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 :

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

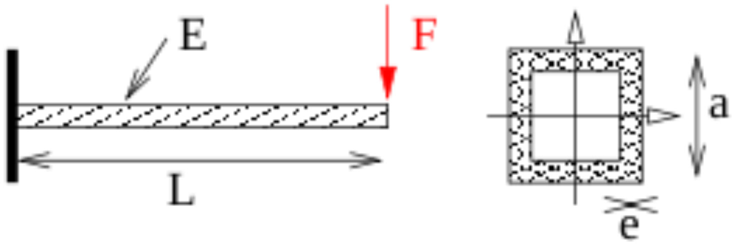


Figure 1: *cantilever beam under a ponctual bending load.*

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 :

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

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 $r = 0.93, t = 3.2, a = 2.8e7, b = 4.8e7$,
- 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 :

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

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 :

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

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 (9) to (11) 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)

# Visualize the probability density functions

pdfLoiE = distributionE.drawPDF()
# Change the legend
draw_E = pdfLoiE.getDrawable(0)
draw_E.setLegend("Beta(0.93, 3.2, 2.8e7, 4.8e7)")
pdfLoiE.setDrawable(draw_E,0)
# Change the title
pdfLoiE.setTitle("PDF of E")

pdfLoiE.draw("distributionE_pdf", 640, 480)
#View(pdfLoiE).show()

pdfLoiF = distributionF.drawPDF()
# Change the legend
draw_F = pdfLoiF.getDrawable(0)
draw_F.setLegend("LogNormal(30000, 9000, 15000)")
pdfLoiF.setDrawable(draw_F,0)
# Change the title
pdfLoiF.setTitle("PDF of F")

pdfLoiF.draw("distributionF_pdf", 640, 480)
#View(pdfLoiF).show()

pdfLoiL = distributionL.drawPDF()
# Change the legend
draw_L = pdfLoiL.getDrawable(0)
draw_L.setLegend("Uniform(250, 260)")
pdfLoiL.setDrawable(draw_L,0)
# Change the title
pdfLoiL.setTitle("PDF of L")

pdfLoiL.draw("distributionL_pdf", 640, 480)
#View(pdfLoiL).show()


pdfLoiI = distributionI.drawPDF()
# Change the legend
draw_I = pdfLoiI.getDrawable(0)
draw_I.setLegend("Beta(2.5, 4.0, 3.1e2, 4.5e2)")
pdfLoiI.setDrawable(draw_I,0)
# Change the title
pdfLoiI.setTitle("PDF of I")

pdfLoiI.draw("distributionI_pdf", 640, 480)
#View(pdfLoiI).show()

# Create the Spearman correlation matrix of the input random vector
RS = CorrelationMatrix(4)
RS[2,3] = -0.2

# Evaluate the correlation matrix of the Normal copula from RS
R = NormalCopula.GetCorrelationFromSpearmanCorrelation(RS)

# Create the Normal copula parametrized by R
copuleNormal = NormalCopula(R)

# Create a collection of the marginal distributions
collectionMarginals = [distributionE, distributionF, distributionL, distributionI]

# Create the input probability distribution of dimension 4
inputDistribution = ComposedDistribution(collectionMarginals, copuleNormal)

# Give a description of each component of the input distribution
inputDistribution.setDescription( ("E","F","L","I") )

# Create the input random vector
inputRandomVector = RandomVector(inputDistribution)
inputRandomVector.setDescription( ("E","F","L","I") )

# Create the output variable of interest
outputVariableOfInterest =  RandomVector(deviation, inputRandomVector)


#########################
### Min/Max approach Study
#########################


###################################################
# Min/Max study with deterministic design of experiment
###################################################

print "###################################################"
print " Min/Max study with deterministic design of experiments "
print "###################################################"


dim = deviation.getInputDimension()

# Create the structure of the design of experiments : Composite type

# On each direction separately, several levels are evaluated
# here,  3 levels : +/-0.5, +/-1., +/-3. from the center
levelsNumber = 3
levels = NumericalPoint(levelsNumber, 0.0)
levels[0] = 0.5
levels[1] = 1.0
levels[2] = 3.0
# Creation of the composite design of experiments
myDesign = Composite(dim, levels)

# Generation of points according to the structure of the design of experiments
# (in a reduced centered space)
inputSample = myDesign.generate()

# Scaling of the structure of the design of experiments
# scaling vector for each dimension of the levels of the structure
# to take into account the dimension of each component
# for example : the standard deviation of each component of 'inputRandomVector'
# in case of a RandomVector
scaling = NumericalPoint(dim)
scaling[0] = sqrt(inputRandomVector.getCovariance()[0,0])
scaling[1] = sqrt(inputRandomVector.getCovariance()[1,1])
scaling[2] = sqrt(inputRandomVector.getCovariance()[2,2])
scaling[3] = sqrt(inputRandomVector.getCovariance()[3,3])

inputSample *= scaling


# Translation of the nonReducedSample onto the center of the design of experiments
# center = mean point of the inputRandomVector distribution
center = inputRandomVector.getMean()
inputSample += center

# Get the number of points in the design of experiments
pointNumber = inputSample.getSize()

# Evaluate the ouput variable of interest on the design of experiments
outputSample = deviation(inputSample)


# Evaluate the range of the output variable of interest on that design of experiments
minValue = outputSample.getMin()
maxValue = outputSample.getMax()

print "From a composite  design of experiments of size = ", pointNumber
print "Levels = ", levels[0], ", ", levels[1], ", ", levels[2]
print "Min Value = ", minValue[0]
print "Max Value = ", maxValue[0]
print ""

############################################################
# Min/Max study by random sampling
############################################################

print "################################"
print " Min/Max study by random sampling"
print "################################"

pointNumber = 10000
print "From random sampling = ", pointNumber
outputSample2 = outputVariableOfInterest.getSample(pointNumber)

minValue2 = outputSample2.getMin()
maxValue2 = outputSample2.getMax()

print "Min Value = ", minValue2[0]
print "Max Value = ", maxValue2[0]
print ""





print ""
###############################################
### Random Study : central tendance of
### the output variable of interest
###############################################

print "#########################################"
print "Random Study : central tendance of"
print "the output variable of interest"
print "#########################################"
print ""

#####################################
# Taylor variance decomposition
#####################################

print "#############################"
print "Taylor variance decomposition"
print "#############################"
print ""

# We create a quadraticCumul algorithm
myQuadraticCumul = QuadraticCumul(outputVariableOfInterest)

# We compute the several elements provided by the quadratic cumul algorithm
# and evaluate the number of calculus needed
nbBefr = deviation.getEvaluationCallsNumber()

# Mean first order
meanFirstOrder = myQuadraticCumul.getMeanFirstOrder()[0]
nbAfter1 = deviation.getEvaluationCallsNumber()
```

```
# Mean second order
meanSecondOrder = myQuadraticCumul.getMeanSecondOrder()[0]
nbAfter2 = deviation.getEvaluationCallsNumber()

# Standard deviation
stdDeviation = sqrt(myQuadraticCumul.getCovariance()[0,0])
nbAfter3 = deviation.getEvaluationCallsNumber()

print "First order mean=", myQuadraticCumul.getMeanFirstOrder()[0]
print "Evaluation calls number = ", nbAfter1 - nbBefr
print "Second order mean=", myQuadraticCumul.getMeanSecondOrder()[0]
print "Evaluation calls number = ", nbAfter2 - nbAfter1
print "Standard deviation=", sqrt(myQuadraticCumul.getCovariance()[0,0])
print "Evaluation calls number = ", nbAfter3 - nbAfter2

print  "Importance factors="
for i in range(inputRandomVector.getDimension()) :
  print inputDistribution.getDescription()[i], " = ", myQuadraticCumul.getImportanceFactors()[i]


############################
# Random sampling
############################

print "######################"
print "Random sampling"
print "######################"

size1 = 10000
output_Sample1 = outputVariableOfInterest.getSample(size1)
outputMean = output_Sample1.computeMean()
outputCovariance = output_Sample1.computeCovariance()

print "Sample size = ", size1
print "Mean from sample = ", outputMean[0]
print "Standard deviation from sample = ", sqrt(outputCovariance[0,0])
print ""


#########################
# Kernel Smoothing Fitting
#########################


print "#########################"
print "# Kernel Smoothing Fitting"
print "#########################"

# We generate a sample of the output variable
size = 10000
output_sample = outputVariableOfInterest.getSample(size)

# We build the kernel smoothing distribution
kernel = KernelSmoothing()
bw = kernel.computeSilvermanBandwidth(output_sample)
smoothed = kernel.build(output_sample, bw)
print "Sample size = ", size
print  "Kernel bandwidth=" , kernel.getBandwidth()[0]

# We draw the pdf and cdf from kernel smoothing
# Evaluate at best the range of the graph
mean_sample = output_sample.computeMean()[0]
standardDeviation_sample = sqrt(output_sample.computeCovariance()[0,0])
xmin = mean_sample - 4*standardDeviation_sample
xmax = mean_sample + 4*standardDeviation_sample

# Draw the PDF
smoothedPDF = smoothed.drawPDF(xmin, xmax, 251)
# Change the title
smoothedPDF.setTitle("Kernel smoothing of the deviation - PDF")
# Change the legend
smoothedPDF_draw = smoothedPDF.getDrawable(0)
title = "PDF from Normal kernel (" + str(size) + " data)"
smoothedPDF_draw.setLegend(title)
smoothedPDF.setDrawable(smoothedPDF_draw,0)
smoothedPDF.draw("smoothedPDF", 640, 480)

# Draw the CDF
smoothedCDF = smoothed.drawCDF(xmin, xmax, 251)
# Change the title
smoothedCDF.setTitle("Kernel smoothing of the deviation - CDF")
# Change the legend
smoothedCDF_draw = smoothedCDF.getDrawable(0)
title = "CDF from Normal kernel (" + str(size) + " data)"
smoothedCDF_draw.setLegend(title)
smoothedCDF.setDrawable(smoothedCDF_draw,0)
# Change the legend position
smoothedCDF.setLegendPosition("bottomright")
smoothedCDF.draw("smoothedCDF", 640, 480)

# In order to see the graph whithout creating the associated files
#View(smoothedCDF).show()
#View(smoothedPDF).show()

# Mean of the output variable of interest
print "Mean from kernel smoothing = ", smoothed.getMean()[0]
print ""

# Superposition of the kernel smoothing pdf and the gaussian one
# which mean and standard deviation are those of the output_sample
normalDist = NormalFactory().build(output_sample)
normalDistPDF = normalDist.drawPDF(xmin, xmax, 251)
normalDistPDFDrawable = normalDistPDF.getDrawable(0)
normalDistPDFDrawable.setColor('blue')
```

```
smoothedPDF.add(normalDistPDFDrawable)
smoothedPDF.draw("smoothedPDF_and_NormalPDF", 640, 480)

# In order to see the graph whithout creating the associated files
#View(smoothedPDF).show()

#################################################################
### Probabilistic Study : threshold exceedance: deviation > 30cm
#################################################################

print ""
print "############################################################"
print "Probabilistic Study : threshold exceedance: deviation <-1cm"
print "############################################################"
print ""

######
# FORM
######

print "#####"
print "FORM"
print "#####"

# We create an Event from this RandomVector
# threshold has been defined in the kernel smoothing section
threshold = 30
myEvent = Event(outputVariableOfInterest, Greater(), threshold)
myEvent.setName("Deviation > 30 cm")

# We create a NearestPoint algorithm
myCobyla = Cobyla()
myCobyla.setMaximumIterationsNumber(1000)
myCobyla.setMaximumAbsoluteError(1.0e-10)
myCobyla.setMaximumRelativeError(1.0e-10)
myCobyla.setMaximumResidualError(1.0e-10)
myCobyla.setMaximumConstraintError(1.0e-10)

# We create a FORM algorithm
# The first parameter is a NearestPointAlgorithm
# The second parameter is an event
# The third parameter is a starting point for the design point research
meanVector = inputRandomVector.getMean()
myAlgoFORM = FORM(myCobyla, myEvent, meanVector)

# Get the number of times the limit state function has been evaluated so far
deviationCallNumberBeforeFORM = deviation.getEvaluationCallsNumber()

# Perform the simulation
myAlgoFORM.run()

# Get the number of times the limit state function has been evaluated so far
deviationCallNumberAfterFORM = deviation.getEvaluationCallsNumber()

# Stream out the result
resultFORM = myAlgoFORM.getResult()
print  "FORM event probability=" , resultFORM.getEventProbability()
print "Number of evaluations of the limit state function = ", deviationCallNumberAfterFORM - deviationCallNumberBeforeFORM
print  "Generalized reliability index=" , resultFORM.getGeneralisedReliabilityIndex()
print  "Standard space design point="
for i in range(inputRandomVector.getDimension()) :
  print inputDistribution.getDescription()[i], " = ", resultFORM.getStandardSpaceDesignPoint()[i]
print  "Physical space design point="
for i in range(inputRandomVector.getDimension()) :
  print inputDistribution.getDescription()[i], " = ", resultFORM.getPhysicalSpaceDesignPoint()[i]

print  "Importance factors="
for i in range(inputRandomVector.getDimension()) :
  print inputDistribution.getDescription()[i], " = ", resultFORM.getImportanceFactors()[i]

print  "Hasofer reliability index=" , resultFORM.getHasoferReliabilityIndex()
print ""

# Graph 1 : Importance Factors graph */
importanceFactorsGraph = resultFORM.drawImportanceFactors()
title = "FORM Importance factors - "+ myEvent.getName()
importanceFactorsGraph.setTitle( title)
importanceFactorsGraph.draw("ImportanceFactorsDrawingFORM", 640, 480)

# In order to see the graph whithout creating the associated files
#View(importanceFactorsGraph).show()


######
# MC
######

print "###########"
print "Monte Carlo"
print "###########"
print ""


maximumOuterSampling = 40000
blockSize = 100
coefficientOfVariation = 0.10

# We create a Monte Carlo algorithm
myAlgoMonteCarlo = MonteCarlo(myEvent)
myAlgoMonteCarlo.setMaximumOuterSampling(maximumOuterSampling)
myAlgoMonteCarlo.setBlockSize(blockSize)
myAlgoMonteCarlo.setMaximumCoefficientOfVariation(coefficientOfVariation)

# Define the HistoryStrategy to store the values of the probability estimator
# and the variance estimator
# used ot draw the convergence graph
# Full strategy
myAlgoMonteCarlo.setConvergenceStrategy(Full())

# Perform the simulation
myAlgoMonteCarlo.run()

# Display number of iterations and number of evaluations
# of the limit state function
print "Number of evaluations of the limit state function = ", myAlgoMonteCarlo.getResult().getOuterSampling()* myAlgoMonteCarlo.getResult().g

# Display the Monte Carlo probability of 'myEvent'
print "Monte Carlo probability estimation = ", myAlgoMonteCarlo.getResult().getProbabilityEstimate()

# Display the variance of the Monte Carlo probability estimator
print "Variance of the Monte Carlo probability estimator = ", myAlgoMonteCarlo.getResult().getVarianceEstimate()

# Display the confidence interval length centered around
# the MonteCarlo probability MCProb
# IC = [MCProb - 0.5*length, MCProb + 0.5*length]
# level 0.95

print "0.95 Confidence Interval = [", myAlgoMonteCarlo.getResult().getProbabilityEstimate() - 0.5*myAlgoMonteCarlo.getResult().getConfidenceL
print ""

# Draw the convergence graph and the confidence intervalle of level alpha
alpha = 0.90
convergenceGraphMonteCarlo = myAlgoMonteCarlo.drawProbabilityConvergence(alpha)
# In order to see the graph whithout creating the associated files
#View(convergenceGraphMonteCarlo).show()

# Create the file .EPS
convergenceGraphMonteCarlo.draw("convergenceGrapheMonteCarlo", 640, 480)
#View(convergenceGraphMonteCarlo).show()


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

print "#######################"
print "Directional Sampling"
print "#######################"
print " "

# Directional sampling from an event (slow and safe strategy by default)

# We create a Directional Sampling algorithm */
myAlgoDirectionalSim = DirectionalSampling(myEvent)
myAlgoDirectionalSim.setMaximumOuterSampling(maximumOuterSampling * blockSize)
myAlgoDirectionalSim.setBlockSize(1)
myAlgoDirectionalSim.setMaximumCoefficientOfVariation(coefficientOfVariation)

# Define the HistoryStrategy to store the values of the probability estimator
# and the variance estimator
# used ot draw the convergence graph
# Full strategy
myAlgoDirectionalSim.setConvergenceStrategy(Full())

# Save the number of calls to the limit state fucntion, its gradient and hessian already done
deviationCallNumberBefore = deviation.getEvaluationCallsNumber()
deviationGradientCallNumberBefore = deviation.getGradientCallsNumber()
deviationHessianCallNumberBefore = deviation.getHessianCallsNumber()

# Perform the simulation */
myAlgoDirectionalSim.run()

# Save the number of calls to the limit state fucntion, its gradient and hessian already done
deviationCallNumberAfter = deviation.getEvaluationCallsNumber()
deviationGradientCallNumberAfter = deviation.getGradientCallsNumber()
deviationHessianCallNumberAfter = deviation.getHessianCallsNumber()

# Display number of iterations and number of evaluations
# of the limit state function
print "Number of evaluations of the limit state function = ", deviationCallNumberAfter - deviationCallNumberBefore

# Display the Directional Simumation probability of 'myEvent'
print "Directional Sampling probability estimation = ", myAlgoDirectionalSim.getResult().getProbabilityEstimate()

# Display the variance of the Directional Simumation probability estimator
print "Variance of the Directional Sampling probability estimator = ", myAlgoDirectionalSim.getResult().getVarianceEstimate()

# Display the confidence interval length centered around
# the Directional Simumation probability DSProb
# IC = [DSProb - 0.5*length, DSProb + 0.5*length]
# level 0.95
print "0.95 Confidence Interval = [", myAlgoDirectionalSim.getResult().getProbabilityEstimate() - 0.5*myAlgoDirectionalSim.getResult().getCon
print ""


# Draw the convergence graph and the confidence intervalle of level alpha
alpha = 0.90
convergenceGraphDS = myAlgoDirectionalSim.drawProbabilityConvergence(alpha)
# In order to see the graph whithout creating the associated files
#View(convergenceGraphDS).show()

# Create the file .EPS
convergenceGraphDS.draw("convergenceGrapheDS", 640, 480)
#View(convergenceGraphDS).show()

####################
# Importance Sampling
####################
```

```
print "####################"
print "Importance Sampling"
print "####################"
print ""

maximumOuterSampling = 40000
blockSize = 1
standardSpaceDesignPoint = resultFORM.getStandardSpaceDesignPoint()
mean = standardSpaceDesignPoint
sigma = NumericalPoint(4, 1.0)
importanceDistribution = Normal(mean, sigma, CorrelationMatrix(4))

myStandardEvent = StandardEvent(myEvent)

myAlgoImportanceSampling = ImportanceSampling(myStandardEvent, importanceDistribution)
myAlgoImportanceSampling.setMaximumOuterSampling(maximumOuterSampling)
myAlgoImportanceSampling.setBlockSize(blockSize)
myAlgoImportanceSampling.setMaximumCoefficientOfVariation(coefficientOfVariation)

# Define the HistoryStrategy to store the values of the probability estimator
# and the variance estimator
# used ot draw the convergence graph
# Full strategy
myAlgoImportanceSampling.setConvergenceStrategy(Full())

# Perform the simulation
myAlgoImportanceSampling.run()

# Display number of iterations and number of evaluations
# of the limit state function
print "Number of evaluations of the limit state function = ", myAlgoImportanceSampling.getResult().getOuterSampling()* myAlgoImportanceSampli

# Display the Importance Sampling probability of 'myEvent'
print "Importance Sampling probability estimation = ", myAlgoImportanceSampling.getResult().getProbabilityEstimate()

# Display the variance of the Importance Sampling probability estimator
print "Variance of the Importance Sampling probability estimator = ", myAlgoImportanceSampling.getResult().getVarianceEstimate()

# Display the confidence interval length centered around
# the ImportanceSampling probability ISProb
# IC = [ISProb - 0.5*length, ISProb + 0.5*length]
# level 0.95
print "0.95 Confidence Interval = [", myAlgoImportanceSampling.getResult().getProbabilityEstimate() - 0.5*myAlgoImportanceSampling.getResult(

# Draw the convergence graph and the confidence intervalle of level alpha
alpha = 0.90
convergenceGraphIS = myAlgoImportanceSampling.drawProbabilityConvergence(alpha)
# In order to see the graph whithout creating the associated files
#View(convergenceGraphIS).show()

# Create the file .EPS
convergenceGraphIS.draw("convergenceGrapheIS", 640, 480)
#View(convergenceGraphIS).show()




###############################################
# Response surface : Polynomial expansion chaos
###############################################

print "##########################"
print "Polynomial expansion chaos"
print "##########################"
print " "

#############################################################
# STEP 1 : Construction of the multivariate orthonormal basis

# Dimension of the input random vector
dim = 4

# Create the univariate polynomial family collection
# which regroups the polynomial families for each direction
polyColl = PolynomialFamilyCollection(dim)

# Variable E
#Jacobi(alpha, beta) <=> Beta(\beta + 1, \alpha + \beta + 2, -1, 1)
alphaJ = 1.27
betaJ = -0.07
jacobiFamily = JacobiFactory(alphaJ, betaJ)
polyColl[0] = jacobiFamily


# Variable F
# Laguerre(k) <=> Gamma(k+1,1,0) (parametrage ppal)
kLaguerre = 1.78
laguerreFamily = LaguerreFactory(kLaguerre)
polyColl[1] = laguerreFamily

# Variable L
# Legendre <=> Unif(-1,1)
legendreFamily = LegendreFactory()
polyColl[2] = legendreFamily

# Variable E
# Jacobi(alpha, beta) <=> Beta(\beta + 1, \alpha + \beta + 2, -1, 1)
alphaJ2 = 0.5
betaJ2 = 1.5
jacobiFamily2 = JacobiFactory(alphaJ2, betaJ2)
polyColl[3] = jacobiFamily2
```

```
# Create the multivariate orthonormal basis
# which is the the cartesian product of the univariate basis
multivariateBasis = OrthogonalProductPolynomialFactory(polyColl, LinearEnumerateFunction(dim))

# Build a term of the basis as a NumericalMathFunction
# Generally, we do not need to construct manually any term,
# all terms are constructed automatically by a strategy of construction of the basis
i = 5
Psi_i = multivariateBasis.build(i)

# Get the measure mu associated to the multivariate basis
distributionMu = multivariateBasis.getMeasure()

###################################################################
# STEP 2 : Truncature strategy of the multivariate orthonormal basis

# CleaningStrategy :
# among the maximumConsideredTerms = 500 first polynoms,
# those which have the mostSignificant = 50 most significant contributions
# with significance criterion significanceFactor = 10^(-4)
# The True boolean indicates if we are interested
# in the online monitoring of the current basis update
# (removed or added coefficients)
maximumConsideredTerms = 500
mostSignificant = 50
significanceFactor = 1.0e-4
truncatureBasisStrategy = CleaningStrategy(multivariateBasis, maximumConsideredTerms, mostSignificant, significanceFactor, True)

###############################################################
# STEP 3 : Evaluation strategy of the approximation coefficients

# The technique proposed is the Least Squares technique
# where the points come from an design of experiments
# Here : the Monte Carlo sampling technique
sampleSize = 10000
evaluationCoeffStrategy = LeastSquaresStrategy(MonteCarloExperiment(sampleSize))

# STEP 4 : Creation of the Functional Chaos Algorithm

# FunctionalChaosAlgorithm :
# combination of the model : limitStateFunction
# the distribution of the input random vector : Xdistribution
# the truncature strategy of the multivariate basis
# and the evaluation strategy of the coefficients
polynomialChaosAlgorithm = FunctionalChaosAlgorithm(deviation, inputDistribution, truncatureBasisStrategy, ProjectionStrategy(evaluationCoeff

########################################################
# Run and results exploitation

# Perform the simulation
polynomialChaosAlgorithm.run()

# Stream out the result
polynomialChaosResult = polynomialChaosAlgorithm.getResult()

# Get the polynomial chaos coefficients
coefficients = polynomialChaosResult.getCoefficients()

# Get the meta model as a NumericalMathFunction
metaModel = polynomialChaosResult.getMetaModel()

# Get the indices of the selected polynomials : K
subsetK = polynomialChaosResult.getIndices()

# Get the composition of the polynomials
# of the truncated multivariate basis
for i in range(subsetK.getSize()) :
  print "Polynomial number ", i, " in truncated basis <-> polynomial number ", subsetK[i], " = ", LinearEnumerateFunction(dim)(subsetK[i]), "

# Get the multivariate basis
# as a colletion of NumericalMathFunction
multivariateBasisCollection = polynomialChaosResult.getReducedBasis()

# Get the distribution of variables Z
mu = polynomialChaosResult.getDistribution()
print "Distribution in the tansformed variables = ", mu
print ""

# Get the composed model which is the model of the reduced variables Z
composedModel = polynomialChaosResult.getComposedModel()

# Define the new random vector
newOutputVariableOfInterest = RandomVector(polynomialChaosResult)

# Get the mean and variance of the meta model

print "Mean =", newOutputVariableOfInterest.getMean()
print "Standard deviation =", sqrt(newOutputVariableOfInterest.getCovariance()[0,0])
print ""



########################################
# Graphs validation


# Graph 1 : cloud

# Generate a NumericalSample of the input random vector
# Evaluate the meta model and the real model
# draw the coulds (metamodel, real model)
# Verify points are on the first diagonal
sizeX = 500
```

```
Xsample = inputDistribution.getSample(sizeX)

modelSample = deviation(Xsample)
metaModelSample = metaModel(Xsample)

sampleMixed = NumericalSample(sizeX,2)
for i in range(sizeX) :
  sampleMixed[i, 0] = modelSample[i, 0]
  sampleMixed[i, 1] = metaModelSample[i, 0]

legend = str(sizeX) + " realizations"
comparisonCurve = Curve(modelSample, modelSample, "model")
comparisonCurve.setColor("red")

comparisonCloud = Cloud(sampleMixed, "blue", "fsquare", legend)
graphCloud = Graph("Polynomial chaos expansion", "model", "meta model", True, "topleft")
graphCloud.add(comparisonCurve)
graphCloud.add(comparisonCloud)

#View(graphCloud).show()
graphCloud.draw("PCE_comparisonModels")


# Graph 2 : polynoms family graphs

degreeMax = 5
pointNumber = 101
colorList = Drawable.GetValidColors()

# Jacobi for E
xMinJacobi = -1
xMaxJacobi = 1
titleJacobi = "Jacobi(" + str(alphaJ) + ", " + str(betaJ) + ") polynomials"
graphJacobi = Graph(titleJacobi, "z", "polynomial values", True, "topleft")
for i in range(degreeMax) :
  graphJacobi_temp = jacobiFamily.build(i).draw(xMinJacobi, xMaxJacobi, pointNumber)
  graphJacobi_temp_draw = graphJacobi_temp.getDrawable(0)
  legend = "degree " + str(i)
  graphJacobi_temp_draw.setLegend(legend)
  graphJacobi_temp_draw.setColor(colorList[i])
  graphJacobi.add(graphJacobi_temp_draw)
  #View(graphJacobi).show()
  graphJacobi.draw("PCE_JacobiPolynomials_VariableE")

# Laguerre for F
xMinLaguerre = 0
xMaxLaguerre = 10
titleLaguerre = "Laguerre(" + str(kLaguerre) +  ") polynomials"
graphLaguerre = Graph(titleLaguerre, "z", "polynomial values", True, "topleft")
for i in range(degreeMax) :
  graphLaguerre_temp = laguerreFamily.build(i).draw(xMinLaguerre, xMaxLaguerre, pointNumber)
  graphLaguerre_temp_draw = graphLaguerre_temp.getDrawable(0)
  legend = "degree " + str(i)
  graphLaguerre_temp_draw.setLegend(legend)
  graphLaguerre_temp_draw.setColor(colorList[i])
  graphLaguerre.add(graphLaguerre_temp_draw)
  #View(graphLaguerre).show()
  graphLaguerre.draw("PCE_LaguerrePolynomials_VariableF")

# Legendre for L
xMinLegendre = -1
xMaxLegendre = 1
titleLegendre = "Legendre polynomials"
graphLegendre = Graph(titleLegendre, "z", "polynomial values", True, "topright")
for i in range(degreeMax) :
  graphLegendre_temp = laguerreFamily.build(i).draw(xMinLegendre, xMaxLegendre, pointNumber)
  graphLegendre_temp_draw = graphLegendre_temp.getDrawable(0)
  legend = "degree " + str(i)
  graphLegendre_temp_draw.setLegend(legend)
  graphLegendre_temp_draw.setColor(colorList[i])
  graphLegendre.add(graphLegendre_temp_draw)
  #View(graphLegendre).show()
  graphLegendre.draw("PCE_LegendrePolynomials_VariableL")

# Jacobi for I
xMinJacobi2 = -1
xMaxJacobi2 = 1
titleJacobi2 = "Jacobi(" + str(alphaJ2) + ", " + str(betaJ2) + ") polynomials"
graphJacobi2 = Graph(titleJacobi2, "z", "polynomial values", True, "topright")
for i in range(degreeMax) :
  graphJacobi2_temp = jacobiFamily2.build(i).draw(xMinJacobi2, xMaxJacobi2, pointNumber)
  graphJacobi2_temp_draw = graphJacobi2_temp.getDrawable(0)
  legend = "degree " + str(i)
  graphJacobi2_temp_draw.setLegend(legend)
  graphJacobi2_temp_draw.setColor(colorList[i])
  graphJacobi2.add(graphJacobi2_temp_draw)
  #View(graphJacobi2).show()
  graphJacobi2.draw("PCE_JacobiPolynomials_VariableI")
```

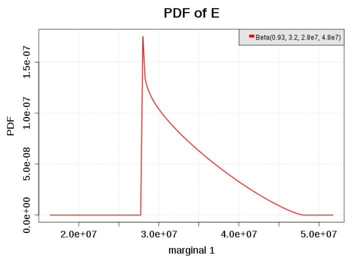
1.8 Output of the Python script

```
###################################################
 Min/Max study with deterministic design of experiments
###################################################
From a composite  design of experiments of size =  73
Levels =  0.5 ,  1.0 ,  3.0
Min Value =  0.649717975365
Max Value =  55.3605185131

#################################
 Min/Max study by random sampling
```

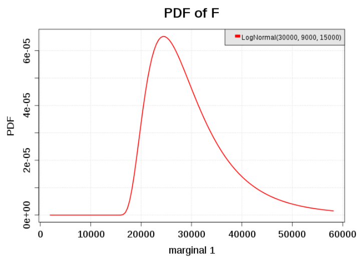
```
#################################
From random sampling =  10000
Min Value =  5.38682360418
Max Value =  52.7553886011


##########################################
Random Study : central tendance of
the output variable of interest
##########################################

##############################
Taylor variance decomposition
##############################

First order mean= 12.3369023123
Evaluation calls number =  1
Second order mean= 12.4198129769
Evaluation calls number =  33
Standard deviation= 4.18703072295
Evaluation calls number =  8
Importance factors=
E  =  0.149096880954
F  =  0.781344650859
L  =  0.0145457110929
I  =  0.0550127570943
######################
Random sampling
######################
Sample size =  10000
Mean from sample =  12.6090425333
Standard deviation from sample =  4.35926574946

#########################
# Kernel Smoothing Fitting
#########################
Sample size =  10000
Kernel bandwidth= 0.555537787263
Mean from kernel smoothing =  12.6194394108


###########################################################
Probabilistic Study : threshold exceedance: deviation <-1cm
###########################################################

#####
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 ]
#########################
Polynomial expansion chaos
#########################

Polynomial number  0  in truncated basis <-> polynomial number  0  =  [0,0,0,0]  in complete basis
Polynomial number  1  in truncated basis <-> polynomial number  1  =  [1,0,0,0]  in complete basis
Polynomial number  2  in truncated basis <-> polynomial number  2  =  [0,1,0,0]  in complete basis
Polynomial number  3  in truncated basis <-> polynomial number  3  =  [0,0,1,0]  in complete basis
Polynomial number  4  in truncated basis <-> polynomial number  4  =  [0,0,0,1]  in complete basis
Polynomial number  5  in truncated basis <-> polynomial number  5  =  [2,0,0,0]  in complete basis
Polynomial number  6  in truncated basis <-> polynomial number  6  =  [1,1,0,0]  in complete basis
Polynomial number  7  in truncated basis <-> polynomial number  7  =  [1,0,1,0]  in complete basis
Polynomial number  8  in truncated basis <-> polynomial number  8  =  [1,0,0,1]  in complete basis
Polynomial number  9  in truncated basis <-> polynomial number  9  =  [0,2,0,0]  in complete basis
Polynomial number  10  in truncated basis <-> polynomial number  10  =  [0,1,1,0]  in complete basis
Polynomial number  11  in truncated basis <-> polynomial number  11  =  [0,1,0,1]  in complete basis
Polynomial number  12  in truncated basis <-> polynomial number  12  =  [0,0,2,0]  in complete basis
Polynomial number  13  in truncated basis <-> polynomial number  13  =  [0,0,1,1]  in complete basis
Polynomial number  14  in truncated basis <-> polynomial number  14  =  [0,0,0,2]  in complete basis
Polynomial number  15  in truncated basis <-> polynomial number  15  =  [3,0,0,0]  in complete basis
Polynomial number  16  in truncated basis <-> polynomial number  16  =  [2,1,0,0]  in complete basis
Polynomial number  17  in truncated basis <-> polynomial number  17  =  [2,0,1,0]  in complete basis
Polynomial number  18  in truncated basis <-> polynomial number  18  =  [2,0,0,1]  in complete basis
Polynomial number  19  in truncated basis <-> polynomial number  19  =  [1,2,0,0]  in complete basis
Polynomial number  20  in truncated basis <-> polynomial number  20  =  [1,1,1,0]  in complete basis
Polynomial number  21  in truncated basis <-> polynomial number  21  =  [1,1,0,1]  in complete basis
Polynomial number  22  in truncated basis <-> polynomial number  23  =  [1,0,1,1]  in complete basis
Polynomial number  23  in truncated basis <-> polynomial number  24  =  [1,0,0,2]  in complete basis
Polynomial number  24  in truncated basis <-> polynomial number  25  =  [0,3,0,0]  in complete basis
Polynomial number  25  in truncated basis <-> polynomial number  26  =  [0,2,1,0]  in complete basis
Polynomial number  26  in truncated basis <-> polynomial number  27  =  [0,2,0,1]  in complete basis
Polynomial number  27  in truncated basis <-> polynomial number  29  =  [0,1,1,1]  in complete basis
Polynomial number  28  in truncated basis <-> polynomial number  30  =  [0,1,0,2]  in complete basis
Polynomial number  29  in truncated basis <-> polynomial number  31  =  [0,0,3,0]  in complete basis
Polynomial number  30  in truncated basis <-> polynomial number  33  =  [0,0,1,2]  in complete basis
Polynomial number  31  in truncated basis <-> polynomial number  34  =  [0,0,0,3]  in complete basis
Polynomial number  32  in truncated basis <-> polynomial number  36  =  [3,1,0,0]  in complete basis
Polynomial number  33  in truncated basis <-> polynomial number  39  =  [2,2,0,0]  in complete basis
Polynomial number  34  in truncated basis <-> polynomial number  45  =  [1,3,0,0]  in complete basis
Polynomial number  35  in truncated basis <-> polynomial number  55  =  [0,4,0,0]  in complete basis
Polynomial number  36  in truncated basis <-> polynomial number  56  =  [0,3,1,0]  in complete basis
Polynomial number  37  in truncated basis <-> polynomial number  57  =  [0,3,0,1]  in complete basis
Polynomial number  38  in truncated basis <-> polynomial number  61  =  [0,1,3,0]  in complete basis
Polynomial number  39  in truncated basis <-> polynomial number  63  =  [0,1,1,2]  in complete basis
Polynomial number  40  in truncated basis <-> polynomial number  66  =  [0,0,3,1]  in complete basis
Polynomial number  41  in truncated basis <-> polynomial number  105  =  [0,5,0,0]  in complete basis
Polynomial number  42  in truncated basis <-> polynomial number  106  =  [0,4,1,0]  in complete basis
Polynomial number  43  in truncated basis <-> polynomial number  120  =  [0,0,5,0]  in complete basis
Polynomial number  44  in truncated basis <-> polynomial number  182  =  [0,6,0,0]  in complete basis
Polynomial number  45  in truncated basis <-> polynomial number  183  =  [0,5,1,0]  in complete basis
Polynomial number  46  in truncated basis <-> polynomial number  294  =  [0,7,0,0]  in complete basis
Polynomial number  47  in truncated basis <-> polynomial number  295  =  [0,6,1,0]  in complete basis
Polynomial number  48  in truncated basis <-> polynomial number  322  =  [0,0,7,0]  in complete basis
Polynomial number  49  in truncated basis <-> polynomial number  450  =  [0,8,0,0]  in complete basis
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
 [   0         1         0         0        ]
 [   0         0         1        -0.209057 ]
 [   0         0        -0.209057  1        ]]))

Mean = [12.626]
Standard deviation = 4.3612562743
```

1.9 Figures

The probability density function (PDF) of each marginal is given in Figures 2 to 3.

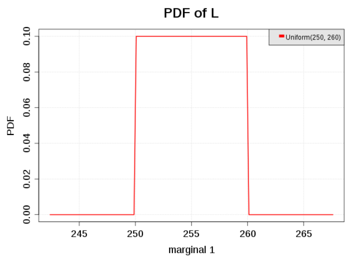


Probability density function of the parameter E

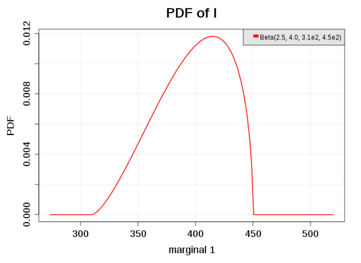


Probability density function of the parameter F

Figure 2



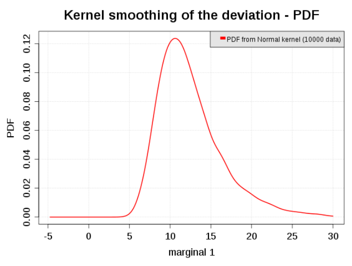
PDFof the parameter L



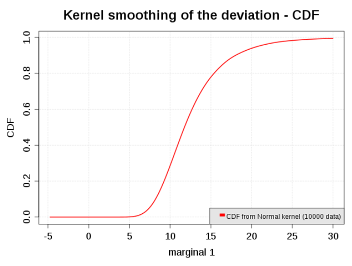
PDF of the parameter I

Figure 3

The probability density function (PDF) and the cumulative density function (CDF) of the deviation fiited with the kernel smoothing metid are drawn in Figures 4 and 4.



PDF of the deviation with the kernel smoothing method.



CDF of the deviation with the kernel smoothing method.

Figure 4

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 5.

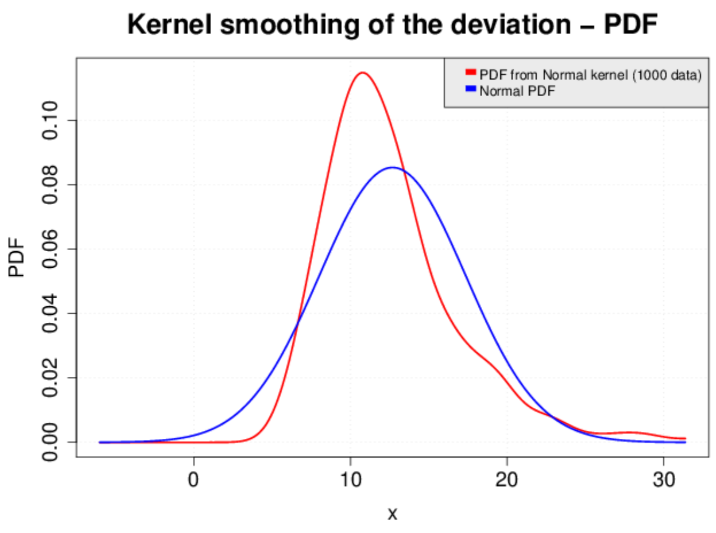


Figure 5: *Superposition of the kernel smoothed density function and the normal fitted from the same sample.*

The importance factors from the FORM method are given in Figure 6.

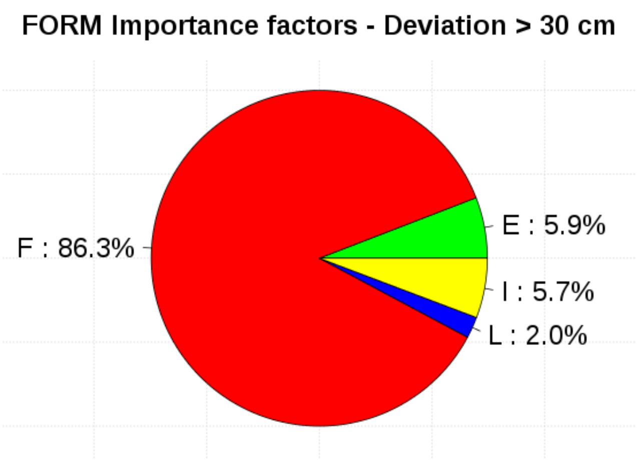
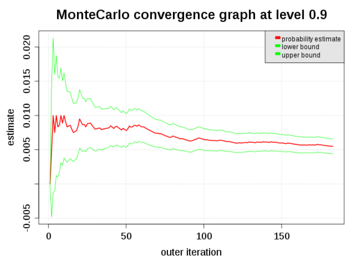
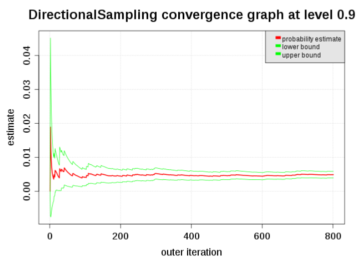


Figure 6: *FORM importance factors of the event : deviation > 30 cm.*

The convergence graphs of the simulation methods are given in Figures 7 to 8.



Monte Carlo convergence graph.



Directional Sampling convergence graph.

Figure 7

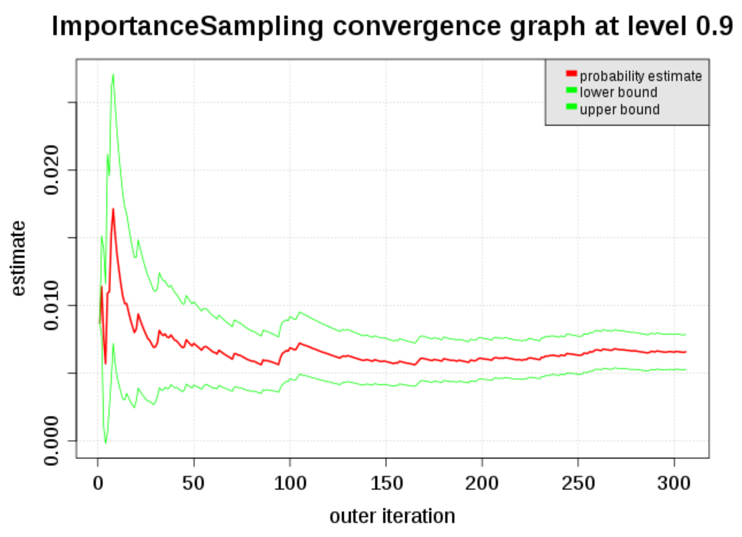
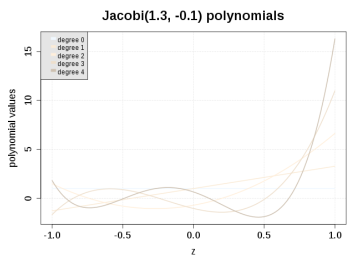


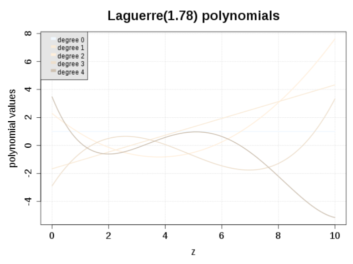
Figure 8: *Importance sampling convergence graph.*

Figures (9) to (11) 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.

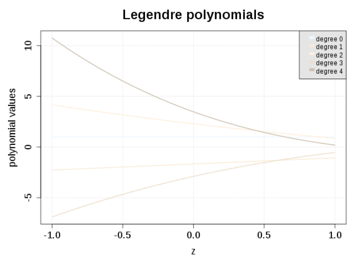


The 5-th first polynomials of the Jacobi family associated to the variable E.

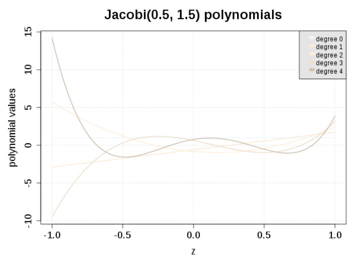


The 5-th first polynomials of the Laguerre family associated to the variable F.

Figure 9



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



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

Figure 10

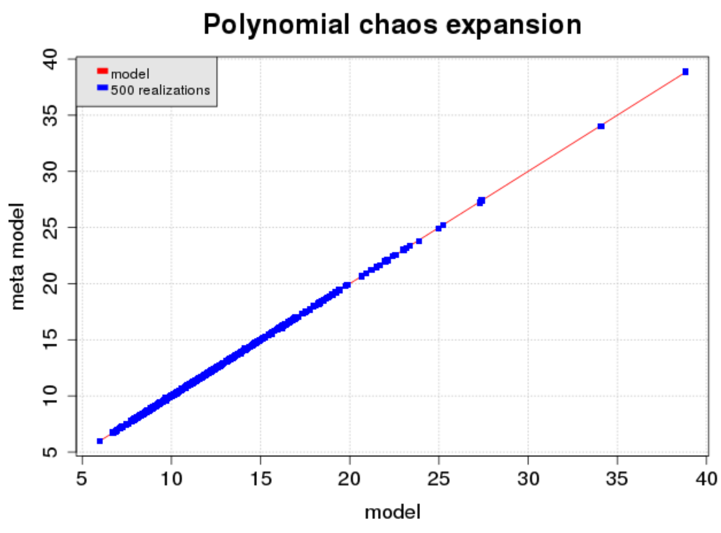


Figure 11: *Comparison of values from the model and the polynomial chaos meta model.*

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.

Table of contents   Example 2: elastic truss structure