDF.add(normalDistPDFDrawable)
344 smoothedPDF.draw("smoothedPDF_and_NormalPDF", 640, 480)
345
346 # In order to see the graph whithout creating the associated files
347 #View(smoothedPDF).show()
348
349 ##########################################################################
350 ### Probabilistic Study : threshold exceedance: deviation > 30cm
351 ##########################################################################
352
353 print ""
354 print "##########################################################################"
355 print "Probabilistic Study : threshold exceedance: deviation <-1cm"
356 print "##########################################################################"
357 print ""
358
359 ######
360 # FORM
361 ######
362
363 print "#####"
364 print "FORM"
365 print "#####"
366
367 # We create an Event from this RandomVector
368 # threshold has been defined in the kernel smoothing section
369 threshold = 30
370 myEvent = Event(outputVariableOfInterest, Greater(), threshold)
371 myEvent.setName("Deviation > 30 cm")
372
373 # We create a NearestPoint algorithm
```

```
374 myCobyla = Cobyla()
375 myCobyla.setMaximumIterationsNumber(1000)
376 myCobyla.setMaximumAbsoluteError(1.0e-10)
377 myCobyla.setMaximumRelativeError(1.0e-10)
378 myCobyla.setMaximumResidualError(1.0e-10)
379 myCobyla.setMaximumConstraintError(1.0e-10)
380
381 # We create a FORM algorithm
382 # The first parameter is a NearestPointAlgorithm
383 # The second parameter is an event
384 # The third parameter is a starting point for the design point research
385 meanVector = inputRandomVector.getMean()
386 myAlgoFORM = FORM(myCobyla, myEvent, meanVector)
387
388 # Get the number of times the limit state function has been evaluated so far
389 deviationCallNumberBeforeFORM = deviation.getEvaluationCallsNumber()
390
391 # Perform the simulation
392 myAlgoFORM.run()
393
394 # Get the number of times the limit state function has been evaluated so far
395 deviationCallNumberAfterFORM = deviation.getEvaluationCallsNumber()
396
397 # Stream out the result
398 resultFORM = myAlgoFORM.getResult()
399 print "FORM event probability=", resultFORM.getEventProbability()
400 print "Number of evaluations of the limit state function = ",
      deviationCallNumberAfterFORM - deviationCallNumberBeforeFORM
401 print "Generalized reliability index=", resultFORM.
      getGeneralisedReliabilityIndex()
402 print "Standard space design point="
403 for i in range(inputRandomVector.getDimension()) :
404   print inputDistribution.getDescription()[i], " = ", resultFORM.
        getStandardSpaceDesignPoint()[i]
405 print "Physical space design point="
406 for i in range(inputRandomVector.getDimension()) :
407   print inputDistribution.getDescription()[i], " = ", resultFORM.
        getPhysicalSpaceDesignPoint()[i]
408
409 print "Importance factors="
410 for i in range(inputRandomVector.getDimension()) :
411   print inputDistribution.getDescription()[i], " = ", resultFORM.
        getImportanceFactors()[i]
412
413 print "Hasofer reliability index=", resultFORM.getHasoferReliabilityIndex()
414 print ""
415
416 # Graph 1 : Importance Factors graph */
```

```
417 importanceFactorsGraph = resultFORM.drawImportanceFactors()
418 title = "FORM Importance factors - "+ myEvent.getName()
419 importanceFactorsGraph.setTitle( title)
420 importanceFactorsGraph.draw("ImportanceFactorsDrawingFORM", 640, 480)
421
422 # In order to see the graph whithout creating the associated files
423 #View(importanceFactorsGraph).show()
424
425
426 ######
427 # MC
428 ######
429
430 print "#############"
431 print "Monte Carlo"
432 print "#############"
433 print ""
434
435
436 maximumOuterSampling = 40000
437 blockSize = 100
438 coefficientOfVariation = 0.10
439
440 # We create a Monte Carlo algorithm
441 myAlgoMonteCarlo = MonteCarlo(myEvent)
442 myAlgoMonteCarlo.setMaximumOuterSampling(maximumOuterSampling)
443 myAlgoMonteCarlo.setBlockSize(blockSize)
444 myAlgoMonteCarlo.setMaximumCoefficientOfVariation(coefficientOfVariation)
445
446 # Define the HistoryStrategy to store the values of the probability estimator
447 # and the variance estimator
448 # used ot draw the convergence graph
449 # Full strategy
450 myAlgoMonteCarlo.setConvergenceStrategy(Full())
451
452 # Perform the simulation
453 myAlgoMonteCarlo.run()
454
455 # Display number of iterations and number of evaluations
456 # of the limit state function
457 print "Number of evaluations of the limit state function = ", myAlgoMonteCarlo.
      getResult().getOuterSampling()* myAlgoMonteCarlo.getResult().getBlockSize()
458
459 # Display the Monte Carlo probability of 'myEvent'
460 print "Monte Carlo probability estimation = ", myAlgoMonteCarlo.getResult().
      getProbabilityEstimate()
461
462 # Display the variance of the Monte Carlo probability estimator
```

```
463 print "Variance of the Monte Carlo probability estimator = ", myAlgoMonteCarlo.
       getResult().getVarianceEstimate()
464
465 # Display the confidence interval length centered around
466 # the MonteCarlo probability MCProb
467 # IC = [MCProb - 0.5*length, MCProb + 0.5*length]
468 # level 0.95
469
470 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), "]"
471 print ""
472
473 # Draw the convergence graph and the confidence intervalle of level alpha
474 alpha = 0.90
475 convergenceGraphMonteCarlo = myAlgoMonteCarlo.drawProbabilityConvergence(alpha)
476 # In order to see the graph whithout creating the associated files
477 #View(convergenceGraphMonteCarlo).show()
478
479 # Create the file .EPS
480 convergenceGraphMonteCarlo.draw("convergenceGrapheMonteCarlo", 640, 480)
481 #View(convergenceGraphMonteCarlo).show()
482
483
484 ############################
485 # Directional Sampling
486 ############################
487
488 print "############################"
489 print "Directional Sampling"
490 print "############################"
491 print " "
492
493 # Directional sampling from an event (slow and safe strategy by default)
494
495 # We create a Directional Sampling algorithm */
496 myAlgoDirectionalSim = DirectionalSampling(myEvent)
497 myAlgoDirectionalSim.setMaximumOuterSampling(maximumOuterSampling * blockSize)
498 myAlgoDirectionalSim.setBlockSize(1)
499 myAlgoDirectionalSim.setMaximumCoefficientOfVariation(coefficientOfVariation)
500
501 # Define the HistoryStrategy to store the values of the probability estimator
502 # and the variance estimator
503 # used ot draw the convergence graph
504 # Full strategy
505 myAlgoDirectionalSim.setConvergenceStrategy(Full())
```

```
506
507 # Save the number of calls to the limit state fucntion, its gradient and hessian
        already done
508 deviationCallNumberBefore = deviation.getEvaluationCallsNumber()
509 deviationGradientCallNumberBefore = deviation.getGradientCallsNumber()
510 deviationHessianCallNumberBefore = deviation.getHessianCallsNumber()
511
512 # Perform the simulation */
513 myAlgoDirectionalSim.run()
514
515 # Save the number of calls to the limit state fucntion, its gradient and hessian
        already done
516 deviationCallNumberAfter = deviation.getEvaluationCallsNumber()
517 deviationGradientCallNumberAfter = deviation.getGradientCallsNumber()
518 deviationHessianCallNumberAfter = deviation.getHessianCallsNumber()
519
520 # Display number of iterations and number of evaluations
521 # of the limit state function
522 print "Number of evaluations of the limit state function = ",
       deviationCallNumberAfter - deviationCallNumberBefore
523
524 # Display the Directional Simumation probability of 'myEvent'
525 print "Directional Sampling probability estimation = ", myAlgoDirectionalSim.
       getResult().getProbabilityEstimate()
526
527 # Display the variance of the Directional Simumation probability estimator
528 print "Variance of the Directional Sampling probability estimator = ",
       myAlgoDirectionalSim.getResult().getVarianceEstimate()
529
530 # Display the confidence interval length centered around
531 # the Directional Simumation probability DSProb
532 # IC = [DSProb - 0.5*length, DSProb + 0.5*length]
533 # level 0.95
534 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), "]"
535 print ""
536
537
538 # Draw the convergence graph and the confidence intervalle of level alpha
539 alpha = 0.90
540 convergenceGraphDS = myAlgoDirectionalSim.drawProbabilityConvergence(alpha)
541 # In order to see the graph whithout creating the associated files
542 #View(convergenceGraphDS).show()
543
544 # Create the file .EPS
```

```
545 convergenceGraphDS.draw("convergenceGrapheDS", 640, 480)
546 #View(convergenceGraphDS).show()
547
548 ######################
549 # Importance Sampling
550 ######################
551
552
553 print "######################"
554 print "Importance Sampling"
555 print "######################"
556 print ""
557
558 maximumOuterSampling = 40000
559 blockSize = 1
560 standardSpaceDesignPoint = resultFORM.getStandardSpaceDesignPoint()
561 mean = standardSpaceDesignPoint
562 sigma = NumericalPoint(4, 1.0)
563 importanceDistribution = Normal(mean, sigma, CorrelationMatrix(4))
564
565 myStandardEvent = StandardEvent(myEvent)
566
567 myAlgoImportanceSampling = ImportanceSampling(myStandardEvent,
        importanceDistribution)
568 myAlgoImportanceSampling.setMaximumOuterSampling(maximumOuterSampling)
569 myAlgoImportanceSampling.setBlockSize(blockSize)
570 myAlgoImportanceSampling.setMaximumCoefficientOfVariation(coefficientOfVariation
        )
571
572 # Define the HistoryStrategy to store the values of the probability estimator
573 # and the variance estimator
574 # used ot draw the convergence graph
575 # Full strategy
576 myAlgoImportanceSampling.setConvergenceStrategy(Full())
577
578 # Perform the simulation
579 myAlgoImportanceSampling.run()
580
581 # Display number of iterations and number of evaluations
582 # of the limit state function
583 print "Number of evaluations of the limit state function = ",
        myAlgoImportanceSampling.getResult().getOuterSampling()*
        myAlgoImportanceSampling.getResult().getBlockSize()
584
585 # Display the Importance Sampling probability of 'myEvent'
586 print "Importance Sampling probability estimation = ", myAlgoImportanceSampling.
        getResult().getProbabilityEstimate()
587
```

```
588 # Display the variance of the Importance Sampling probability estimator
589 print "Variance of the Importance Sampling probability estimator = ",
       myAlgoImportanceSampling.getResult().getVarianceEstimate()
590
591 # Display the confidence interval length centered around
592 # the ImportanceSampling probability ISProb
593 # IC = [ISProb - 0.5*length, ISProb + 0.5*length]
594 # level 0.95
595 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), "]"
596
597 # Draw the convergence graph and the confidence intervalle of level alpha
598 alpha = 0.90
599 convergenceGraphIS = myAlgoImportanceSampling.drawProbabilityConvergence(alpha)
600 # In order to see the graph whithout creating the associated files
601 #View(convergenceGraphIS).show()
602
603 # Create the file .EPS
604 convergenceGraphIS.draw("convergenceGrapheIS", 640, 480)
605 #View(convergenceGraphIS).show()
606
607
608
609
610 ################################################
611 # Response surface : Polynomial expansion chaos
612 ################################################
613
614 print "############################"
615 print "Polynomial expansion chaos"
616 print "############################"
617 print " "
618
619 ################################################################
620 # STEP 1 : Construction of the multivariate orthonormal basis
621
622 # Dimension of the input random vector
623 dim = 4
624
625 # Create the univariate polynomial family collection
626 # which regroups the polynomial families for each direction
627 polyColl = PolynomialFamilyCollection(dim)
628
629 # Variable E
630 #Jacobi(alpha, beta) <=> Beta(\beta + 1, \alpha + \beta + 2, -1, 1)
```

```
631 alphaJ = 1.27
632 betaJ = -0.07
633 jacobiFamily = JacobiFactory(alphaJ, betaJ)
634 polyColl[0] = jacobiFamily
635
636
637 # Variable F
638 # Laguerre(k) <=> Gamma(k+1,1,0) (parametrage ppal)
639 kLaguerre = 1.78
640 laguerreFamily = LaguerreFactory(kLaguerre)
641 polyColl[1] = laguerreFamily
642
643 # Variable L
644 # Legendre <=> Unif(-1,1)
645 legendreFamily = LegendreFactory()
646 polyColl[2] = legendreFamily
647
648 # Variable E
649 # Jacobi(alpha, beta) <=> Beta(\beta + 1, \alpha + \beta + 2, -1, 1)
650 alphaJ2 = 0.5
651 betaJ2 = 1.5
652 jacobiFamily2 = JacobiFactory(alphaJ2, betaJ2)
653 polyColl[3] = jacobiFamily2
654
655
656 # Create the multivariate orthonormal basis
657 # which is the the cartesian product of the univariate basis
658 multivariateBasis = OrthogonalProductPolynomialFactory(polyColl,
       LinearEnumerateFunction(dim))
659
660 # Build a term of the basis as a NumericalMathFunction
661 # Generally, we do not need to construct manually any term,
662 # all terms are constructed automatically by a strategy of construction of the
       basis
663 i = 5
664 Psi_i = multivariateBasis.build(i)
665
666 # Get the measure mu associated to the multivariate basis
667 distributionMu = multivariateBasis.getMeasure()
668
669 ##############################################################################
670 # STEP 2 : Truncature strategy of the multivariate orthonormal basis
671
672 # CleaningStrategy :
673 # among the maximumConsideredTerms = 500 first polynoms,
674 # those which have the mostSignificant = 50 most significant contributions
675 # with significance criterion significanceFactor = 10^(-4)
676 # The True boolean indicates if we are interested
```

```
677 # in the online monitoring of the current basis update
678 # (removed or added coefficients)
679 maximumConsideredTerms = 500
680 mostSignificant = 50
681 significanceFactor = 1.0e-4
682 truncatureBasisStrategy = CleaningStrategy(multivariateBasis,
       maximumConsideredTerms, mostSignificant, significanceFactor, True)
683
684 ###############################################################
685 # STEP 3 : Evaluation strategy of the approximation coefficients
686
687 # The technique proposed is the Least Squares technique
688 # where the points come from an design of experiments
689 # Here : the Monte Carlo sampling technique
690 sampleSize = 10000
691 evaluationCoeffStrategy = LeastSquaresStrategy(MonteCarloExperiment(sampleSize))
692
693 # STEP 4 : Creation of the Functional Chaos Algorithm
694
695 # FunctionalChaosAlgorithm :
696 # combination of the model : limitStateFunction
697 # the distribution of the input random vector : Xdistribution
698 # the truncature strategy of the multivariate basis
699 # and the evaluation strategy of the coefficients
700 polynomialChaosAlgorithm = FunctionalChaosAlgorithm(deviation, inputDistribution
       , truncatureBasisStrategy, ProjectionStrategy(evaluationCoeffStrategy))
701
702 ###########################################################
703 # Run and results exploitation
704
705 # Perform the simulation
706 polynomialChaosAlgorithm.run()
707
708 # Stream out the result
709 polynomialChaosResult = polynomialChaosAlgorithm.getResult()
710
711 # Get the polynomial chaos coefficients
712 coefficients = polynomialChaosResult.getCoefficients()
713
714 # Get the meta model as a NumericalMathFunction
715 metaModel = polynomialChaosResult.getMetaModel()
716
717 # Get the indices of the selected polynomials : K
718 subsetK = polynomialChaosResult.getIndices()
719
720 # Get the composition of the polynomials
721 # of the truncated multivariate basis
722 for i in range(subsetK.getSize()) :
```

```
723     print "Polynomial number ", i, " in truncated basis <-> polynomial number ",
            subsetK[i], " = ", LinearEnumerateFunction(dim)(subsetK[i]), " in complete
            basis"
724
725 # Get the multivariate basis
726 # as a colletion of NumericalMathFunction
727 multivariateBasisCollection = polynomialChaosResult.getReducedBasis()
728
729 # Get the distribution of variables Z
730 mu = polynomialChaosResult.getDistribution()
731 print "Distribution in the tansformed variables = ", mu
732 print ""
733
734 # Get the composed model which is the model of the reduced variables Z
735 composedModel = polynomialChaosResult.getComposedModel()
736
737 # Define the new random vector
738 newOutputVariableOfInterest = RandomVector(polynomialChaosResult)
739
740 # Get the mean and variance of the meta model
741
742 print "Mean =", newOutputVariableOfInterest.getMean()
743 print "Standard deviation =", sqrt(newOutputVariableOfInterest.getCovariance()
        [0,0])
744 print ""
745
746
747
748 ############################################
749 # Graphs validation
750
751
752 # Graph 1 : cloud
753
754 # Generate a NumericalSample of the input random vector
755 # Evaluate the meta model and the real model
756 # draw the coulds (metamodel, real model)
757 # Verify points are on the first diagonal
758 sizeX = 500
759 Xsample = inputDistribution.getSample(sizeX)
760
761 modelSample = deviation(Xsample)
762 metaModelSample = metaModel(Xsample)
763
764 sampleMixed = NumericalSample(sizeX,2)
765 for i in range(sizeX) :
766   sampleMixed[i, 0] = modelSample[i, 0]
767   sampleMixed[i, 1] = metaModelSample[i, 0]
```

```
768
769 legend = str(sizeX) + " realizations"
770 comparisonCurve = Curve(modelSample, modelSample, "model")
771 comparisonCurve.setColor("red")
772
773 comparisonCloud = Cloud(sampleMixed, "blue", "fsquare", legend)
774 graphCloud = Graph("Polynomial chaos expansion", "model", "meta model", True, "
       topleft")
775 graphCloud.add(comparisonCurve)
776 graphCloud.add(comparisonCloud)
777
778 #View(graphCloud).show()
779 graphCloud.draw("PCE_comparisonModels")
780
781
782 # Graph 2 : polynoms family graphs
783
784 degreeMax = 5
785 pointNumber = 101
786 colorList = Drawable.GetValidColors()
787
788 # Jacobi for E
789 xMinJacobi = -1
790 xMaxJacobi = 1
791 titleJacobi = "Jacobi(" + str(alphaJ) + ", " + str(betaJ) + ") polynomials"
792 graphJacobi = Graph(titleJacobi, "z", "polynomial values", True, "topleft")
793 for i in range(degreeMax) :
794   graphJacobi_temp = jacobiFamily.build(i).draw(xMinJacobi, xMaxJacobi,
       pointNumber)
795   graphJacobi_temp_draw = graphJacobi_temp.getDrawable(0)
796   legend = "degree " + str(i)
797   graphJacobi_temp_draw.setLegend(legend)
798   graphJacobi_temp_draw.setColor(colorList[i])
799   graphJacobi.add(graphJacobi_temp_draw)
800   #View(graphJacobi).show()
801   graphJacobi.draw("PCE_JacobiPolynomials_VariableE")
802
803 # Laguerre for F
804 xMinLaguerre = 0
805 xMaxLaguerre = 10
806 titleLaguerre = "Laguerre(" + str(kLaguerre) + ") polynomials"
807 graphLaguerre = Graph(titleLaguerre, "z", "polynomial values", True, "topleft")
808 for i in range(degreeMax) :
809   graphLaguerre_temp = laguerreFamily.build(i).draw(xMinLaguerre, xMaxLaguerre,
       pointNumber)
810   graphLaguerre_temp_draw = graphLaguerre_temp.getDrawable(0)
811   legend = "degree " + str(i)
812   graphLaguerre_temp_draw.setLegend(legend)
```

```
813     graphLaguerre_temp_draw.setColor(colorList[i])
814     graphLaguerre.add(graphLaguerre_temp_draw)
815     #View(graphLaguerre).show()
816     graphLaguerre.draw("PCE_LaguerrePolynomials_VariableF")
817
818 # Legendre for L
819 xMinLegendre = -1
820 xMaxLegendre = 1
821 titleLegendre = "Legendre polynomials"
822 graphLegendre = Graph(titleLegendre, "z", "polynomial values", True, "topright")
823 for i in range(degreeMax) :
824     graphLegendre_temp = laguerreFamily.build(i).draw(xMinLegendre, xMaxLegendre,
          pointNumber)
825     graphLegendre_temp_draw = graphLegendre_temp.getDrawable(0)
826     legend = "degree " + str(i)
827     graphLegendre_temp_draw.setLegend(legend)
828     graphLegendre_temp_draw.setColor(colorList[i])
829     graphLegendre.add(graphLegendre_temp_draw)
830     #View(graphLegendre).show()
831     graphLegendre.draw("PCE_LegendrePolynomials_VariableL")
832
833 # Jacobi for I
834 xMinJacobi2 = -1
835 xMaxJacobi2 = 1
836 titleJacobi2 = "Jacobi(" + str(alphaJ2) + ", " + str(betaJ2) + ") polynomials"
837 graphJacobi2 = Graph(titleJacobi2, "z", "polynomial values", True, "topright")
838 for i in range(degreeMax) :
839     graphJacobi2_temp = jacobiFamily2.build(i).draw(xMinJacobi2, xMaxJacobi2,
          pointNumber)
840     graphJacobi2_temp_draw = graphJacobi2_temp.getDrawable(0)
841     legend = "degree " + str(i)
842     graphJacobi2_temp_draw.setLegend(legend)
843     graphJacobi2_temp_draw.setColor(colorList[i])
844     graphJacobi2.add(graphJacobi2_temp_draw)
845     #View(graphJacobi2).show()
846     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 =  5.38682360418
14 Max Value =  52.7553886011
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  =  0.781344650859
35 L  =  0.0145457110929
36 I  =  0.0550127570943
37 ########################
38 Random sampling
39 ########################
40 Sample size =  10000
41 Mean from sample =  12.6090425333
42 Standard deviation from sample =  4.35926574946
43
44 ############################
45 # Kernel Smoothing Fitting
46 ############################
47 Sample size =  10000
48 Kernel bandwidth= 0.555537787263
49 Mean from kernel smoothing =  12.6194394108
50
51
52 ###################################################################
53 Probabilistic Study : threshold exceedance: deviation <-1cm
54 ###################################################################
55
56 #####
```

```
FORM
#####
FORM event probability= 0.0067098042649
Number of evaluations of the limit state function =  194
Generalized reliability index= 2.47243507875
Standard space design point=
E  =  -0.602386524562
F  =  2.31055510668
L  =  0.3557936838
I  =  -0.533677474898
Physical space design point=
E  =  30327158.1435
F  =  61318.4682096
L  =  256.390024602
I  =  378.63472752
Importance factors=
E  =  0.0586820272304
F  =  0.863350757047
L  =  0.0204715666874
I  =  0.0574956490355
Hasofer reliability index= 2.47243507875

############
Monte Carlo
############

Number of evaluations of the limit state function =  18300
Monte Carlo probability estimation =  0.00551912568306
Variance of the Monte Carlo probability estimator =  3.02926673715e-07
0.95 Confidence Interval = [ 0.00444038551844 ,  0.00659786584768 ]

########################
Directional Sampling
########################

Number of evaluations of the limit state function =  14378
Directional Sampling probability estimation =  0.004890168236
Variance of the Directional Sampling probability estimator =  2.39111788287e-07
0.95 Confidence Interval = [ 0.0039317643085 ,  0.0058485721635 ]

####################
Importance Sampling
####################

Number of evaluations of the limit state function =  324
Importance Sampling probability estimation =  0.00549138956736
Variance of the Importance Sampling probability estimator =  2.99203917702e-07
0.95 Confidence Interval = [ 0.00441929837308 ,  0.00656348076164 ]
```

```
105 ############################
106 Polynomial expansion chaos
107 ############################
108
109 Polynomial number 0 in truncated basis <-> polynomial number 0 = [0,0,0,0]
        in complete basis
110 Polynomial number 1 in truncated basis <-> polynomial number 1 = [1,0,0,0]
        in complete basis
111 Polynomial number 2 in truncated basis <-> polynomial number 2 = [0,1,0,0]
        in complete basis
112 Polynomial number 3 in truncated basis <-> polynomial number 3 = [0,0,1,0]
        in complete basis
113 Polynomial number 4 in truncated basis <-> polynomial number 4 = [0,0,0,1]
        in complete basis
114 Polynomial number 5 in truncated basis <-> polynomial number 5 = [2,0,0,0]
        in complete basis
115 Polynomial number 6 in truncated basis <-> polynomial number 6 = [1,1,0,0]
        in complete basis
116 Polynomial number 7 in truncated basis <-> polynomial number 7 = [1,0,1,0]
        in complete basis
117 Polynomial number 8 in truncated basis <-> polynomial number 8 = [1,0,0,1]
        in complete basis
118 Polynomial number 9 in truncated basis <-> polynomial number 9 = [0,2,0,0]
        in complete basis
119 Polynomial number 10 in truncated basis <-> polynomial number 10 =
       [0,1,1,0] in complete basis
120 Polynomial number 11 in truncated basis <-> polynomial number 11 =
       [0,1,0,1] in complete basis
121 Polynomial number 12 in truncated basis <-> polynomial number 12 =
       [0,0,2,0] in complete basis
122 Polynomial number 13 in truncated basis <-> polynomial number 13 =
       [0,0,1,1] in complete basis
123 Polynomial number 14 in truncated basis <-> polynomial number 14 =
       [0,0,0,2] in complete basis
124 Polynomial number 15 in truncated basis <-> polynomial number 15 =
       [3,0,0,0] in complete basis
125 Polynomial number 16 in truncated basis <-> polynomial number 16 =
       [2,1,0,0] in complete basis
126 Polynomial number 17 in truncated basis <-> polynomial number 17 =
       [2,0,1,0] in complete basis
127 Polynomial number 18 in truncated basis <-> polynomial number 18 =
       [2,0,0,1] in complete basis
128 Polynomial number 19 in truncated basis <-> polynomial number 19 =
       [1,2,0,0] in complete basis
129 Polynomial number 20 in truncated basis <-> polynomial number 20 =
       [1,1,1,0] in complete basis
130 Polynomial number 21 in truncated basis <-> polynomial number 21 =
       [1,1,0,1] in complete basis
```

```
131 Polynomial number 22 in truncated basis <-> polynomial number 23 =
      [1,0,1,1] in complete basis
132 Polynomial number 23 in truncated basis <-> polynomial number 24 =
      [1,0,0,2] in complete basis
133 Polynomial number 24 in truncated basis <-> polynomial number 25 =
      [0,3,0,0] in complete basis
134 Polynomial number 25 in truncated basis <-> polynomial number 26 =
      [0,2,1,0] in complete basis
135 Polynomial number 26 in truncated basis <-> polynomial number 27 =
      [0,2,0,1] in complete basis
136 Polynomial number 27 in truncated basis <-> polynomial number 29 =
      [0,1,1,1] in complete basis
137 Polynomial number 28 in truncated basis <-> polynomial number 30 =
      [0,1,0,2] in complete basis
138 Polynomial number 29 in truncated basis <-> polynomial number 31 =
      [0,0,3,0] in complete basis
139 Polynomial number 30 in truncated basis <-> polynomial number 33 =
      [0,0,1,2] in complete basis
140 Polynomial number 31 in truncated basis <-> polynomial number 34 =
      [0,0,0,3] in complete basis
141 Polynomial number 32 in truncated basis <-> polynomial number 36 =
      [3,1,0,0] in complete basis
142 Polynomial number 33 in truncated basis <-> polynomial number 39 =
      [2,2,0,0] in complete basis
143 Polynomial number 34 in truncated basis <-> polynomial number 45 =
      [1,3,0,0] in complete basis
144 Polynomial number 35 in truncated basis <-> polynomial number 55 =
      [0,4,0,0] in complete basis
145 Polynomial number 36 in truncated basis <-> polynomial number 56 =
      [0,3,1,0] in complete basis
146 Polynomial number 37 in truncated basis <-> polynomial number 57 =
      [0,3,0,1] in complete basis
147 Polynomial number 38 in truncated basis <-> polynomial number 61 =
      [0,1,3,0] in complete basis
148 Polynomial number 39 in truncated basis <-> polynomial number 63 =
      [0,1,1,2] in complete basis
149 Polynomial number 40 in truncated basis <-> polynomial number 66 =
      [0,0,3,1] in complete basis
150 Polynomial number 41 in truncated basis <-> polynomial number 105 =
      [0,5,0,0] in complete basis
151 Polynomial number 42 in truncated basis <-> polynomial number 106 =
      [0,4,1,0] in complete basis
152 Polynomial number 43 in truncated basis <-> polynomial number 120 =
      [0,0,5,0] in complete basis
153 Polynomial number 44 in truncated basis <-> polynomial number 182 =
      [0,6,0,0] in complete basis
154 Polynomial number 45 in truncated basis <-> polynomial number 183 =
      [0,5,1,0] in complete basis
```

```
155 Polynomial number 46 in truncated basis <-> polynomial number 294 =
      [0,7,0,0] in complete basis
156 Polynomial number 47 in truncated basis <-> polynomial number 295 =
      [0,6,1,0] in complete basis
157 Polynomial number 48 in truncated basis <-> polynomial number 322 =
      [0,0,7,0] in complete basis
158 Polynomial number 49 in truncated basis <-> polynomial number 450 =
      [0,8,0,0] in complete basis
159 Distribution in the tansformed variables = ComposedDistribution(Beta(r = 0.93,
      t = 3.2, a = 2.8e+07, b = 4.8e+07), LogNormal(muLog = 9.46206, sigmaLog =
      0.554513, gamma = 15000), Uniform(a = 250, b = 260), Beta(r = 2.5, t = 4, a =
       310, b = 450), NormalCopula(R = [[ 1          0          0          0          ]
160   [ 0          1          0          0          ]
161   [ 0          0          1          -0.209057  ]
162   [ 0          0          -0.209057  1          ]]))
163
164 Mean = [12.626]
165 Standard deviation = 4.3612562743
```

1.9 Figures

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

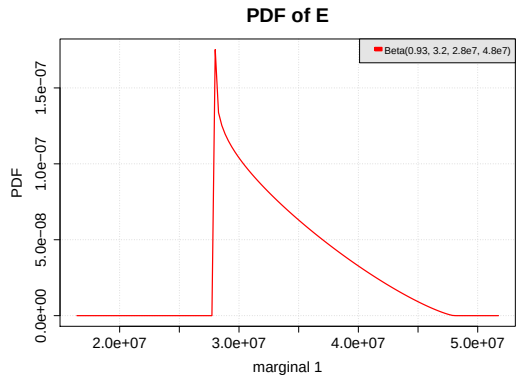


Figure 2: Probability density function of the parameter E

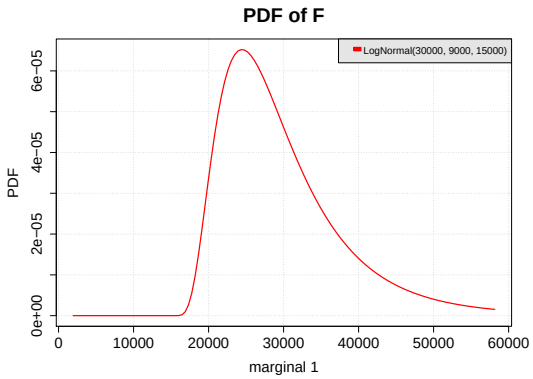


Figure 3: Probability density function of the parameter F

The probability density function (PDF) and the cumulative density function (CDF) of the deviation fiited with the kernel smoothing metid 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 12.
Figures (13) to (17) contain the graphs :

- Graph 1 : the drawings of the fith 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.

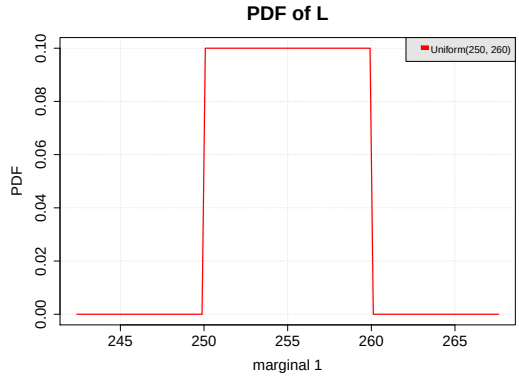


Figure 4: PDFof the parameter L

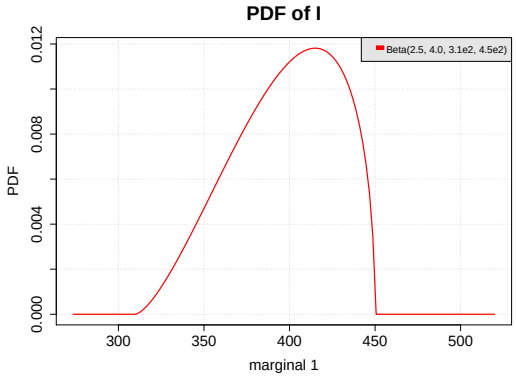


Figure 5: PDF of the parameter I

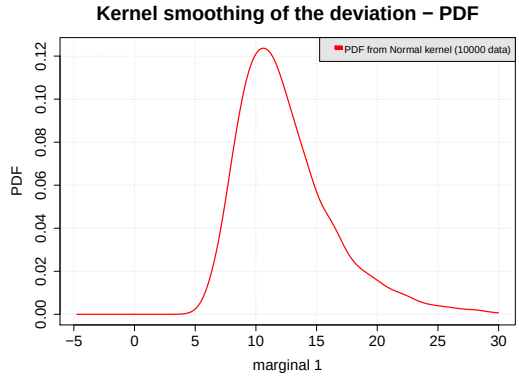


Figure 6: PDF of the deviation with the kernel smoothing method.

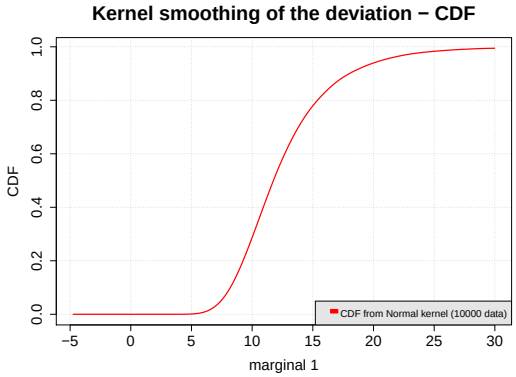


Figure 7: CDF of the deviation with the kernel smoothing method.

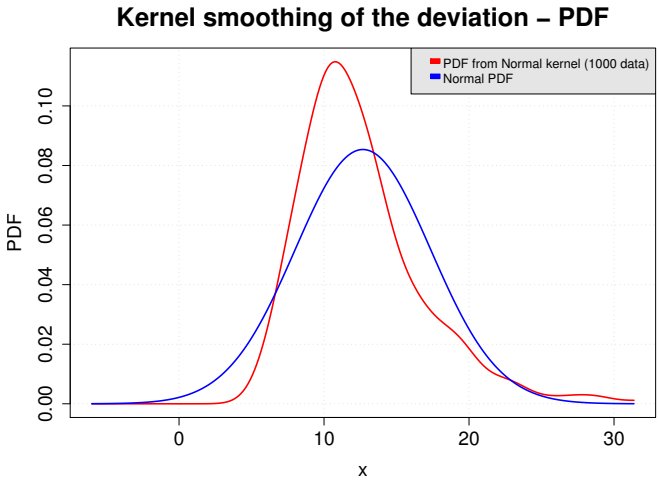


Figure 8: Superposition of the kernel smoothed density function and the normal fitted from the same sample.

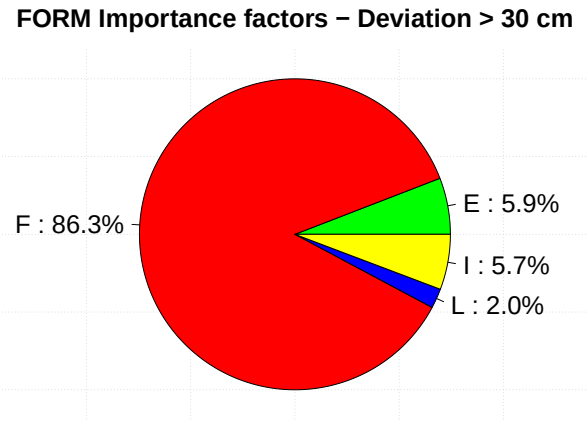


Figure 9: FORM importance factors of the event : deviation ¿ 30 cm.

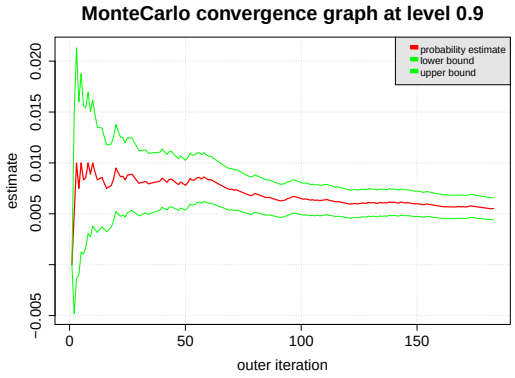


Figure 10: Monte Carlo convergence graph.

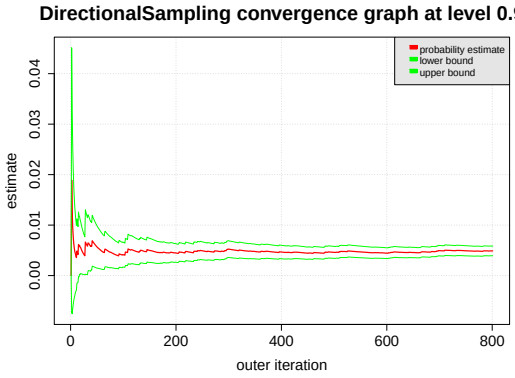


Figure 11: Directional Sampling convergence graph.

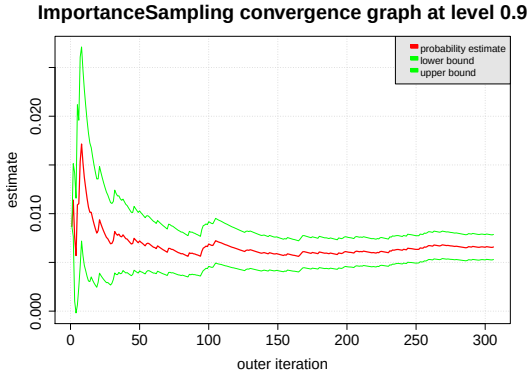


Figure 12: Importance sampling convergence graph.

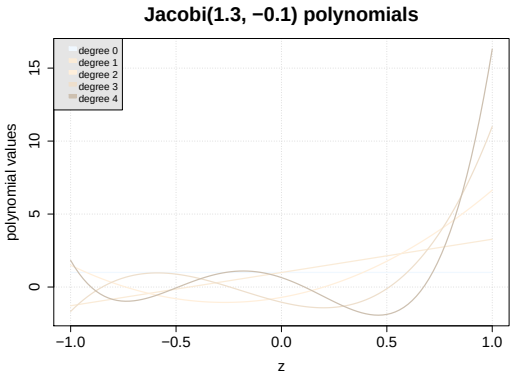


Figure 13: The 5-th first polynomials of the Jacobi family associated to the variable E.

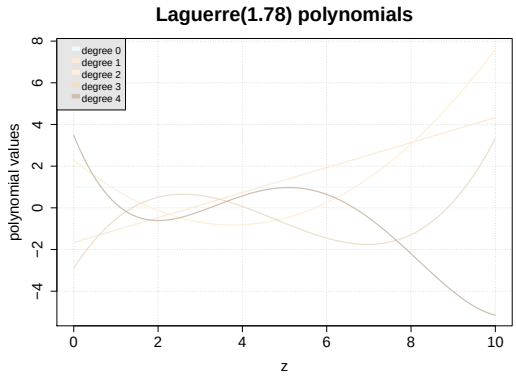


Figure 14: The 5-th first polynomials of the Laguerre family associated to the variable F.

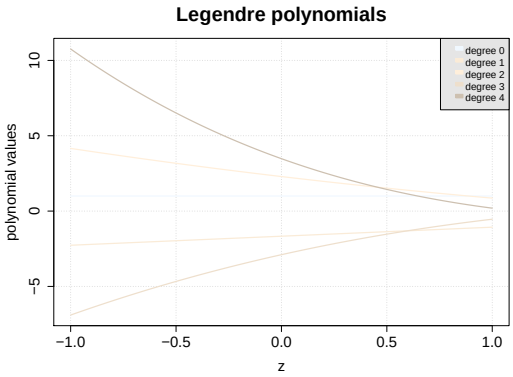


Figure 15: The 5-th first polynomials of the Legendre associated to the variable L.

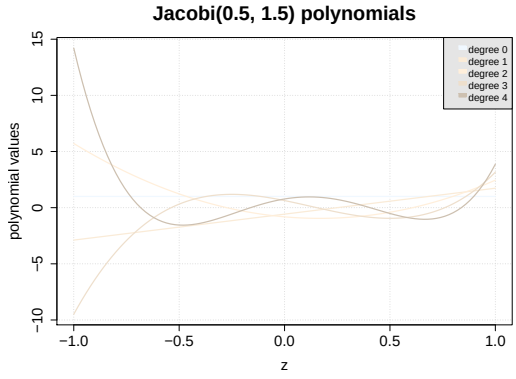


Figure 16: The 5-th first polynomials of the Jacobi family associated to the variable I.

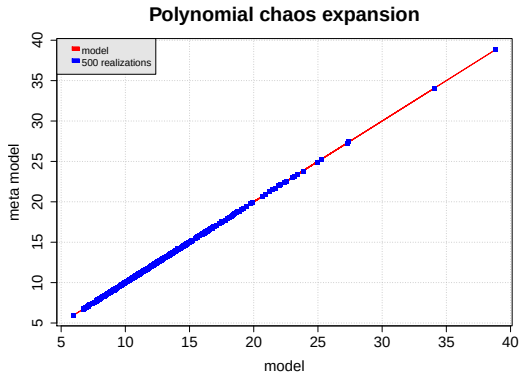


Figure 17: Comparison of values from the model and the polynomial chaos meta model.

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 benefical 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 :h ere, 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 tendancy 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 194 evaluations of the limit state function whereas the Monte Carlo probability required 18300 evaluations and the Directional Sampling one 14378 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 recommanded 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 10000 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 18. 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 oblical point load P such that the angle between P and bar #3 is equal to θ.

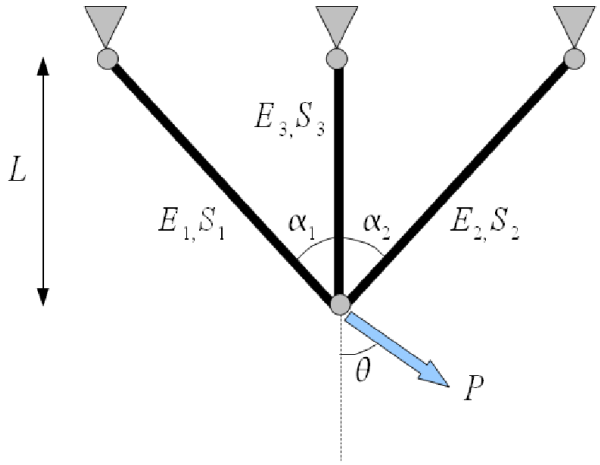


Figure 18: Elastic three-bar truss structure

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:

$$N_1^v = \frac{\sin(\alpha_2+\alpha_1)}{\sin(\alpha_2)}\left[\frac{E_3S_3}{\cos(\alpha_1)E_1S_1} + \frac{E_3S_3}{\cos(\alpha_2)E_2S_2}\left(\frac{\sin(\alpha_1)}{\sin(\alpha_2)}\right)^2 + \left(\frac{\sin(\alpha_1+\alpha_2)}{\sin(\alpha_2)}\right)^2\right]^{-1} P_v = C_v\, P_v \quad (1)$$

$$N_1^h = \frac{\frac{\sin(\alpha_1)}{\sin(\alpha_2)^2\cos(\alpha_2)E_2S_2} + \frac{\sin(\alpha_1+\alpha_2)\cos(\alpha_2)}{\sin(\alpha_2)E_3S_3}}{\frac{1}{\cos(\alpha_1)E_1S_1} + \frac{1}{\cos(\alpha_2)E_2S_2}\left(\frac{\sin(\alpha_1)}{\sin(\alpha_2)}\right)^2 + \left(\frac{\sin(\alpha_1+\alpha_2)}{\sin(\alpha_2)}\right)^2\frac{1}{E_3S_3}} P_h = C_h\, P_h \quad (2)$$

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

$$\delta_v = \left[\frac{\alpha_v^2}{E_1S_1\cos(\alpha_1)} + \frac{\alpha_v^2\sin^2(\alpha_1)}{E_2S_2\cos(\alpha_2)\sin^2(\alpha_2)} + \frac{\left(1-\frac{\sin(\alpha_1+\alpha_2)}{\sin(\alpha_2)}\right)^2}{E_3S_3}\right] P_v\, L \quad (3)$$

$$\delta_h = \left[\frac{\alpha_v^2}{E_1S_1\cos(\alpha_1)} + \frac{\left(\alpha_h\frac{\sin(\alpha_1)}{\sin(\alpha_2)}\right)^2}{E_2S_2\cos(\alpha_2)} + \frac{\left(\alpha_h\frac{\sin(\alpha_1+\alpha_2)}{\sin(\alpha_2)} - \frac{\cos(\alpha_2)}{\sin(\alpha_2)}\right)^2}{E_3S_3}\right] P_h\, L \quad (4)$$

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

$$\delta \ = \ \sqrt{\delta_h^2 \ + \ \delta_v^2} \tag{5}$$

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:

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

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	$1.5 \cdot 10^{-3}$ m^2	5%
P	Gumbel	$2.5 \cdot 10^5$ 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:

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

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 $\underline{\xi} = \{\xi_1, \ldots, \xi_{10}\}$ as follows:

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

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

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

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

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

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 $a_{\underline{\alpha}}$'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:

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

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 19.

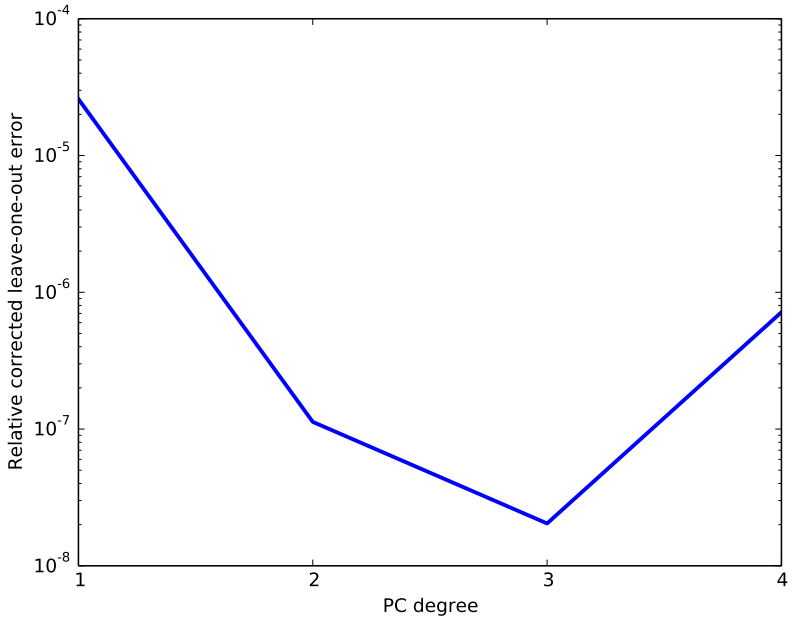


Figure 19: Corrected leave-one-out errors associated with the various polynomial chaos approximations

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:

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

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 20.

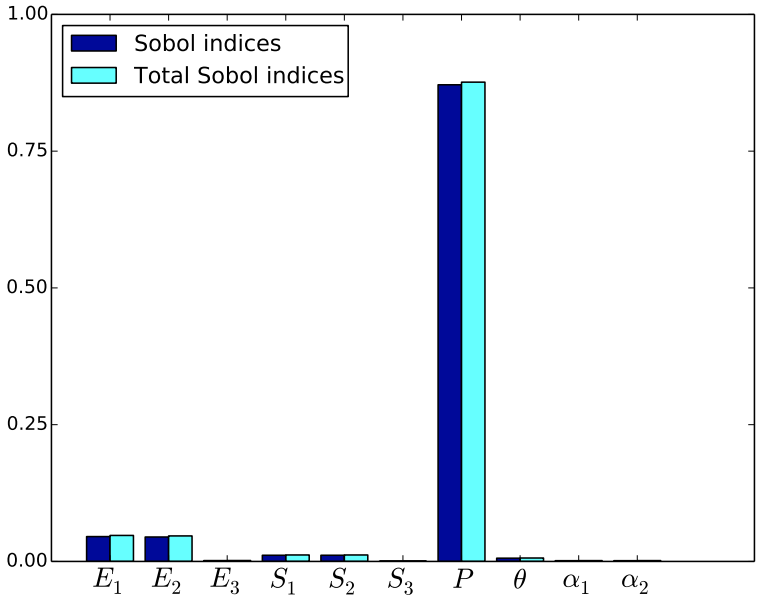


Figure 20: Single-effect and total sensitivity indices of the truss displacement to the various input parameters

It is observed that both kinds of indices strongly coincidate, 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 21. From visual inspection, the PDF of Y appears to be slightly positively skewed.

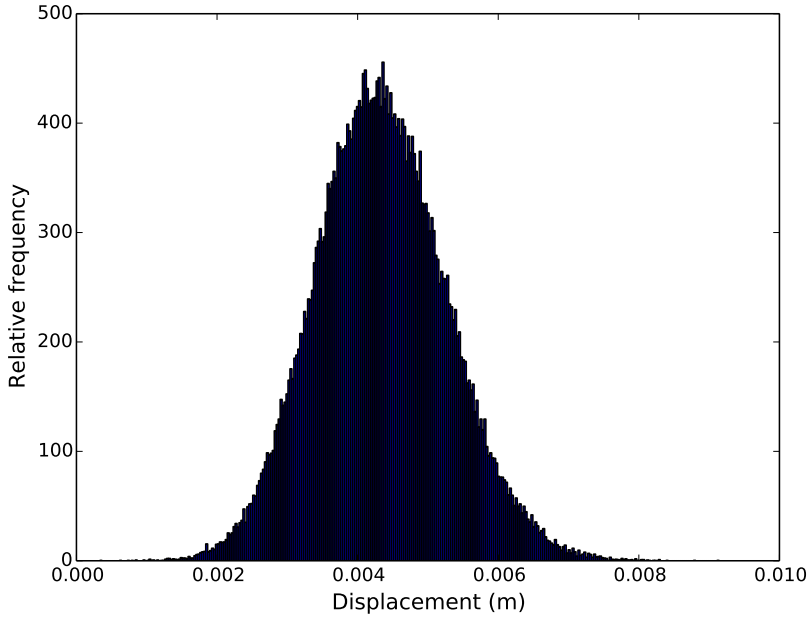


Figure 21: Histogram of the random displacement

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

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

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 19). 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 – scriptExample_Truss_PhysicalModel.py

```
from openturns import *
import numpy as np
#
#============================================================================
#
# Number of input random variables
dim=10
#
# Definition of the physical model
class myfunction(OpenTURNSPythonFunction):
    def __init__(self):
        OpenTURNSPythonFunction.__init__(self, dim, 1)
    def _exec(self, X):

        # Length of bar 3 (deterministic)
        L = 5.   # in m

        # Extract input variables from X
        E1,E2,E3,S1,S2,S3,P,Theta,alpha1,alpha2 = X

        # Degrees to radians
        theta_rad=Theta*np.pi/180.0
        alpha1_rad=alpha1*np.pi/180.0
        alpha2_rad=alpha2*np.pi/180.0

        # Vertical and horizontal loads
        Pv=-1.*P*np.cos(theta_rad)
        Ph=P*np.sin(theta_rad)

        # Intermediate variables
        alphav=(np.sin(alpha2_rad+alpha1_rad)/np.sin(alpha2_rad))*(1.0/((E3*S3/(
            np.cos(alpha1_rad)*E1*S1))+(E3*S3/(np.cos(alpha2_rad)*E2*S2))*(np.sin
            (alpha1_rad)/np.sin(alpha2_rad))**2+(np.sin(alpha2_rad+alpha1_rad)/np
            .sin(alpha2_rad))**2))

        alphah=(np.sin(alpha1_rad)/(np.sin(alpha2_rad)**2*np.cos(alpha2_rad)*E2*
            S2)+(np.sin(alpha1_rad+alpha2_rad)*np.cos(alpha2_rad))/(np.sin(
            alpha2_rad)**2*E3*S3))/(1.0/(np.cos(alpha1_rad)*E1*S1)+((np.sin(
            alpha1_rad)/np.sin(alpha2_rad))**2)*(1.0/(np.cos(alpha2_rad)*E2*S2))
            +((np.sin(alpha1_rad+alpha2_rad)/np.sin(alpha2_rad))**2)*(1.0/(E3*S3)
            ))

        # Horizontal and vertical displacements
        deltah=(alphah**2/(E1*S1*np.cos(alpha1_rad))+((alphah*np.sin(alpha1_rad)
            /np.sin(alpha2_rad)-1.0/np.sin(alpha2_rad))**2)/(E2*S2*np.cos(
            alpha2_rad))+((alphah*np.sin(alpha1_rad+alpha2_rad)/np.sin(alpha2_rad
            )-np.cos(alpha2_rad)/np.sin(alpha2_rad))**2)/(E3*S3))*Ph*L
```

```
37          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
            ,0.0015,0.0015,0.0015,2500.,45.,45.,45.]))
53      Ynom = truss_model(Xnom)
54      #
55      print "" ; print "Value of the displacement (m):", Ynom[0] ; print ""
```

2.3.2 Probabilistic model – scriptExample_Truss_ProbaModel.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
```

```
# Angle (bar1-bar3)
distalpha1 = Normal(45.,45.*0.03)      # Mean in degrees
# Angle (bar2-bar3)
distalpha2 = Normal(45.,45.*0.03)      # Mean in degrees
#
#============================================================
#                      Input random vector
#============================================================
#
myCollection = DistributionCollection(dim)
myCollection[0] = distE1
myCollection[1] = distE2
myCollection[2] = distE3
myCollection[3] = distS1
myCollection[4] = distS2
myCollection[5] = distS3
myCollection[6] = distP
myCollection[7] = distTheta
myCollection[8] = distalpha1
myCollection[9] = distalpha2
myDistribution = ComposedDistribution(myCollection)
vectX = RandomVector(myDistribution)
```

2.3.3 Uncertainty analysis based on PC expansions – scriptExample_Truss_mainSparsePolyChaos.py

```
from openturns import *
from scriptExample_elasticTruss_PhysicalModel  import *
from scriptExample_elasticTruss_ProbaModel import *
from numpy import empty, argmin, array, arange, floor, sqrt, linspace
from pylab import ion, figure, semilogy, xlabel, ylabel, bar, legend, xticks,
    yticks, hist
#
# Model function
#
truss_model = NumericalMathFunction(myfunction())
#
# Output random vector
#
vectY = RandomVector(truss_model, vectX)
#
# Basis of the polynomial chaos (PC) expansion (Hermite polynomials are selected
    )
#
polyColl = PolynomialFamilyCollection(dim)
for i in range(dim):
    polyColl[i] = HermiteFactory()
#
```

```
21 enumerateFunction = LinearEnumerateFunction(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(multivariateBasis,P)
50     #
51     polynomialChaosAlgorithm = FunctionalChaosAlgorithm (truss_model, \
52     Distribution(myDistribution) , truncatureBasisStrategy, evalStrategy)
53     polynomialChaosAlgorithm.run()
54     #
55     new_result = polynomialChaosAlgorithm.getResult()
56     Results_tuple = Results_tuple + (new_result,)
57     #
58     Relative_errors[p-1] = array(new_result.getRelativeErrors())[0]
59 #
60 # Plot the obtained relative errors
61 ion()
62 lw = 2.5
63 figure(1)
64 semilogy(p_list,Relative_errors,linewidth=lw)
65 xticks(list(arange(pmax)+1))
66 xlabel('PC degree') ; ylabel('Relative corrected leave-one-out error')
```

```
#
# Identify the most accurate PC expansion
p_optim = argmin(Relative_errors) + 1
The_Result = Results_tuple[p_optim-1]
#
print "" ; print "Optimal degree: ", p_optim
print "" ; print "Relative corrected LOO error:", Relative_errors[p_optim-1]
print "" ; print "Number of nonzero coefficients:", len(The_Result.getIndices())
#
# Post-processing of the optimal PC expansion
#
ChaosRV = FunctionalChaosRandomVector(The_Result)
#
Mean = ChaosRV.getMean()[0]
StD = sqrt(ChaosRV.getCovariance()[0,0])
print "" ; print "Response mean: ", Mean ; print ""
print "" ; print "Response standard deviation: ", StD ; print ""
#
SU = empty(dim) ; SUT = empty(dim)
for i in range(dim):
    SU[i] = ChaosRV.getSobolIndex(i)
    SUT[i] = ChaosRV.getSobolTotalIndex(i)
#
# Plot the sensitivity indices
w = 0.4
figure(2)
b1 = bar((arange(dim)+1)-w,SU,width=w,color='#000999')
b2 = bar((arange(dim)+1),SUT,width=w,color='#66FFFF')
legend((b1[0],b2[0]),('Sobol indices','Total Sobol indices'),'upper left')
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)
yticks(list( linspace(0.,1.,5) ))
#
#Plot histogram
truss_model_PC = The_Result.getMetaModel()
#
samplesize = 100000
sample_X = vectX.getSample(samplesize)
sample_YPC = truss_model_PC(sample_X)
asampleYPC = array(sample_YPC).flatten()
#
figure(3)
hist(asampleYPC,normed=True,bins=floor(sqrt(samplesize)))
xlabel("Displacement (m)", fontsize=14)
ylabel("Relative frequency", fontsize=14)
#
print "" ; print "Skewness:",  sample_YPC.computeSkewnessPerComponent()[0] ;
    print ""
```

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