FE based structural reliability analysis using STAND environment

R. Stocki, K. Kolanek, J. Knabel, P. Tauzowski

Institute of Fundamental Technological Research, Polish Academy of Sciences, ul. Swietokrzyska 21 00-049 Warsaw, Poland

Abstract

An assessment of structural reliability requires multiple evaluations of the limit state function for various realizations of random parameters of the structural system. In the majority of industrial applications the limit state functions cannot be expressed explicitly in terms of the random parameters but they are specified using selected outcomes of the FE analysis. In consequence, in order to be useful in practice, a structural reliability analysis program should be closely integrated with a FE module or it should be interfaced with an advanced external FE program. When the FE source code is not available, which is usually the case, the only option is to establish a communication between the reliability analysis program and an external FE software through the batch mechanism of data modification, job submission and results extraction.

The main subject of this article is to present the reliability analysis capabilities of STAND software, which is being developed in the Institute of Fundamental Technological Research of Polish Academy of Sciences. A special emphasis is put on the issues related to its interfacing with external general purpose FE codes. It is shown that when shape type random variables are used, leading to modifications of the FE mesh, or when the limit state function contains numerical noise, standard algorithms for localizing the design point often fail to converge and a special method based on some response surface approximation is needed. A proposition of such a strategy that employs an adaptive response surface approximation of the limit state function is presented in this article.

Development of a reliability analysis program is a challenging project and calls for such a code organization, which would facilitate a simultaneous work of many programmers and allow for easy maintenance and modifications. The so-called object-oriented programming seems to provide a convenient framework to realize these objectives. The object-oriented approach is used in STAND development. The advantages of this programming paradigm and a short description of the STAND's class hierarchy are presented in the text.

The study is concluded with two numerical examples of interfacing STAND with state of the art commercial FE programs.

1 Introduction

An essential part of the analysis and design of engineering structures is dealing with uncertainties of structural parameters and loading. It is, therefore, very important to be able to examine how the uncertainties "propagate" in the structural system resulting in a scatter of its responses. In the design practice the random character of material parameters and loads is taken into account by means of safety factors. In most of the common cases this approach allows engineers for designing safe structures. On the other hand, it usually prohibits an evaluation of the structural safety level.

A more realistic way of uncertainties treatment is provided by reliability analysis, which employs stochastic modeling of design parameters to estimate probabilities of violating respective limit states. In the last thirty years the reliability analysis has been a subject of an intensive development leading to its evolution from an academic discipline to modern methodology supporting limit states analysis of structural systems. Recently, also thanks to a development of dedicated computer programs, the reliability analysis has become a tool, which is available for a wide group of engineers and researchers. Popular reliability analysis codes like ANSYS PDS and DesignXplorer, CalREL/FERUM/OpenSees, COSSAN, NESSUS, PERMAS-RA/STRUREL, PHIMECA-SOFT, PROBAN, PROFES, UNIPASS were reviewed in [17].

Irrespective of the method employed, an assessment of the structural reliability requires multiple evaluations of the limit state function (LSF) for various realizations of random parameters of the structural system. In the majority of industrial applications the LSFs cannot be expressed explicitly in terms of the random variables but they are given using selected outcomes of the FE analysis of the considered structural problem. Therefore, in order to be useful in practice, a structural reliability analysis program should be closely integrated with a FE module or it should be interfaced with an advanced external FE program.

The first approach, which requires an access to the source code of the programs, is for obvious reasons available only for developers of the FE software. Good examples of such integration are ANSYS Probabilistic Design System and ANSYS DesignXplorer [18] or PERMAS-RA [8]. When the FE source code is not available, which is usually the case, the only option is to establish a communication between the reliability analysis program and the external FE software through the batch mechanism of data modification, job submission and results extraction. Depending on the type of random variables and a way of introducing their updated values into input data files of the numerical model this approach may be quite demanding to implement. On the other hand, it allows for an application of the reliability analysis in a variety of problems supported by commercial FE programs.

Many of the reliability analysis programs that are available on market communicate with third party software via a user provided function that needs to be compiled and linked with the reliability analysis code. This method, however, is considered cumbersome because programming of the interfacing function leads to a substantial additional effort from the user. The requirement of programming skills may also be considered as a drawback of such a solution. Therefore, providing user friendly interfacing with external software is currently a crucial objective for reliability analysis software developers. Here, one should mention the NESSUS package [24], where the mapping of random variables to the FE code is greatly facilitated by the graphical user interface (GUI).

Applicability of individual reliability analysis methods very much depends on the problem considered and the required accuracy. In order to be robust an advanced reliability analysis software should offer a variety of algorithms that can deal with a large class of problems. In the case of complex structures, for which a single FE analysis takes hours or even days of the CPU time, estimation of the probability of failure requires creation of an analytical metamodel to substitute computationally expensive LSFs. In consequence, an essential part of the reliability analysis software turns out to be the response surface module. In addition to modules responsible for the failure probability computation, development of a comfortable integration/interfacing with external FE codes and a user friendly GUI constitute a substantial programming task. All these make developing the reliability analysis program a challenging project and calls for such a code organization, which would facilitate a simultaneous work of many programmers and allow for easy maintenance and modifications. The so-called object-oriented programming seems to provide a convenient framework to realize these objectives. However, despite its obvious advantages and being a standard programming paradigm for most of the present day software, still the object-oriented approach is not common in developing reliability analysis programs. Only few of the newer codes (ANSYS/DesignXplorer [18], PHIMECA-SOFT [11], Opensees [3]) are designed according to this methodology.

The object-oriented approach is used in the reliability analysis program STAND (STochastic ANalysis and Design) developed in the Institute of Fundamental Technological Research (IFTR) of Polish Academy of Sciences. The program was initially meant as a research tool and a test platform for various structural reliability analysis methods investigated in IFTR. STAND, in fact, is a set of C++ libraries grouping classes that support data structures and algorithms utilized in the reliability analysis. Together with standard methods, STAND contains also several original reliability analysis algorithms developed for particular applications, e.g. the crashworthiness reliability, see [22]. For the last two years STAND has "matured" so far that currently an important part of its development concerns GUI functionality and an automatic interfacing with external computational software.

The main subject of this article is a presentation of STAND capabilities for the reliability analysis. A special emphasis is put on the issues related to interfacing with external general purpose FE codes. It is shown that when shape type random variables are used, leading to modifications of the mesh of finite elements, standard algorithms for localizing the most probable failure point may have problems to converge and a special method based on some response surface approximation of LSF is needed. A detailed description of such an algorithm together with a short presentation of other reliability analysis methods implemented in STAND is given in Sec. 2. In Sec. 3 the object-oriented architecture of STAND is explained and in Sec. 4.1 an example of the failure probability computation by interfacing STAND with the state of the art FE software ABAQUS is presented.

From the point of view of algorithmic implementation, reliability analysis methods are particularly suited for parallelization, as most of the methods are in fact intrinsically parallel. This means that the solution of the various subtasks can be performed independently without a need of communication or synchronization between the computational nodes. All the sampling type techniques are examples of the methods that are particularly easy to parallelize. An efficient reliability analysis program should, therefore, facilitate parallel processing. In Sec. 4.2 a case of multiprocessor reliability computation performed by STAND is illustrated by the crashworthiness reliability analysis that employs the FE program RADIOSS [14] for crash simulation.

2 Structural reliability analysis with STAND

In the reliability analysis of structural systems the uncertain parameters such us material constants, geometrical dimensions or load magnitudes are represented by random variables $X_1, X_2, \dots, X_n$. The vector $\mathbf{X} = \{X_1, X_2, \dots, X_n\}$ is usually called the vector of basic random variables with joint probability density function (PDF) $f_{\mathbf{X}}(\mathbf{x})$, where $\mathbf{x}$ denotes a realization of $\mathbf{X}$.

Depending on a sample of the basic variables, the structural system will satisfy the required work conditions and be safe and intact or not. The criterion of structural failure is usually expressed by the equality $g(\mathbf{x}) = 0$ that defines a hypersurface, called the limit state surface, in the space of realization of the vector $\mathbf{X}$. The limit state surface divides this space into two domains: the failure domain $\Omega_f = \{\mathbf{x} : g(\mathbf{x}) \leq 0\}$ and the safe domain $\Omega_s = \{\mathbf{x} : g(\mathbf{x}) > 0\}$. Hence, the failure probability of the structural system is determined by the following integral:

$$P_f = \mathbb{P}[g(\mathbf{X}) \leq 0] = \int_{\Omega_f} f_{\mathbf{X}}(\mathbf{x})\,\mathrm{d}\mathbf{x}. \tag{1}$$

where $\mathbb{P}[A]$ means the probability of the random event A. Such defined probability of failure constitutes the basic reliability measure. There exist other more complex formulations of structural reliability, in which multiple LSFs or time dependence of random variables are taken into account, however, in the current paper only the time independent, component reliability analysis problems are considered.

An estimation of the integral in (1) is one of the main tasks in failure probability computation. The numerical integration seems to be efficient only for integrals defined in up to four dimensional spaces. Unfortunately, the number of random variables for most of the realistic stochastic models of structural systems is substantially greater. Therefore, the P_f value is usually estimated by stochastic integration methods (Monte Carlo sampling) or by some approximation techniques, like FORM or SORM, see e.g. [4, 16].

A schematic data flow and the most important stages of the reliability analysis performed using STAND are shown in Fig. 1. Since all the major building blocks of STAND will be presented in more detail in next sections, here only a short description is given. Obviously, the first step is a definition of the stochastic model. The user must provide marginal distributions of the basic random variables and, for correlated variables, their correlation matrix. At this step, definitions of the so-called external variables, which are outputs of third party programs, can also be introduced. In order to compute values of external variables that correspond to realizations of basic random variables an "instruction" must be provided on how the input data files for the third party programs should be updated with the current values of variables. It is carried out through the mechanism of tagging the variables described later in the text. After defining the variables LSF is defined. Using the graphical interface of STAND it is simply typed in a window in standard math-notation as an expression involving both basic and external variables. The consecutive steps are dedicated to selection, parameter setting and execution of the reliability analysis algorithm. Most of the basic reliability analysis methods are supported by STAND. The object oriented design of the code allows for rapid implementation of new methods. Some of them are presented in Sec. 2.3. The analysis terminates with presentation of the results.

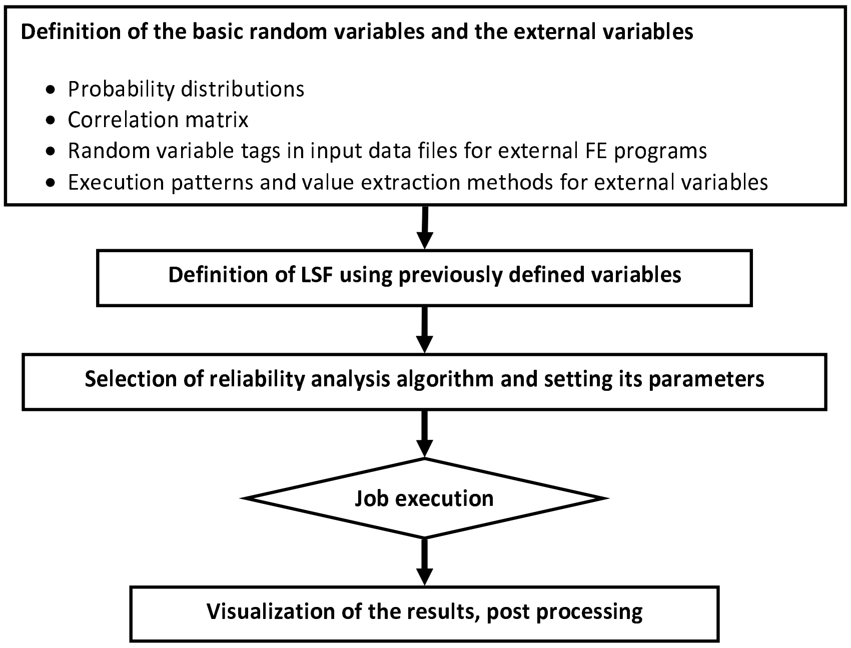

Definition of the basic random variables and the external variables
• Probability distributions
• Correlation matrix
• Random variable tags in input data files for external FE programs
• Execution patterns and value extraction methods for external variables
Definition of LSF using previously defined variables
Selection of reliability analysis algorithm and setting its parameters
Job execution
Visualization of the results, post processing

Fig. 1: Major steps of reliability analysis task using STAND

2.1 Description of the probabilistic model

Two types of variables are clearly distinguished in STAND. These are the basic random variables and the external variables. The external variables are, in fact, implicit random functions of the basic variables. Their corresponding values are found as outputs of third party programs for certain realizations of the basic random variables. Due to their particular nature both types of variables need to be treated individually in the code and require different information in order to be defined.

2.1.1 Basic random variables

Unfortunately, very often in case of real life reliability analysis problems the actual joint PDF of the basic random variables is difficult or even impossible to obtain. As a compromise solution, in STAND an approximation of $f_{\mathbf{X}}(\mathbf{x})$ is build by means of the so-called Nataf model [13] defined by marginal PDFs of the random variables and the matrix of correlation coefficients. For structural reliability analysis applications it is particularly important that the Nataf model can be efficiently transformed, $\mathbf{U} = \mathbf{T}(\mathbf{X})$, to the so-called standard normal space $\boldsymbol{\mathcal{U}}$. Most of the reliability analysis algorithms implemented in STAND take advantage of the $\boldsymbol{\mathcal{U}}$ space properties, where the probability measure is defined by the probability density function $f_{\mathbf{U}}(\mathbf{u}) = \prod_{i=1}^{n} \varphi(u_i)$ being the product of n one-dimensional standard normal PDFs of random variables $U_i = T_i(\mathbf{X})$.

The transformation $\mathbf{U} = \mathbf{T}(\mathbf{X})$ consists of two steps. First, the marginal distributions of basic random variables are transformed to the standard normal distribution according to the formula:

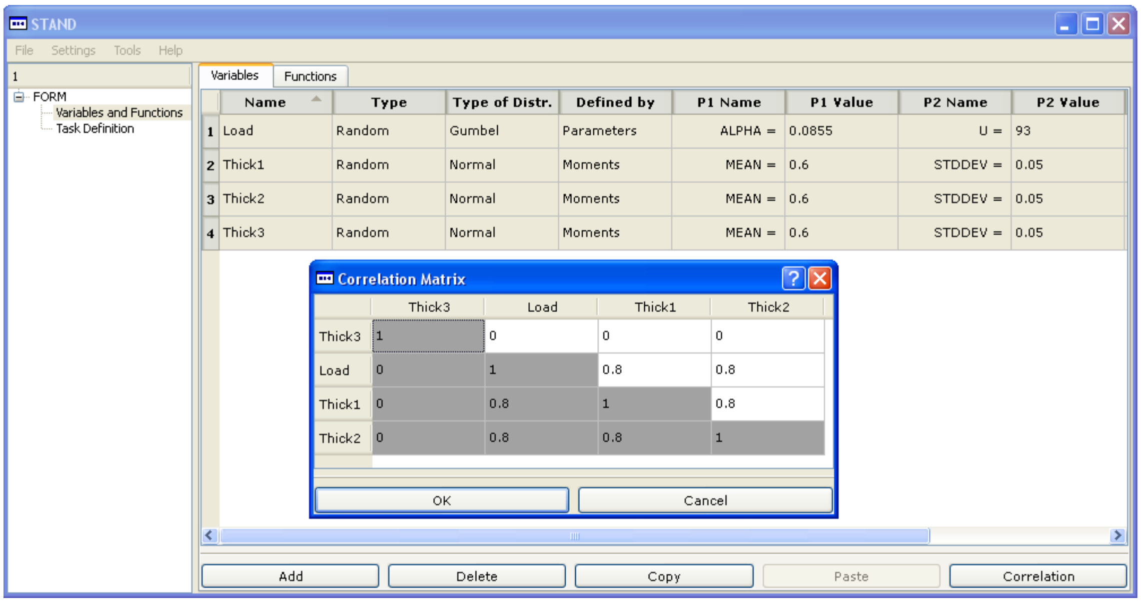
$$y_i = \Phi^{-1}[F_{X_i}(x_i)] \qquad i = 1, \ldots, n, \tag{2}$$

where $\Phi^{-1}(\cdot)$ is the inverse of the standard normal cumulative distribution function (CDF) and F_{X_i} is the CDF of respective basic random variable. The standard normal variables Y_i, $i = 1, \ldots, n$ are correlated with the correlation matrix $\mathbf{R}$ defined by a proper integral equation, see [13]. Then, the variables $\mathbf{Y}$ are transformed to the independent standard normal variables $\mathbf{U}$ using the formula

$$\mathbf{U} = \mathbf{L}^{-1}\mathbf{Y} \tag{3}$$

where $\mathbf{L}$ is a lower triangular matrix obtained from Cholesky decomposition of the correlation matrix $\mathbf{R}$. The approximative character of the described transformation results from the assumption that variables $\mathbf{Y}$ are also jointly normal (which is, of course, not guaranteed by the Gaussianity of marginal distribution of Y_i variables). On the other hand, when the basic random variables $\mathbf{X}$ are normal the Nataf model is exact.

In the current version of the program the following marginal distributions are available: exponential, Frechet, Gumbel, log-normal, normal, Rayleigh, uniform, Weibull and a general empirical distribution described by a set of experimental points. The distributions can be specified by the probabilistic moments (the mean value and the standard deviation) or by distribution parameters. By default random variables are assumed independent, however, their correlation coefficients can be easily introduced by filling appropriate cells of the correlation matrix in the dedicated GUI window, see Fig. 2.

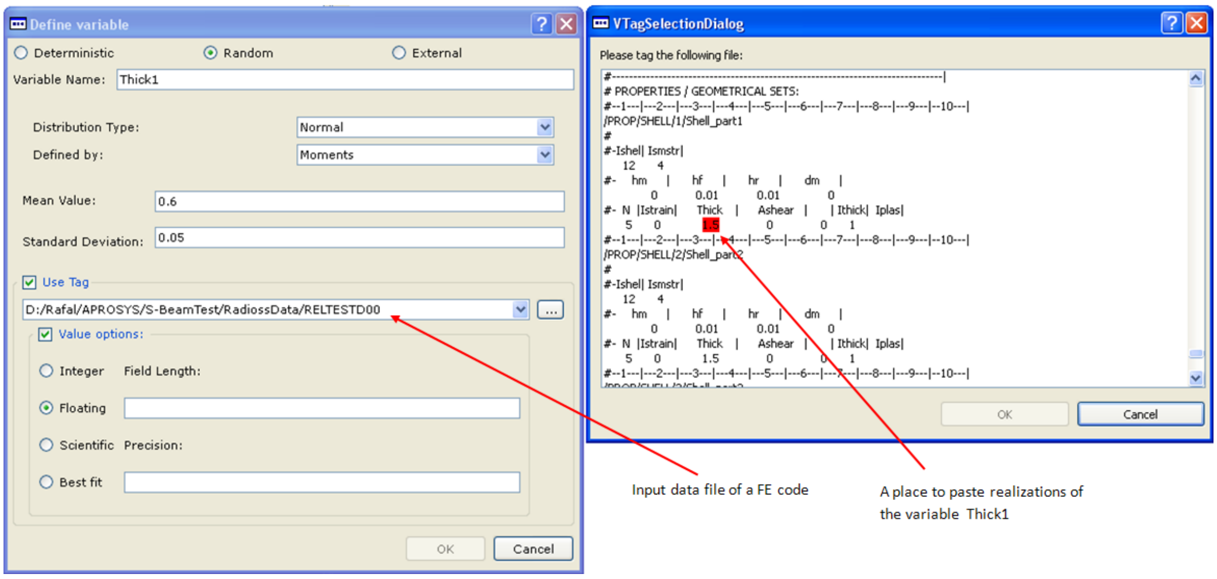


	Name	Type	Type of Distr.	Defined by	P1 Name	P1 Value	P2 Name	P2 Value
1	Load	Random	Gumbel	Parameters	ALPHA =	0.0855	U =	93
2	Thick1	Random	Normal	Moments	MEAN =	0.6	STDDEV =	0.05
3	Thick2	Random	Normal	Moments	MEAN =	0.6	STDDEV =	0.05
4	Thick3	Random	Normal	Moments	MEAN =	0.6	STDDEV =	0.05

	Thick3	Load	Thick1	Thick2
Thick3	1	0	0	0
Load	0	1	0.8	0.8
Thick1	0	0.8	1	0.8
Thick2	0	0.8	0.8	1

Fig. 2: GUI for introducing random variables description

As it was mentioned, an estimation of the structural failure probability requires multiple LSF evaluations for various realizations of the basic random variables. Usually LSF is computed by third party programs (in most of the cases these are FE codes). In order to facilitate and automate the task of communicating with external programs a simple interfacing mechanism has been implemented in STAND. It is possible to interface STAND with programs, which can be called in the batch mode and support text files for reading input data and writing results. STAND has capability to execute these programs for input files containing realizations of the basic random variables "transmitted" by reliability analysis algorithms. An appropriate GUI window allows users to specify tags in the input files, defining places where to insert the values of random variables, see Fig. 3.



Variable Name: Thick1
Distribution Type: Normal
Defined by: Moments
Mean Value: 0.6
Standard Deviation: 0.05
D:/Rafal/APROSYS/S-BeamTest/RadiossData/RELTESTD00
Field Length:
Precision:

Please tag the following file:

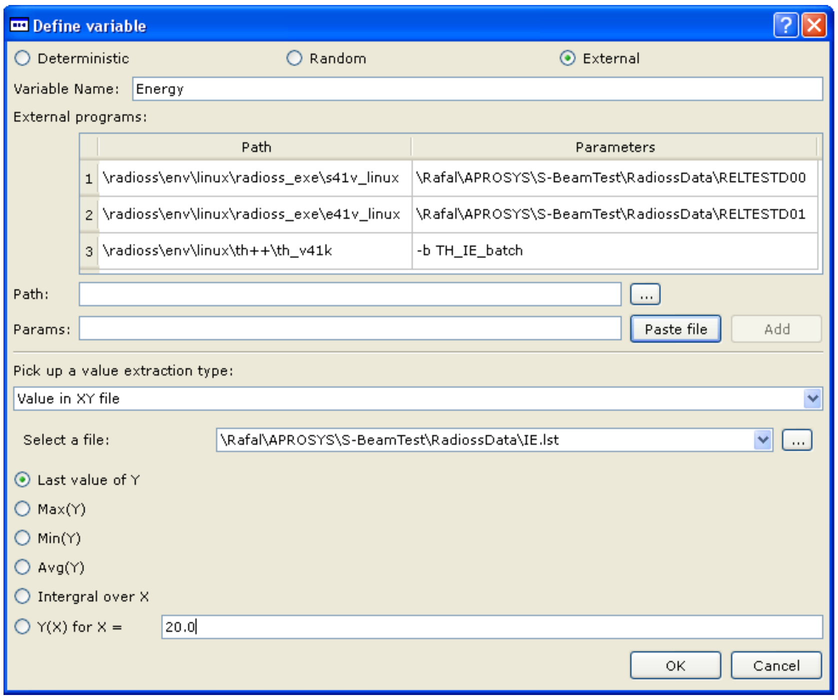
```
#-------------------------------------------------------------------------|
# PROPERTIES / GEOMETRICAL SETS:
#--1---|---2---|---3---|---4---|---5---|---6---|---7---|---8---|---9---|--10---|
/PROP/SHELL/1/Shell_part1
#
#-Ishel| Ismstr|
    12      4
#-    hm     |       hf     |       hr     |       dm     |
           0          0.01          0.01            0
#-  N  |Istrain|     Thick    |     Ashear    |       | Ithick|  Iplas|
    5      0          1.5            0              0      1
#--1---|---2---|---3---|---4---|---5---|---6---|---7---|---8---|---9---|--10---|
/PROP/SHELL/2/Shell_part2
#
#-Ishel| Ismstr|
    12      4
#-    hm     |       hf     |       hr     |       dm     |
           0          0.01          0.01            0
#-  N  |Istrain|     Thick    |     Ashear    |       | Ithick|  Iplas|
    5      0          1.5            0              0      1
#--1---|---2---|---3---|---4---|---5---|---6---|---7---|---8---|---9---|--10---|
```

Fig. 3: Associating the random variables with particular locations in the input data files of FE programs

2.1.2 External variables

The second group of variables considered in STAND are the external variables. Their values are extracted from the output files of third party programs. In order to define the external variable a sequence of system commands calling external programs must be given, see Fig. 4. The commands are provided with arguments, which can include input data files specified in the basic random variables description. It is assumed that the value of external variable is written to a text file, which is obtained as a result of executing the specified sequence of commands. The user should also choose a method of extracting the external variable value from the results file. Depending on the case multiple strategies can be selected. The value can be identified either by its absolute position in the file or relative to a text label or as a scalar value corresponding to a resulting time history curve.

Very often the same programs are used to produce different external variables. For example, as a result of running a FE code we can get displacements, strains, stresses or temperature values that are subsequently used in the LSF definition. To improve computational efficiency, when performing the reliability analysis the external variables with the same calling sequence are identified and grouped in order to avoid repeated executions of the external programs for the same data. This strategy is fully automated and no user intervention is required here.


Define variable

Deterministic [✓ External] Random

Variable Name: Energy

External programs:

	Path	Parameters
1	\radioss\env\linux\radioss_exe\s41v_linux	\Rafal\APROSYS\S-BeamTest\RadiossData\RELTESTD00
2	\radioss\env\linux\radioss_exe\e41v_linux	\Rafal\APROSYS\S-BeamTest\RadiossData\RELTESTD01
3	\radioss\env\linux\th++\th_v41k	-b TH_IE_batch

Path:

Params:

Pick up a value extraction type: Value in XY file

Select a file: \Rafal\APROSYS\S-BeamTest\RadiossData\IE.lst

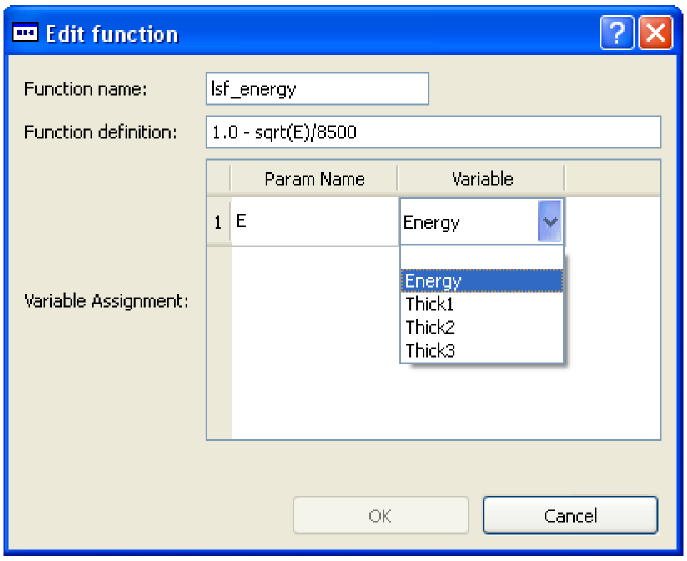
[✓ Last value of Y] Max(Y) Min(Y) Avg(Y) Intergral over X Y(X) for X =

Y(X) for X = 20.0

Fig. 4: Interface for external variables definition. A sequence of three programs needs to be executed in order to get the output file as a time history function. The value of the variable is obtained as the last one from the tabulated relationship.

2.2 Definition of the limit state function

In STAND the limit state function is symbolically given in the standard math notation (similar as e.g. in MATLAB) as a function of the basic random variables and/or external variables. The definition may include most of the basic mathematical functions. When typing the LSF formula there is no need to use the same variable names as the ones specified in the stochastic model. The variables involved in the expression are identified by parser and the user can subsequently associate them with the random and/or external variables, see Fig. 5. If the external variables appear in the definition, it is identified which basic random variables implicitly influence the LSF by analyzing which of them are involved in modifying input data files used by programs that generate values of the considered external variables.


Edit function

Function name: lsf_energy

Function definition: 1.0 - sqrt(E)/8500

Variable Assignment:

	Param Name	Variable
1	E	Energy

Fig. 5: Definition of the limit state function.

2.3 Reliability analysis algorithms

One of the reasons for developing STAND was to provide a user friendly reliability analysis package, which can be employed for safety analysis of various technical problems. Therefore, a collection of basic, general purpose reliability analysis methods as well as specialized, application oriented algorithms is available in STAND.

The most computationally efficient methods for failure probability estimation are based on an approximation of the failure domain in the standard normal space $\boldsymbol{\mathcal{U}}$. In the first order reliability method (FORM) the failure domain is approximated by the half space that is defined using the limit state surface linearized in the so-called design point. In the standard normal space the design point $\mathbf{u}^*$ (also called the most probable failure point or the β-point) is the point on the limit-state surface which is closest to the origin. It is obtained by solving the constrained optimization problem:

$$\mathbf{u}^* = \arg\min_{\mathbf{u}\in\Delta_f} \|\mathbf{u}\| \tag{4}$$

where Δ_f is the failure domain Ω_f transformed to the standard normal space $\Delta_f = \{\mathbf{u} : h(\mathbf{u}) \leq 0\} = \{\mathbf{u} : g(\mathbf{T}^{-1}(\mathbf{u})) \leq 0\}$. The LSF transformed to the $\boldsymbol{\mathcal{U}}$ space is denoted as $h(\mathbf{u})$. There are two standard, gradient based algorithms for solving this problem implemented in STAND. These are the Abdo-Rackwitz-Fiessler algorithm (ARF) [1] and the improved Hasofer-Lind-Rackwitz-Fiessler algorithm (HLRF) [25]. Knowing the design point $\mathbf{u}^*$ the so-called reliability index is computed as $\beta = \boldsymbol{\alpha}\mathbf{u}^*$, where $\boldsymbol{\alpha} = -\nabla h(\mathbf{u}^*)/\|\nabla h(\mathbf{u}^*)\|$ is the negative normalized gradient of the transformed LSF h at the design point. Taking advantage of the standard normal space properties the first order approximation of the probability of failure is given by

$$P_f \approx P_f^{\mathrm{FORM}} = \Phi(-\beta). \tag{5}$$

Another standard reliability analysis metod implemented in STAND is the second order reliability method (SORM) (cf. [2,7]), which provides more accurate approximation of the failure domain. In SORM, the limit state surface $h(\mathbf{u}) = 0$ is replaced by a second order surface fitted at the design point $\mathbf{u}^*$. It can be shown that the SORM approximation of failure probability is defined by

$$P_f \approx P_f^{\text{SORM}} = \Phi(-\beta) \prod_{i=1}^{n-1} (1 + \kappa_i \beta)^{-\frac{1}{2}}, \tag{6}$$

where κ_i, $i = 1, \ldots, n-1$ are the principal curvatures of the limit state surface at the design point in appropriately rotated coordinate system. Despite the better estimation accuracy offered by SORM, for practical reliability analysis problems this approach may turn out to be inefficient. Computation of the Hessian matrix will require at least $n(n-1)/2$ LSF calls more than FORM if the finite difference approach is employed. For large n this number can be prohibitively high. Moreover, the probability of failure P_f^{SORM} is estimated by local limit state surface approximation and its accuracy depends on how different is the actual shape of the limit state surface from the hyperparabolid. Obviously applicability of FORM/SORM methods is limited to problems with sufficiently smooth limit state function.

If the limit state function does not satisfy regularity requirements of the approximation methods, probability of failure can be estimated by the Monte Carlo integration. In the basic crude Monte Carlo (CMC) method implemented in STAND a random sample $\{\mathbf{U}_1, \mathbf{U}_2, \ldots, \mathbf{U}_K\}$ is generated using n dimensional standard normal distribution. The sample elements are transformed to the original random space $\mathbf{X}_i = \mathbf{T}^{-1}(\mathbf{U}_i)$, $i = 1, \ldots, K$ and the LSF values $g(\mathbf{X}_i) = h(\mathbf{U}_i)$ are computed. The probability of failure is estimated by the following empirical average:

$$\hat{P}_f^{\text{CMC}} = \frac{1}{K} \sum_{i=1}^{K} I_{\Delta_f}(\mathbf{U}_i), \tag{7}$$

where $I_{\Delta_f}(\cdot)$ is the so called failure domain indicator function such that $I_{\Delta_f}(\mathbf{U}_i) = 1$ if $h(\mathbf{U}_i) \leq 0$ and $I_{\Delta_f}(\mathbf{U}_i) = 0$ when $h(\mathbf{U}_i) > 0$. The estimate (7) is normally distributed random variable with the mean equal to the probability of failure and the variance

$$\mathbb{V}\text{ar}[\hat{P}_f^{\text{CMC}}] = \frac{1}{K}(P_f - P_f^2). \tag{8}$$

The accuracy of Monte Carlo estimate is quantified by the coefficient of variation, defined as

$$\nu_{\hat{P}_f} = \frac{\sqrt{\mathbb{V}\text{ar}[\hat{P}_f^{\text{CMC}}]}}{P_f} = \sqrt{\frac{1 - P_f}{K P_f}}. \tag{9}$$

It is easy to check that for problems where the expected probabilities of failure are low, $10^{-7} \div 10^{-3}$, to get an accurate result, say $\nu_{\hat{P}_f} = 5\%$, it is required to perform $K = 4 \cdot 10^5 \div 4 \cdot 10^9$ simulations. This computational burden is certainly not acceptable, especially when obtaining LSF values requires a nonlinear FE analysis.

The method that allows to significantly reduce the number of necessary simulations is the importance sampling (IS). It can be shown that in the $\boldsymbol{\mathcal{U}}$ space the importance sampling estimator for failure probability $\hat{P}_f^{\text{IS}}$ takes the form

$$\hat{P}_f^{\text{IS}} = \frac{1}{K}\sum_{i=1}^{K} I_{\Delta_f}(\mathbf{V}_i)\frac{\varphi_n(\mathbf{V}_i,\mathbf{0},\mathbf{I})}{\varrho_{\mathbf{V}}(\mathbf{V}_i)}, \tag{10}$$

where $\varrho_{\mathbf{V}}(\cdot)$ is the joint PDF (also called the sampling density) of the n-element random vector $\mathbf{V}$, and $\varphi_n(\cdot,\mathbf{0},\mathbf{I})$ is the n-dimensional normal joint PDF with zero mean values and unit covariance matrix. In order to compute the value of the above estimator we use the realizations $\mathbf{v}_i$ of the vector $\mathbf{V}_i$ generated from $\varrho_{\mathbf{V}}(\cdot)$. The key element for the efficiency of importance sampling method is a good choice of the sampling density that should minimize the variance of (10). Unfortunately, in general, derivation of the optimal $\varrho_{\mathbf{V}}(\cdot)$ function is difficult. However, very often quite substantial improvement of the computational efficiency can be obtained by selecting it as the n-dimensional normal PDF located over the design point $\mathbf{u}^*$ – the region with the largest contribution to the value of P_f (see [20])

$$\varrho_{\mathbf{V}}(\mathbf{v}) = \varphi_n(\mathbf{v},\mathbf{u}^*,\mathbf{I}) = \prod_{i=1}^{n}\varphi(v_i - u_i^*). \tag{11}$$

In such a case the estimator is given by

$$\hat{P}_f^{\text{IS}} = \frac{1}{K}\sum_{i=1}^{K} I_{\Delta_f}(\mathbf{V}_i)\frac{\varphi_n(\mathbf{V}_i,\mathbf{0},\mathbf{I})}{\varphi_n(\mathbf{V}_i,\mathbf{u}^*,\mathbf{I})}. \tag{12}$$

The above importance sampling approach has been implemented in STAND. With the choice of multi-normal sampling density (11) the estimate of P_f is not very sensitive to the shape of Δ_f and unless it is extremely nonlinear, the "success" rate for sample points selected from $\varrho_{\mathbf{V}}(\cdot)$ is about 50% (i.e. approximately equal likelihood of falling into either the safe or the failure domain). This is in marked contrast to the conventional Monte Carlo method for which the probability of having a sample point in the failure region is approximately equal to the probability of failure to be computed. Experience shows, that in most of the cases only few thousand simulations are necessary to get good P_f assessment with the estimator (12) (the estimation quality measured by the coefficient of variation of the estimator), many orders of magnitude less than for crude Monte Carlo sampling.

Of course, there exist some limitations of the above importance sampling procedure. Listing after [15] only these relevant to sampling in the $\boldsymbol{\mathcal{U}}$ space and single LSF, these are: a poor choice of the sampling distribution $\varrho_{\mathbf{V}}(\cdot)$, multiple design points of the LSF under consideration (local minima of the function $\|\mathbf{u}\|$ provided $h(\mathbf{u}) = 0$) or extremely concave LSF, which may lead to low efficiency in sampling. A way to deal with these problems is using adaptive methods of selecting the sampling density. It can be shown that the importance sampling is more effective when the sampling PDF $\varrho_{\mathbf{V}}(\cdot)$ is more closely proportional to the original sampling density in the failure region Ω_f (or Δ_f if sampling in the $\boldsymbol{\mathcal{U}}$ space). A popular approach is to let $\varrho_{\mathbf{V}}(\cdot)$ be a composite of k pre-selected elementary sampling densities $\varrho_{\mathbf{V}}^{(i)}(\cdot)$

$$\varrho_{\mathbf{V}}(\mathbf{v}) = \sum_{i=1}^{k} w^{(i)} \varrho_{\mathbf{V}}^{(i)}(\mathbf{v}), \tag{13}$$

where $w^{(i)}$, $i = 1, \ldots, k$ are weights to be determined so as to let $\varrho_{\mathbf{V}}(\cdot)$ approach $f_{\mathbf{X}}(\cdot)$ in Ω_f (or $\varphi_n(\cdot, \mathbf{0}, \mathbf{I})$ in Δ_f). Each of the $\varrho_{\mathbf{V}}^{(i)}(\cdot)$ in Eq. (13) may be pre-selected or may be updated with increased knowledge of the problem as sampling progresses. In [10] Karamchandani et al. selected $\varrho_{\mathbf{V}}^{(i)}(\cdot)$, $i = 1, \ldots, k$ to be equal to the original PDF but with the mean shifted to some selected representative points, denoted $\hat{\mathbf{v}}^{(i)}$. The corresponding weights are defined (in case of sampling in the standard normal space) as

$$w^{(i)} = \frac{\varphi_n(\hat{\mathbf{v}}^{(i)}, \mathbf{0}, \mathbf{I})}{\sum\limits_{j=1}^{k} \varphi_n(\hat{\mathbf{v}}^{(j)}, \mathbf{0}, \mathbf{I})}. \tag{14}$$

This means that the weights are proportional to the contributions of the respective representative point to the current estimate of the failure probability. It should be clear that the effectiveness of sampling in developing the ideal sampling density depends very much on the set of initial k representative points and on the initial sampling distribution from which they are drown. The above version of the multimodal adaptive important sampling approach is available in STAND.

The reliability analysis by simulation methods can be further facilitated by using Latin hypercubes for sampling. The Latin hypercubes are the plans of numerical experiments that populate the random space in much more efficient way, when compared to random sampling, see [21]. In STAND the so-called Optimal latin hypercube (OLH) sampling is employed [12]. It offers better analysis efficiency with respect to classical (random) Latin hypercubes. However, especially for high dimensional problems, OLHs require additional effort for finding optimal localization of the sample points. To overcome this STAND offers a choice of OLH generation algorithms as well as a large database of pre-generated OLH designs.

For problems that involve computationally expensive structural analysis or when the structural response exhibits numerical instabilities and discontinuities, reliability computations are often carried out using LSFs approximated by response surfaces (metamodels). Based on a set of numerical experiments, the parameters of the approximating function can be determined according to some criteria - usually by minimizing the least-squares approximation error. In most of the cases response surfaces are designed such, that a complex functional relationship between the structural response and the basic variables is described by an appropriate, but preferably as simple as possible mathematical model. Therefore, in the area of reliability assessment the most common choice are linear or quadratic response surfaces, see [6, 19].

The response surface methodology is a valuable tool for problems with implicit limit state functions. However, for problems with large number of random variables, even efficient design of experiments (DOE) techniques such as fractional factorials, central composite design or Box-Behnken design become unaffordable. When using a response surface as a surrogate of the actual LSF the accuracy of reliability estimate is greatly affected by the accuracy of the response surface approximation. Therefore, it is rather difficult to propose an accurate and efficient reliability estimation by using the traditional response surface approach. Bearing this in mind, an efficient adaptive response surface methodology for localizing the design point has been implemented in STAND. From the authors' names of the initial version of this method, see [26], it it denoted as ZMMM in the program's GUI. The two numerical examples presented in Sec. 4 employ ZMMM, therefore, it is appropriate to present it below in more detail.

2.3.1 ZMMM design point search method

The main issue in the presented algorithm is to enable its convergence for highly nonlinear limit states. The most probable failure point is obtained by repeatedly searching for the local design point (by some constrained optimization procedure) within the current "trust region" and updating the trust region until convergence. A polynomial response surface is used for the local approximation of the implicit LSF. It is based on sample points generated within the trust region by the OLH design. The following steps describe the iterative procedure realized in the standard normal space:

1. *Define the initial trust region (box) – its position and size.*
   Usually, the initial box center is assumed at the origin and the box size has to be big enough to include the sample points from safe as well as failure domains. This, of course, implies some initial knowledge of the problem. If no prior information is given the hypercube with the side length equal to 6 may be assumed, which corresponds to $[-3\sigma_{X_i}, 3\sigma_{X_i}]$ ranges of the random variables, where σ_{X_i} is the standard deviation of the i-th random variable.

2. *Generate a number of points using the OLH-based DOE.*
   The sample points are realizations of uniformly distributed random variables with mean values at the box center and the bounds determined by the box size. The points are transformed back to the original space and the corresponding LSF values are computed. Other DOEs, like central composite design, factorial design or axial design can be also used provided that the number of variables is not too large.

3. *Build a local LSF approximation with linear or quadratic response surface function.*
   In the study presented in this article the moving least-squares approach (MLS) is used for finding coefficients of the regression equation. For the MLS method, weights are assigned to the squared difference between OLH sample points and the point $\mathbf{u}$ where the LSF approximation is to be computed. By using a weight function $w(\mathbf{u} - \mathbf{u}_i)$, more emphasis/weight is locally placed on those experiments $\mathbf{u}_i$, which are close to $\mathbf{u}$. Hence, the response surface coefficients are reevaluated each time a new $\mathbf{u}$ point is considered. Such models are able to account for higher than second-order nonlinearity with simple polynomial models. The LSF model $\tilde{h}(\mathbf{u})$ built upon the results of designed experiments has the form

$$\tilde{h}(\mathbf{u}) = \mathbf{a}^{\mathrm{T}}(\mathbf{u})\mathbf{b}(\mathbf{u}) \tag{15}$$

where

$$\widehat{\mathbf{b}} = (\mathbf{A}^{\mathrm{T}}\mathbf{W}\mathbf{A})^{-1}\mathbf{A}^{\mathrm{T}}\mathbf{W}\mathbf{h} \tag{16}$$

is the least-square estimate for the unknown parameters $\mathbf{b}$, $\mathbf{W}$ is the weights matrix, $\mathbf{A}$ the regression design matrix, $\mathbf{a}$ the vector of linear independent regression functions and $\mathbf{h}$ is the vector of LSF values computed at OLH generated experimental points. An exponential, multi-dimensional weight function similar to Gaussian distribution is usually employed. $\mathbf{W}$ is a diagonal matrix with the weights

$$w_{ii} = \exp\left[-\sum_{j=1}^{n}(u_{ij}^{\mathrm{ex}} - u_j)^2/(2n\eta^2)\right], \quad \eta > 0. \tag{17}$$

In the above equation u_{ij}^{ex} stands for the j-th component of the i-th experimental point, u_j is the j-th component of the reference point $\mathbf{u}$ and η is a parameter controlling the shape of the weight function.

4. *Find an approximate design point location by solving the following optimization problem:*

$$\text{find:} \quad \mathbf{u}, \tag{18}$$

$$\text{that minimizes:} \quad \|\mathbf{u}\|^2 = \mathbf{u}^{\mathrm{T}}\mathbf{u}, \tag{19}$$

$$\text{subject to:} \quad \tilde{h}(\mathbf{u}) = 0, \tag{20}$$

where $\tilde{h}(\mathbf{u})$ is the local approximation of the LSF given by Eq. (15). The exact LSF value corresponding to the design point is computed and the point is added to the database of generated experimental points.

5. *Move the trust region center and decide if to change its size*

   5.1 If the design point found in the previous step is inside the trust region then the box center is moved there and the size of the trust region (measured by the length of the hypercube's side) is reduced. The reduction strategy is a very important factor for the convergence of the algorithm. The solution proposed in [26], consists in dividing the side of the box by a constant factor e.g., by two. Unfortunately, such an approach can lead to too rapid trust region reduction, which may greatly impair the convergence, especially for problems with many random variables and noisy LSFs. The method adopted in the algorithm implemented in STAND consists in dividing the volume of thet trust region rather than dividing the side length.

   5.2 If the design point found as the solution of the problem (18)–(20) is outside the trust region then it is projected on the trust region boundary in the direction of the current trust region center. The projection point becomes the new box center but the trust region size is not changed. However, sometimes it may be advantageous to enlarge it, especially when a number of consecutive iterations lead to design points outside the trust region. This is usually the case for nonlinear LSFs. Basing on the experience gained by performing a number of numerical tests it is proposed to adopt a strategy where three consecutive design points outside the trust region result in the trust region expansion by the same factor as used for the reduction.

6. *Repeat the steps 2 to 5 until the convergence criteria are satisfied.*
   The convergence check is performed only if the design point is found inside the trust region (see step 5.1). In addition to the stop criterion based on the distance between the last two iteration points (box centers), it is proposed to verify if the design point approximation is really located on the limit state surface $h(\mathbf{u}) = 0$. The LSF value at $\mathbf{u}^*$ is checked whether it is in an epsilon vicinity of zero or not.

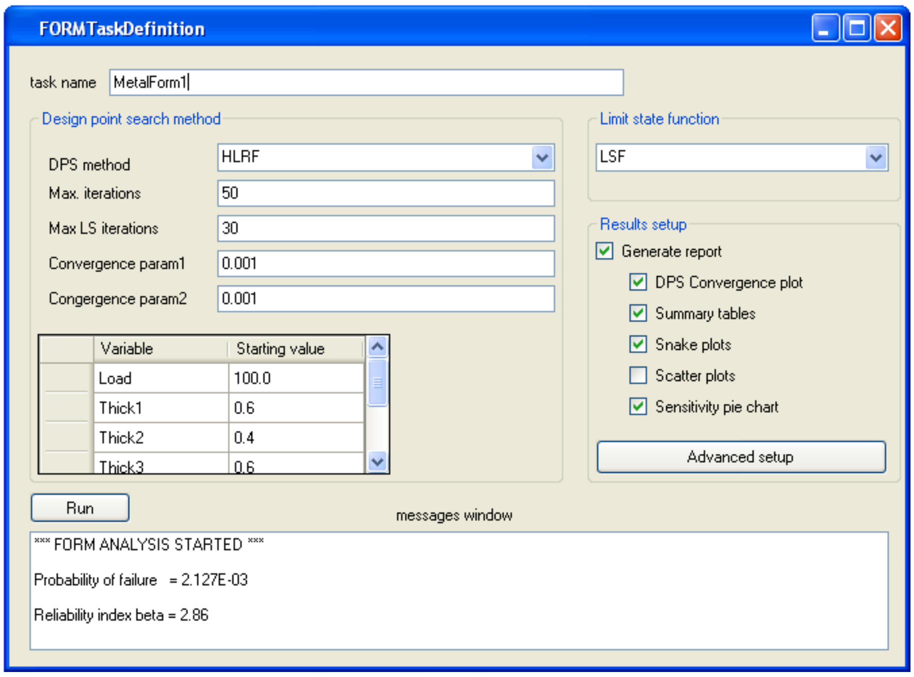
When constructing a local LSF approximation, all the points that 'fall' into the current box should be used in the regression analysis, i.e. not only newly generated points but also the sample points from previous iterations.

2.4 Submission of reliability analysis problem

The window used for defining and submitting the reliability analysis task is presented in Fig. 6. The user can type here the task name, select the algorithm and set values of its control parameters such as convergence parameter, maximal number of iterations etc. The LSF used in the analysis must be chosen from the list containing all the functions defined during the program session. For methods employing the design point concept the starting point for design point search algorithm can also be provided. The post-processing options allow for selecting types of plots and summary tables that will be included in the output report. Start button launches the analysis. Below the start button there is a message window communicating major results and, if possible, diagnosing problems that occurr during the computations.

As it was already emphasized before, in industrial applications of the reliability analysis the LSF value is usually computed using results produced by third party FE programs. Since this is almost always the most CPU time consuming part of the reliability analysis, therefore, an efficient strategy of submitting external FE tasks turns out to be a crucial issue. At many stages of the analysis parallel LSF calls significantly decrease the time of computations. Methods that are "inherently" parallel are all the simulation based techniques such as MC, IS or ZMMM. Moreover, the efficiency of many design point search methods can be also greatly improved by parallelizing the LSF gradient computations. STAND makes it possible to set the number of task that can be launched simultaneously.

Another important problem related to such time consuming tasks is the results recovery. Not to waste days or sometimes even weeks of expensive computations due to, e.g., the power cut or a wrong choice of a convergence parameter it must be possible to restart the analysis recovering as much as possible from the accidently broken run. To facilitate this process STAND stores input and output files corresponding to all LSF value computations in separate directories. When starting the analysis STAND first checks if the LSF values for the required realizations of the random vector already exist. If so, these values are simply read instead of performing expensive FE calculations. It must be remembered that since in many reliability analysis algorithms random values are used, the


FORMTaskDefinition

task name: MetalForm1

Design point search method

DPS method	HLRF
Max. iterations	50
Max LS iterations	30
Convergence param1	0.001
Congergence param2	0.001

Variable	Starting value
Load	100.0
Thick1	0.6
Thick2	0.4
Thick3	0.6

Limit state function: LSF

Results setup

- [✓] Generate report
  - [✓] DPS Convergence plot
  - [✓] Summary tables
  - [✓] Snake plots
  - [ ] Scatter plots
  - [✓] Sensitivity pie chart

```
*** FORM ANALYSIS STARTED ***
Probability of failure   = 2.127E-03
Reliability index beta = 2.86
```

Fig. 6: Run submission window

seed of random number generator must be stored to assure the same sequence of numbers during the recovery process.

2.5 Post-processing of results

The software provides a number of post-processing tools to better explore the results of reliability analysis. In the analysis report in addition to the standard results like the probability of failure and the reliability index value it is possible to visualize various types of data generated by the employed algorithm as well as some additional results, e.g., design point coordinates or reliability index sensitivities with respect to parameters of random variables distribution.

Realizations of the random variables and the corresponding LSF values computed during the Monte Carlo simulations or the design point search can be shown using scatter plots (see Fig. 7a) or the so-called snake plots (see Fig. 7b). Scatter plot is a standard post-processing tool for displaying 2D or 3D data points or projections of multidimensional data on these subspaces. On the other hand, snake plots provide an attractive opportunity of visualizing the “entire” multidimensional data points. The idea of the plot is to represent each point in the form of a polyline with vertices placed on equally spaced axes corresponding to all variables and LSF. The lower and upper ends of the axes are related to the observed minimal and maximal values. Polyline vertices are positioned on the axes according to the data point coordinates. Functionality of the snake plot implemented in STAND enables specifying narrowed ranges for the random variables and LSF values to be displayed. For example, filtering out all the realizations of random variables that lead to positive LSF values (by setting the appropriate range on the LSF axis) an information about the dominating failure scenario can be found.

The snake plots seem also to be a convenient way of visualizing design points, see Fig. 7c. Using $\pm 3\sigma$ ranges for the random variables it can be easily seen where, with respect to their mean values, are the most likely failure point realizations of random variables.

One of the major features of FORM is that it readily provides the sensitivity of the reliability index and the first-order estimate of the failure probability, P_f^{FORM}, with respect to any set of distribution or model parameters. These sensitivities are useful in identifying important sources of uncertainty and making decisions in the design process. For a moderate number of random variables an efficient tool for analyzing this information is a pie chart, Fig. 7d. By examining such a chart made for reliability index sensitivities with respect to standard deviations of random variables one can immediately find out, the quality of which variables (measured by their scatter) should be particularly controlled to ensure low failure probability.

3 Object oriented code architecture of STAND

The choice of object oriented paradigm in STAND development turned out to be very advantages in terms of the code organization as well as synchronization of the simultaneous work of many programmers. What is also important, by using the object oriented C++, which is one of the major programming languages taught in universities, it is far more likely that the young researchers can quickly get involved and contribute to the project and that the development process will be carried out in the future.

For decades the default programming language for computational codes was FORTRAN. It is an example of the so-called procedural programming paradigm, where the most important concept is algorithm. Still being able to produce high performance programs it is much less flexible in terms of code development and managing complex data structures. Current hardware capabilities allow to tackle more sophisticated computational tasks, which in consequence calls for adequate programming tools that facilitate efficient software development. Therefore, an evolution of the programming paradigm was necessary to cope with the increasing demand for simpler data management and for programming efficiency. It seems that the object-oriented paradigm meets all these needs. Instead of the algorithm, a proper design of data structures is the crucial point in the object-oriented approach. The most prominent concept in this programming philosophy is a possibility of combining the variables (attributes) and procedures (methods) in bodies called classes. The classes are designed to reflect as much as possible the natural "language" of analyzed problems. Another most important concept of the object-oriented modeling is its ability to create structures of classes. The mechanism called inheritance allows to make a structure of classes called hierarchy of classes. The hierarchy is a kind of graph similar to the genealogic tree. The connection between classes i.e. inheritance, means that the lower class (so-called subclass) inherits the attributes and method from the upper class (base class). It allows to effectively utilize common properties of classes. In the case of object oriented code for reliability analysis problems the common attribute

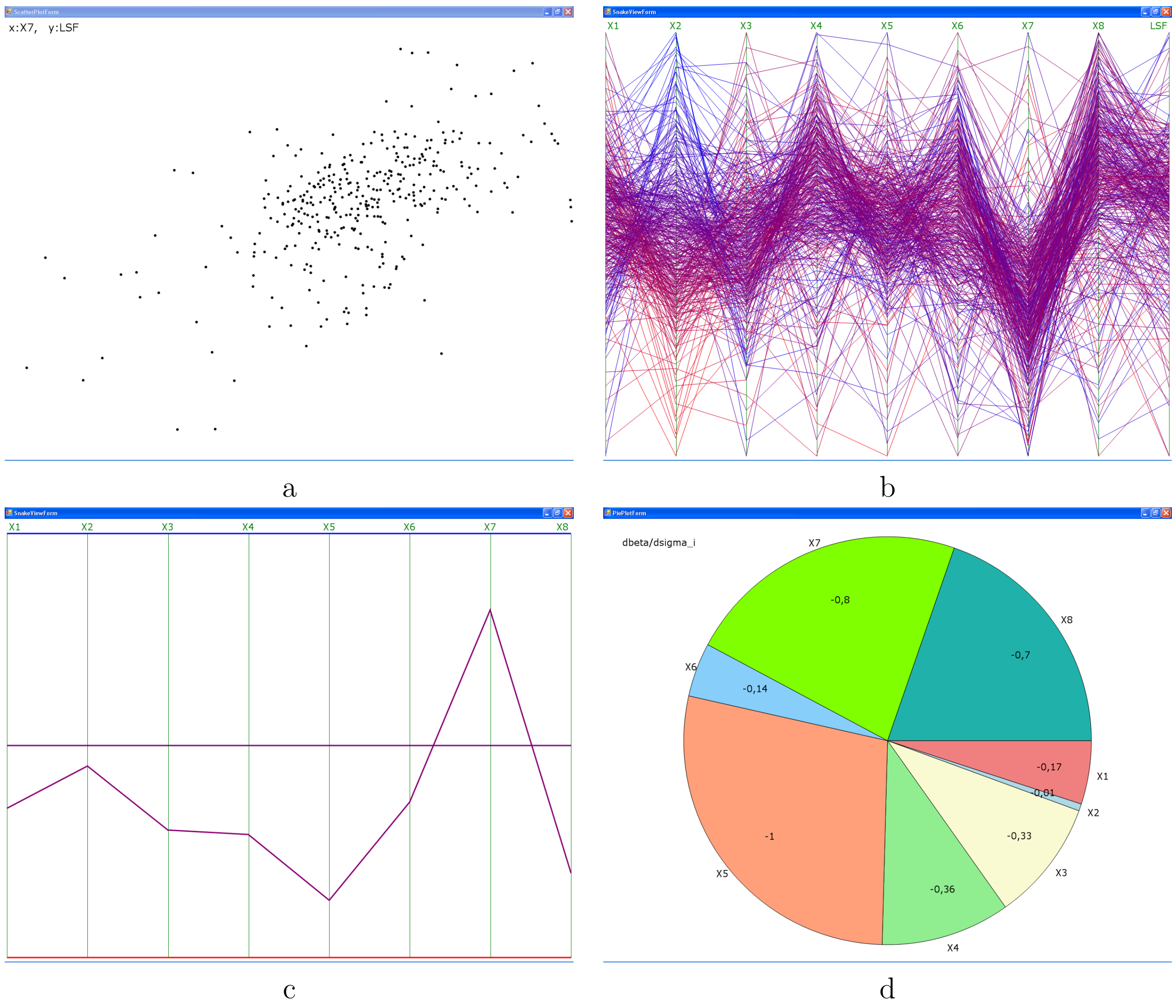


Fig. 7: a - scatter plot, b - snake plots, c - snake plot used to present the design point d - reliability index sensitivities

of classes corresponding to various solution algorithms (FORM, SORM, Monte Carlo ...) is the vector containing random variables (Random Variable type objects). Because all algorithms must have this vector, it is declared in the base class and thanks to inheritance all classes posses it. Using another mechanism of the object-oriented approach called polymorphism the inherited method can be redefined by declaring this method in the class once again and defining its own version of the body. It allows to cope easily with differences between classes. Such a class structure is also called a generalization-specialization because the base class declares most general behavior of the object while the descending classes are getting more and more specific. Some methods defined in the base class constitute an interface allowing to communicate with other objects. Knowing the "fixed" interface methods a separate programmers or teams could work on their respective parts of the code being sure that the objects will cooperate.

All these new features allow to create the model of real-world problems very efficiently. Detailed presentation of the object-oriented features exceeds the framework of this paper. As a comprehensive reference one should mention here the book of Bjarne Stroustrup [23].

Figure 8 presents (much simplified for clarity) class hierarchy diagram of STAND. The most important classes are Reliability Analysis, Random Variable, Distribution, Limit State Function, Response Surface, Design Point Search, Normal Space Transformation and Sampler. The names are self explanatory. The base class Reliability Analysis introduces common properties of the reliability analysis such as random variables and limit state functions. Two classes representing the Monte Carlo analysis and the so-called MVFO method (mean value first order) directly inherit from Reliability Analysis. The class DPS Reliability Analysis, derived from the base class is more specialized but still an abstract class. It represents a reliability analysis method that uses design point search algorithm. Four such classes were implemented in STAND. These are FORM, SORM, IS and MAISM (multimodal adaptive importance sampling), which are derived from DPS Reliability Analysis. Common properties declared in the base class are the kind of interface allowing to cooperate with other objects. This feature is illustrated, for example, by the "diamond" type link between DPS Reliability Analysis and Design Point Search classes. Inheritance is schematically denoted in Fig. 8 by the "triangle" links.

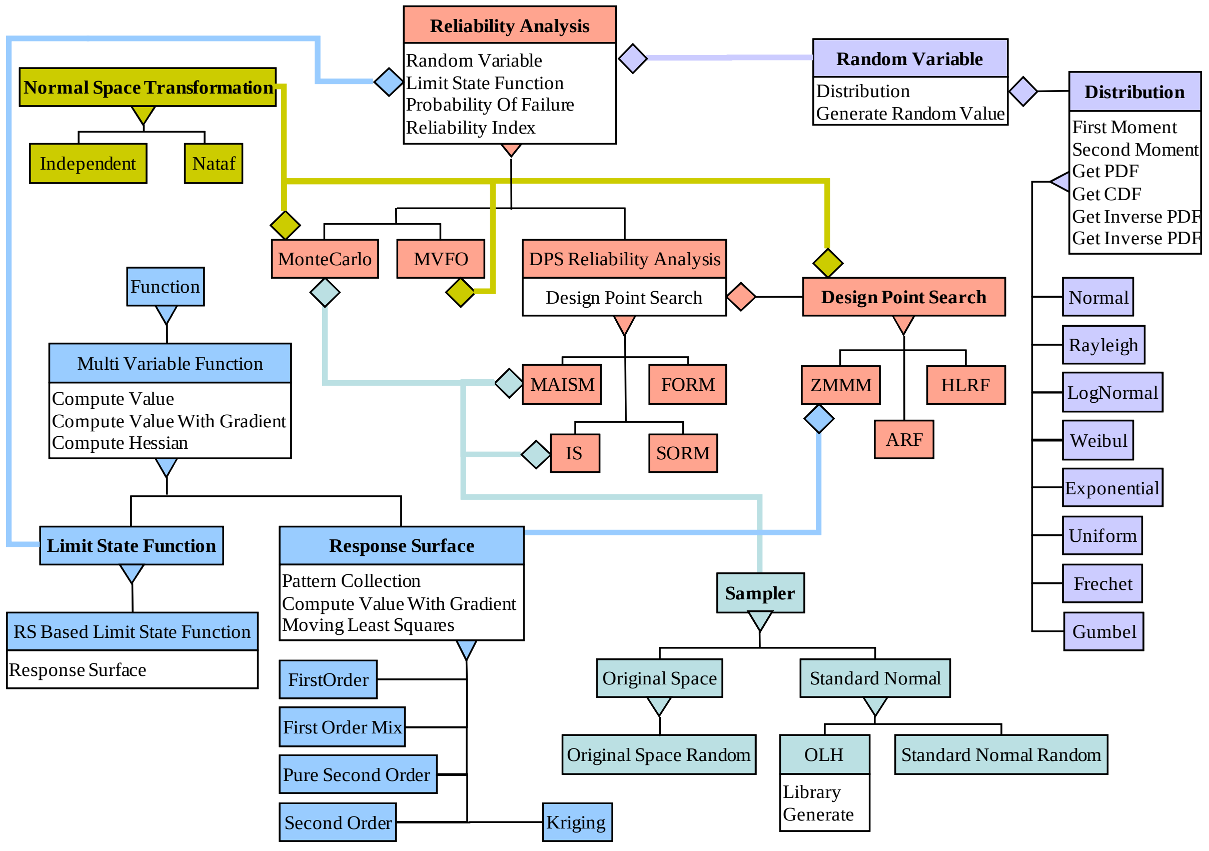


Fig. 8: Simplified class hierarchy diagram of STAND

4 Examples of interfacing STAND with commercial FE programs

4.1 Interfacing with ABAQUS - Reliability analysis of a plane stress problem accounting for geometrical uncertainty

STAND capabilities of interfacing with third party programs are illustrated on the simple example of structural reliability analysis, in which LSF is computed using the state of the art FE code ABAQUS.

Let us consider the elastic plane stress problem shown in Fig. 9. The triangular element with a circular hole is made of steel. It is loaded by uniformly distributed forces acting on the bottom edge of the triangle. The right edge is clamped. The thickness of the element is equal to 0.3 cm. The stochastic model consists of five normally distributed random variables, which are listed in Tab. 1.

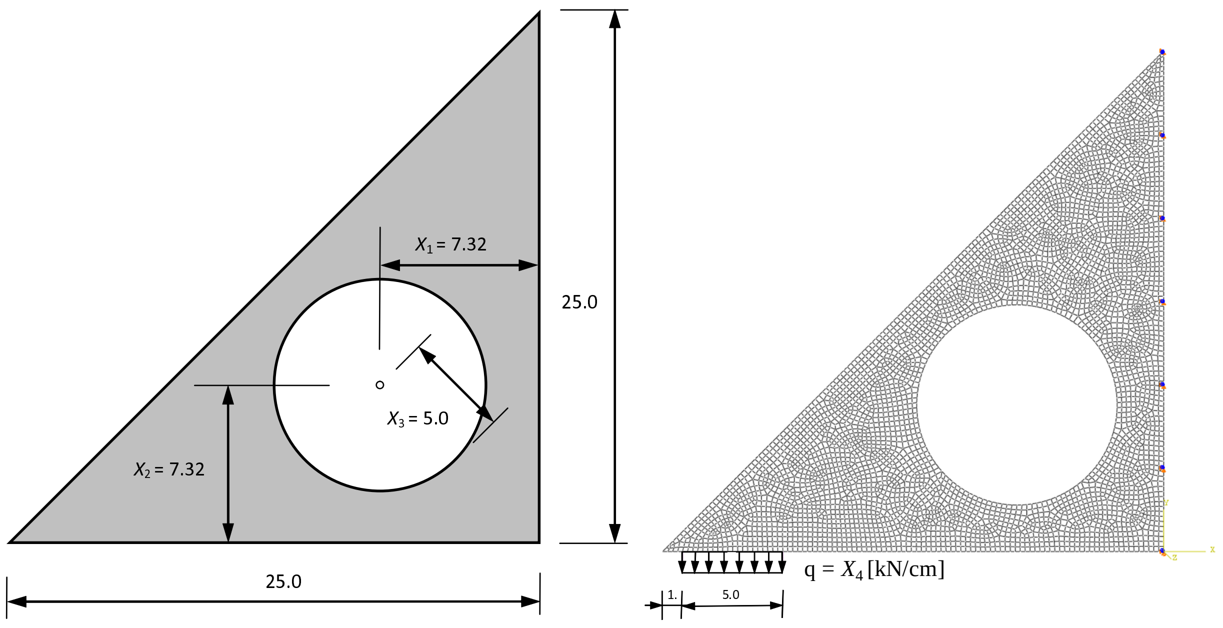


Fig. 9: Left - dimensions of the triangular element. Right - FE mesh, loading and boundary conditions. Dimensions are in centimeters.

It is assumed that failure occurs when the admissible reduced Huber-Mises stress $S_a = 21$ kN/cm^2 is exceeded at any point of the structure. Thus, LSF can be given as:

$$g(\mathbf{X}) = 1 - \frac{\max[S(\mathbf{X})]}{S_a}, \tag{21}$$

where $\max[S(\mathbf{X})]$ denotes the maximal Huber-Mises stress computed over the nodal points of the FE mesh.

STAND communicates with third party programs using text files. Realizations of the random variables are written in input files and the corresponding values of structural responses are retrieved from output files. In case of ABAQUS the FE model and all

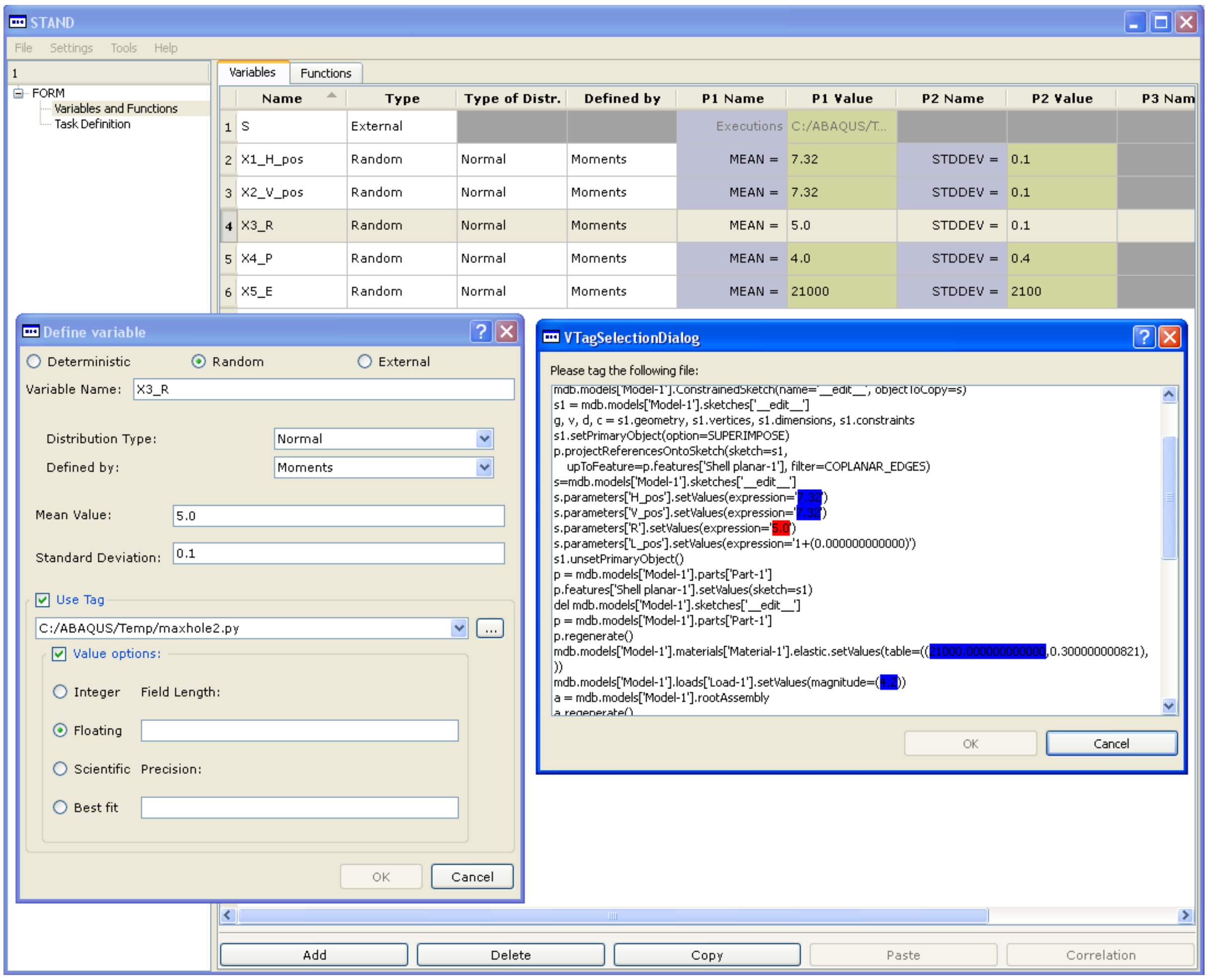
Var.	Description	Mean value	Standard dev.	Value at des. point
X_1	x coord. of the circle center	7.32 cm	0.1 cm	7.50 cm
X_2	y coord. of the circle center	7.32 cm	0.1 cm	7.45 cm
X_3	radius	5.0 cm	0.1 cm	5.07 cm
X_4	load magnitude	4.0 kN/cm	0.4 kN/cm	4.76 kN/cm
X_5	Young modulus	21000 kN/cm^2	2100 kN/cm^2	20089 kN/cm^2

Tab. 1: Parameters and description of normally distributed random variables. The last column - design point coordinates.

the analysis details are always defined in the text files with extension .inp and it can be requested to write the results in the output files with extension .dat. However, in our problem, substantial modifications of the FE model due to the geometrical nature of variables X_1, X_2 and X_3 would lead to changes in the mesh configuration that are too hard to be "programmed" in the .inp data file. In addition, an application of gradient-based design point search algorithms would require LSF values to be given with much higher precision than the one available in .dat files. Therefore, it was decided not to operate on the .inp and .dat files but rather to use the ABAQUS API python script to modify the ABAQUS model data base and extract results from the corresponding output data base. The model data base was defined in the user friendly graphical environment of ABAQUS CAE. A separate ABAQUS CAE session is also called each time the reliability analysis algorithm sends a new realization of the random variable vector. Such a session was previously recorded as the Python script abaqus.rpy. Normally ABAQUS CAE generates the abaqus.rpy to recover the session in case of the system crash. However, we found it useful to modify this script and use it to run ABAQUS computations for STAND generated realization of $\mathbf{X}$. The modifications include commands to proceed in a "silent" mode (without ABAQUS GUI) and additional instructions for opening the output data base, finding maximum value of the Huber-Misses stress and saving it in an appropriate format in a text file. Of course, in the script the mesh generation command is invoked for the current geometry of the circular hole, which allows to avoid direct mesh redefinition in the .inp file.

The stochastic model was defined using the graphical interface of STAND. The program windows for introducing the variables listed in Tab. 1 are presented in Fig. 10. It can be seen that when the "Use Tag" box is checked realizations of the random variable generated by reliability algorithms are to be written in a selected input file. The place where the value of the random variable is to be written is specified by highlighting a text field in the previously specified input data file. As it is shown in the Fig. 10, the fields already assigned to other random variables are highlighted in blue color while the field corresponding to the random variable that is currently being defined is marked in red.

In our test problem there is only one external variable. It is the maximal Huber-Mises stress S. The GUI window used for definition of this variable is shown in Figure 11. The "External programs" table displays the sequence of program paths with execution parameters that are used in order to compute the value of S. In the present example the abaqus.bat batch program is run with options resulting in executing the python script



	Name	Type	Type of Distr.	Defined by	P1 Name	P1 Value	P2 Name	P2 Value	P3 Nam
1	S	External			Executions	C:/ABAQUS/T...			
2	X1_H_pos	Random	Normal	Moments	MEAN =	7.32	STDDEV =	0.1	
3	X2_V_pos	Random	Normal	Moments	MEAN =	7.32	STDDEV =	0.1	
4	X3_R	Random	Normal	Moments	MEAN =	5.0	STDDEV =	0.1	
5	X4_P	Random	Normal	Moments	MEAN =	4.0	STDDEV =	0.4	
6	X5_E	Random	Normal	Moments	MEAN =	21000	STDDEV =	2100	

Define variable

○ Deterministic [✓ Random] ○ External

Variable Name: X3_R

Distribution Type: Normal

Defined by: Moments

Mean Value: 5.0

Standard Deviation: 0.1

[✓ Use Tag] C:/ABAQUS/Temp/maxhole2.py

[✓ Value options:]

○ Integer Field Length:

[✓ Floating]

○ Scientific Precision:

○ Best fit

VTagSelectionDialog

Please tag the following file:

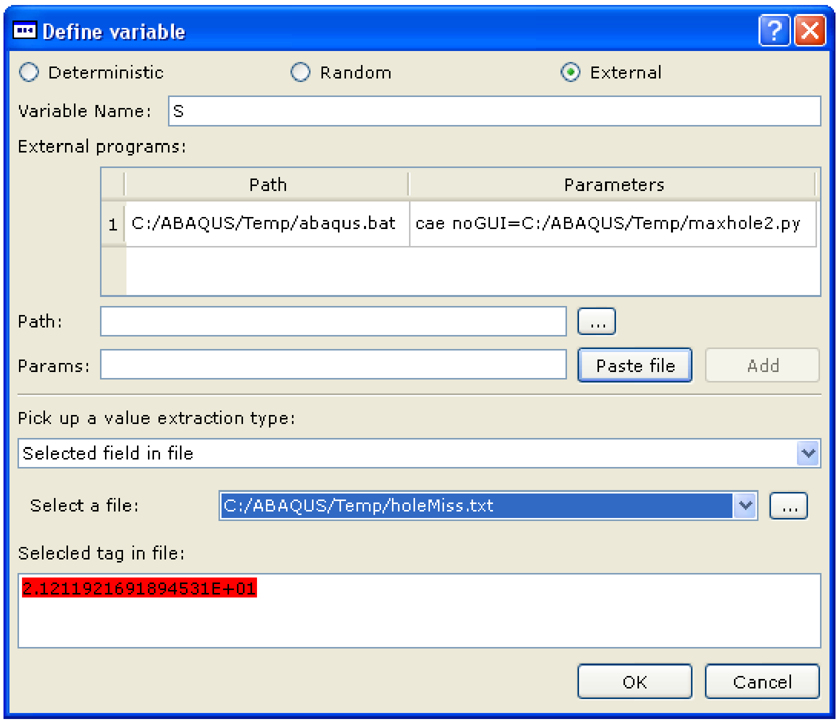
```
mdb.models['Model-1'].ConstrainedSketch(name='__edit__', objectToCopy=s)
s1 = mdb.models['Model-1'].sketches['__edit__']
g, v, d, c = s1.geometry, s1.vertices, s1.dimensions, s1.constraints
s1.setPrimaryObject(option=SUPERIMPOSE)
p.projectReferencesOntoSketch(sketch=s1,
   upToFeature=p.features['Shell planar-1'], filter=COPLANAR_EDGES)
s=mdb.models['Model-1'].sketches['__edit__']
s.parameters['H_pos'].setValues(expression='7.32')
s.parameters['V_pos'].setValues(expression='7.32')
s.parameters['R'].setValues(expression='5.0')
s.parameters['L_pos'].setValues(expression='1+(0.000000000000)')
s1.unsetPrimaryObject()
p = mdb.models['Model-1'].parts['Part-1']
p.features['Shell planar-1'].setValues(sketch=s1)
del mdb.models['Model-1'].sketches['__edit__']
p = mdb.models['Model-1'].parts['Part-1']
p.regenerate()
mdb.models['Model-1'].materials['Material-1'].elastic.setValues(table=((21000.000000000000,0.300000000821),
))
mdb.models['Model-1'].loads['Load-1'].setValues(magnitude=(4.2))
a = mdb.models['Model-1'].rootAssembly
a.regenerate()
```

Fig. 10: Random variables for the ABAQUS simulated reliability problem.

maxhole2.py by ABAQUS CAE without starting GUI. It is assumed that after the execution of the specified program sequence the result of interest is written in the text file holeMiss.txt. Note that this file contains a single field with the maximal value of Huber-Mises stress, which was generated by an appropriate python script especially designed for this example.

The limit state function (21) was entered as arithmetic expression using the implemented function parsing mechanism described in Sec. 2.2.

Before running the reliability computations some observations have been made considering performance of the employed FE analysis. First, the duration of a single ABAQUS run was long enough to abandon the idea of employing crude Monte Carlo simulation, at least for the purposes of this presentation. The other problem was the observed instability of the analysis results due to changes in the FE mesh. This phenomenon obstructs application of design point search methods, which use a finite difference approach for gradient approximation. Therefore, because re-meshing was unavoidable due to configuration changes, it was decided to employ the ZMMM design point search algorithm described in Sec. 2.3.1. Thanks to the adaptive response surface strategy the ZMMM algorithm is


Define variable

Deterministic Random [✓ External]

Variable Name: S

External programs:

	Path	Parameters
1	C:/ABAQUS/Temp/abaqus.bat	cae noGUI=C:/ABAQUS/Temp/maxhole2.py

Path:

Params:

Pick up a value extraction type: Selected field in file

Select a file: C:/ABAQUS/Temp/holeMiss.txt

Selected tag in file: 2.1211921691894531E+01

Fig. 11: Defining external variable S - maximal Huber-Misses stress.

much more likely to converge even in presence of the numerical noise generated by the re-meshing.

Using 50 OLH points for producing the LSF approximation the algorithm started from the origin in the standard normal space and the design point was found after 15 ZMMM iterations. The resulting FORM reliability index is equal to 2.99, which corresponds to the probability of failure approximation $P_f^{\mathrm{FORM}} = 1.4 \cdot 10^{-3}$. Comparing values of the random variables at the design point with their mean values (c.f. Tab. 1) only minor changes of the geometric variables and Young modulus can be observed, however, there is a considerable increase of the load magnitude, which turns out to be the most important source of failure.

4.2 Parallel processing with RADIOSS - Reliability of s-rail subjected to crash

RADIOSS [14] is an explicit finite element solver, developed by the Altair company for analysis of highly dynamic and nonlinear problems, in particular crash. The RADIOSS FE model is described in two text files, D00 file for RADIOSS Starter and D01 file for RADIOSS Engine. In the study presented below only "simple" random variables are taken into account. By this, we consider such variables which realizations are represented by single numbers in D00 or D01 files. However, there is also a possibility to account for more "complex" variables, like model translation or rotation, that lead to modifying numerous entries in data files. For this purpose it is necessary to use a script for HyperStudyDSS, which is one of the family of Altair programs, that can perform all types of manipulations on the RADIOSS data. In our example we are able to directly introduce modifications in D00 and D01.

Automatic job submission on parallel machines should be considered as a crucial component of efficient reliability analysis programs, especially these aimed at solving complex, computationally involved problems. Such a possibility is offered also by STAND. Since the underlying FE computations are very expensive (a single analysis by RADIOSS takes more than two hours of CPU) the reliability analysis was performed on a machine with 8 dual core processors, which allows for simultaneously running 16 FE tasks.

In the example we consider a thin-walled steel s-rail, shown in Fig. 12, clamped at one end and hit at the other end by the 100 kg mass moving with the initial velocity $v_0 = 15$ m/s (54 km/h) in the x-axis direction. The beam consists of 3 omega-shaped parts and the cover plate. The omega parts are attached to the cover with 64 spotwelds. The elastic-plastic-brittle material is assumed [9]. The finite element model consists of 5760 MITC4 type shell elements [5] and 64 spring elements to model the spotwelds.

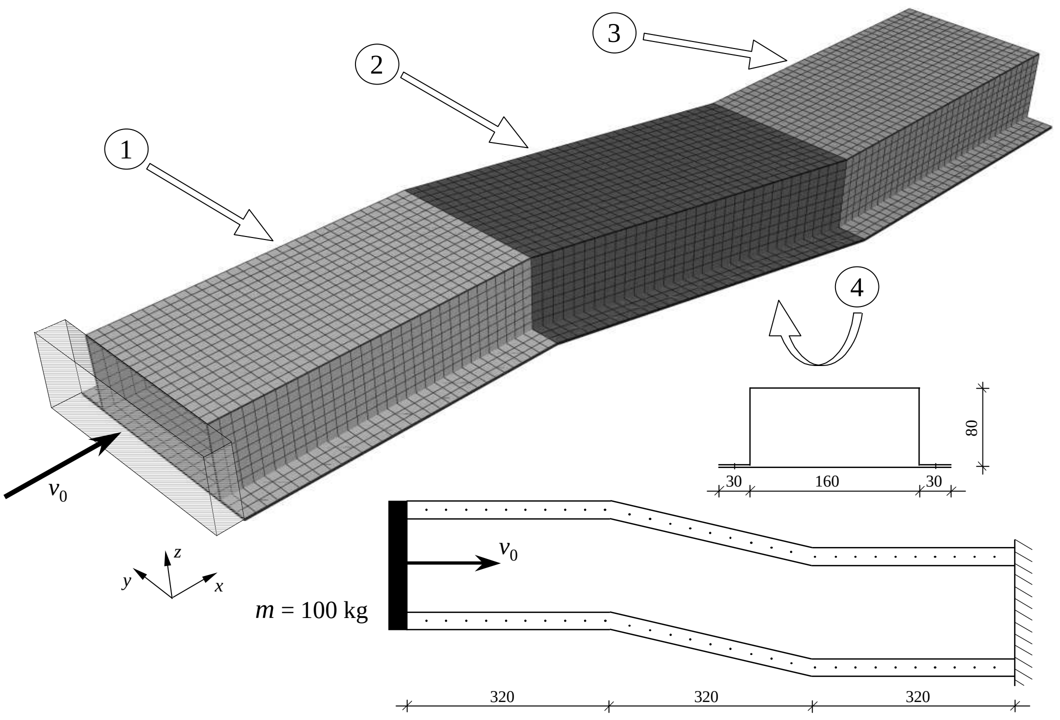


Fig. 12: S-rail crash problem. The finite element model and geometry. Dimensions are in millimeters. The arrows indicate position of the parts.

The beam acts here as an energy absorbing device, so the major concern in its design is to ensure a good energy management, which is collapsing in regular folding rather than buckling mode. However, very often a design that performs satisfactorily in the ideal (nominal) operating conditions is not reliable due to large sensitivity to unavoidable uncertainties of some parameters. For this reason it seems essential to check design sensitivity to parameter uncertainties by assessing the probability of its unsatisfactory behavior.

In our problem there are 8 basic random variables. Their description and properties has been presented in Tab. 2. They correspond to thicknesses of the sheet metal parts, material parameters and initial conditions. To address the problem of uncertain quality of spotweld connections 3 ($\approx 5\%$) randomly selected spring elements are always being deleted from the model, each time the crash analysis is performed. During the design point search, such a simplified approach may be considered as a way of accounting for some average influence of spot weld failures. Choosing randomly the spot weld elements to be deleted an additional noise effect is introduced to LSF computations. Therefore a specially adapted method, like the ZMMM algorithm, is needed.

	Description	Distribution	Mean	Std. dev.
X_1	t_1 - thickness of the part 1	lognormal	1.5 [mm]	0.075 [mm]
X_2	t_2 - thickness of the part 2	lognormal	1.5 [mm]	0.075 [mm]
X_3	t_3 - thickness of the part 3	lognormal	1.5 [mm]	0.075 [mm]
X_4	t_4 - thickness of the part 4	lognormal	1.0 [mm]	0.05 [mm]
X_5	σ_0 - yield stress	lognormal	180 [MPa]	15 [MPa]
X_6	E - Young modulus	lognormal	210000 [MPa]	21000 [MPa]
X_7	v_0^y - y component of the initial velocity of impacting mass	normal	0 [m/s]	1.5 [m/s]
X_8	v_0^z - z component of the initial velocity of impacting mass	normal	0 [m/s]	1.5 [m/s]

Tab. 2: Random variables of the s-rail crash problem

The crash duration is 20 ms. In order to define the reliability analysis problem an insufficient energy absorbtion is assumed as the failure event. The minimal admissible value of absorbed energy $e_{\min}$ is taken to be equal to 6000 J, which is about 80% of the energy absorbed by the nominal beam (corresponding to the mean values of random variables and perfect spot welds). Hence, the limit state function can be expressed as

$$g(\mathbf{X}, \mathbf{A}) = 1 - \frac{e_{\min}}{e(\mathbf{X}, \mathbf{A})}, \tag{22}$$

where $e(\mathbf{X}, \mathbf{A})$ is the energy absorbed by the beam and $\mathbf{A}$ stands for a vector of discrete two-point distributed random variables accounting for the good/failed state of spot welds. It is assumed that every realization of $\mathbf{A}$ produces 3 failed spot welds, which leads to removing the corresponding finite elements from the model. It was observed that due to random spot weld failures, the scatter of values of such defined LSF, measured by the maximal deviation from the sample mean, can be up to 0.07, depending on a given realization $\mathbf{x}$ of the vector $\mathbf{X}$. Since the analytical form of the joint probability density function of $\mathbf{A}$ variables is unknown and it is impossible to include them in the set of basic random variables $\mathbf{X}$, due to the strategy of spot weld removing their influence manifests through the noisy character of LSF.

The most probable failure point search is based on the adaptive response surface strategy. In computer simulated crash problems due to the nonlinear and noisy character of LSF the goal is to determine a vicinity of the design point rather than to find its "exact" location. Therefore, there is no reason for setting too strict convergence criteria since there is a high probability that they will never be fulfilled. In addition, the convergence criterion (described in point 6 of the ZMMM algorithm description) should account somehow for the dimension of the problem. The criterion of the form $d < \varepsilon$, where d is the distance between the last two iteration points (box centers) and, e.g., $\varepsilon = 0.1$, may be too restrictive for problems with many variables. A possible modification of this criterion could be $d < k\sqrt{n\varepsilon^2}$, the right-hand side being k times the length of the diagonal of n-dimensional hypercube, its side equal to ε. In the current example $n = 8$, $\varepsilon = 0.15$ and k is taken equal to 1, which gives the criterion $d < 0.42$. The second criterion imposed on the mean value of LSF at the design point is defined as $h(\mathbf{u}^*) < 0.1$. To avoid an extensive computational effort a limit is also set on the maximal number of iterations. Based on numerical experiments, it was decided to restrict the number of iterations to 15.

Another change with respect to the original algorithm consists in a different strategy of reducing the trust region. In [26] it was proposed to reduce the size of the trust region to the half of the size from previous iteration. Again, from numerical tests we have found that such an approach can often lead to a rapid reduction of the trust region and, in consequence, undesirable behavior of the algorithm. In such cases, the response surface is based on a very localized sample of points, which, accounting for the noise influence, often leads to design point approximations outside the trust region. This effect is particularly important in high dimensional sample spaces. In the version of the algorithm implemented for this study the volume of the trust region (n-dimensional hypercube) rather than its size is reduced by a constant factor. It is assumed that if the current design point approximation is inside the trust region then the new volume is taken to be 25% of the current one. This leads to a simple formula for the size reduction

$$b_i = 0.25^{\frac{1}{n}} b_{i-1}, \tag{23}$$

where b_i and b_{i-1} are the current and the previous trust region sizes, respectively. For $n = 8$ variables and initial size of the trust region equal to 6 such a strategy produces the following sequence of sizes: $6, 5.05, 4.24, 3.57, 3, 2.52, \ldots$, while the halving approach would give $6, 3, 1.5, 0.75, 0.325, 0.1625, \ldots$.

The linear response surface and the moving least squares strategy for weighted regression are employed. The OLH design with $4n = 32$ points is used as the plan of experiments.

The results of the ZMMM design point search are shown in Fig. 13. In the left hand side graph there is presented the history of changes of the FORM reliability index corresponding to subsequent iterations of the design point. Labels “IN” and “OUT” placed by the data point markers indicate whether the design point is found inside or outside the current trust region, respectively. The right hand side graph shows how the value of the convergence criterion changes over iterations. The criterion is fulfilled after 5 iterations and the following results are obtained:

$$\beta = 4.04, \quad P_f^{\text{FORM}} = 2.7 \cdot 10^{-5}, \tag{24}$$

$$\mathbf{u}^* = \{-0.89, -0.29, -1.20, -1.26, -2.19, -0.80, 1.92, -1.81\}, \tag{25}$$

$$h(\mathbf{u}^*) = 0.0038,$$

Analyzing coordinates of the point $\mathbf{u}^*$ it can be noticed that the most probable failure event will occur if thicknesses of parts 3 and 4 and the yield stress are well below their

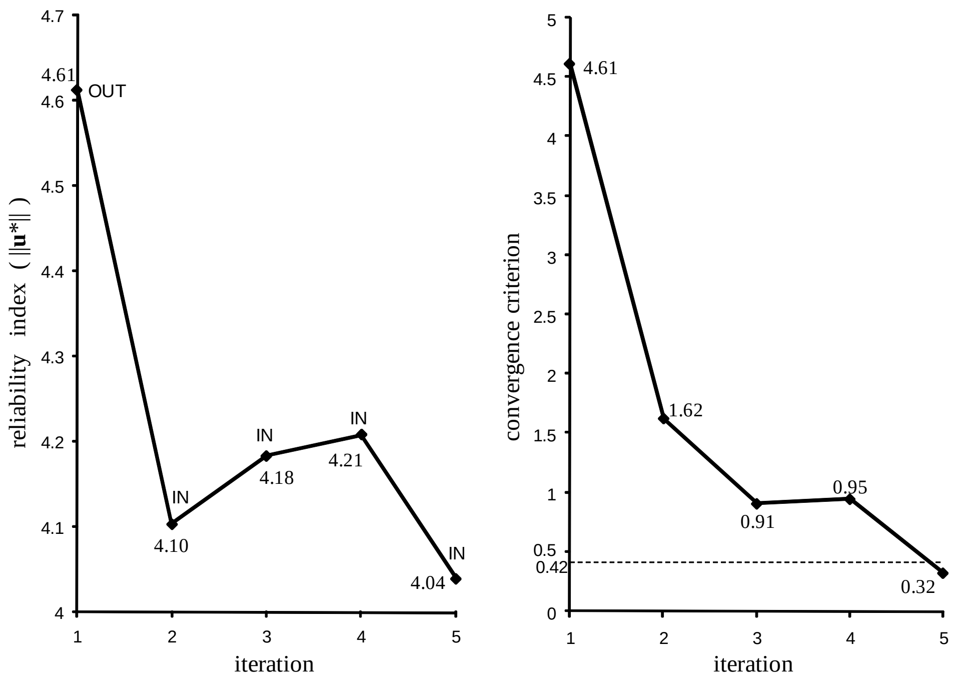


Fig. 13: Design point search: reliability index changes history and the convergence (distance) criterion history

respective mean values and there is an important lateral component of the velocity of impacting mass. It should also be noticed here that the point $\mathbf{u}^*$ corresponds to the most probable realization of the variables $\mathbf{X}$ leading to the failure event and some averaged influence of spot weld failures ($\mathbf{A}$ variables). Hence, it cannot be treated as the equivalent of the most probable failure point obtained for the problem where the random variables $\mathbf{A}$ with a known joint probability distribution would be included in the set of basic random variables. Accounting for $4n$ sample points used to fit each linear response surface, the design point verification calls and the computation of the average LSF value at the final design point the total number of RADIOSS calls was equal to 213.

5 Conclusions

The reliability analysis of complex engineering structures is a computationally expensive task that requires repeated structural analysis performed in most of the cases by dedicated FE programs. Unfortunately, still very few state of the art FE packages provide an advanced reliability analysis functionality. It is therefore of principal importance to develop the reliability analysis codes allowing for easy and efficient interfacing with external computational software.

There is a number of issues that need to addressed in order to make such an integration successful. Besides the user friendly graphical interface, which should automate as much as possible the process of updating FE data files with realizations of the random variables and extracting results, special reliability analysis techniques need to be employed that can cope with numerical noise due to either a solution method of the equilibrium problem or effects of re-meshing. The capability of parallel job processing seems also to be particularly significant for efficiency of the FE-based reliability analysis.

Presented in the paper the reliability analysis program STAND was meant to include all these important features. On the two examples of STAND interfacing with state of the art FE codes ABAQUS and RADIOSS, it was shown that by using the adaptive response surface based method in the design point search it was possible to analyze not trivial reliability analysis problems with noisy limit state functions. Direct application of classical gradient based design point search algorithms leads in these cases to convergence problems.

The object-oriented approach used in STAND development facilitates simultaneous work of many programmers and allows for easy code maintenance.

Acknowledgments
The partial support from the Polish Committee for Scientific Research grants, No. N519 010 31/1601 and No. 3T11F00930 is gratefully acknowledged. The authors would like to express their gratitude to the company Altair Development France for granting them the license for the Radioss software and to Viet Nguyen Hoang for his great contribution to STAND development.

References

[1] T. Abdo and R. Rackwitz. A new beta-point algorithm for large time-invariant and time-variant reliability problems. In A. Der Kiureghian and P. Thoft-Christensen, editors, *Reliability and Optimization of Structural Systems '90, Proc. 3rd WG 7.5 IFIP Conf., Berkeley, 26–28 March 1990*, pages 1–12, Berlin, 1991.

[2] K. Breitung. Asymptotic approximations for multinormal integrals. *Journal of Engineering Mechanics, ASCE*, 110:357–366, 1984.

[3] A. Der Kiureghian, T. Haukaas, and K. Fujimura. Structural reliability software at the university of California, Berkeley. *Structural Safety*, 28:4467, 2006.

[4] O. Ditlevsen and H.O. Madsen. *Structural Reliability Methods.* Wiley, 1996.

[5] E.N. Dvorkin and K.J. Bathe. A continuum mechanics based four-node shell element for general nonlinear analysis. *Eng Comput*, 1:77–88, 1984.

[6] L. Faravelli. Response surface approach for reliability analysis. *J. of Engng. Mech.*, 115:2763–2781, 1989.

[7] B. Fiessler, H.-J Neumann, and R. Rackwitz. Quadratic limit states in structural reliability. *Journal of Engineering Mechanics, ASCE*, 105:661–676, 1979.

[8] S. Gollwitzer, B. Kirchgäßner, R. Fischer, and R. Rackwitz. PERMAS-RA/STRUREL system of programs for probabilistic reliability analysis. *Structural Safety*, 28:108129, 2006.

[9] G. Johnson and W. Cook. A constitutive model and data for metals subjected to large stains, high strain rates and high temperatures. In *7th Symposium on Ballistics, The Hague*, 1983.

[10] A. Karamchandani, P. Bjerager, and C.A. Cornell. Adaptive importance sampling. In A.H.-S. Ang, M. Shinozuka, and G.I. Schuëller, editors, *Proceedings 5th International Conference on Structural Safety and Reliability, San Francisco.* ASCE, 1989.

[11] M. Lemaire and M. Pendola. Phimeca-Soft. *Structural Safety*, 28:130–149, 2006.

[12] M. Liefvendahl and R. Stocki. A study on algorithms for optimization of Latin hypercubes. *Journal of Statistical Planning and Inference*, 136:3231–3247, 2006.

[13] P.-L. Liu and A. Der Kiureghian. Multivariate distribution models with prescribed marginals and covariances. *Probabilistic Engineering Mechanics*, 1(2):105–112, 1986.

[14] Mecalog SARL, 2 Rue de la Renaissance 92160 Antony, France. *RADIOSS Input Manual, Version 4.2*, 2000.

[15] R.E. Melchers. Simulation in time-invariant and time-variant reliability problems. In R. Rackwitz and P. Thoft-Christensen, editors, *Reliability and Optimization of Structural Systems '91, Proc. 4th IFIP WG 7.5 Conf., Munich, 11–13 September 1991*, pages 39–82. Springer-Verlag, 1992.

[16] R.E. Melchers. *Structural Reliability Analysis and Predictions, 2nd Ed.* Wiley, 1999.

[17] M.F. Pellissetti and G.I. Schueller. On general purpose software in structural reliability an overview. *Structural Safety*, 28:316, 2006.

[18] S. Reh, J.D. Beley, S. Mukherjee, and E.H. Khor. Probabilistic finite element analysis using ANSYS. *Structural Safety*, 28:1743, 2006.

[19] G.I. Schueller. Structural reliability–recent advances. In *In: Proc. 7th ICOSSAR97, Kyoto*, pages 3–33. Balkema, Rotterdam, 1998.

[20] G.I. Schuëller and R. Stix. A critical appraisal of methods to determine failure probabilities. *Structural Safety*, 4:293–309, 1987.

[21] R. Stocki, P. Tauzowski, and M. Kleiber. Efficient sampling techniques for stochastic simulation of structural systems. *Computer Assisted Mechanics and Engineering Sciences*, 14:127–140, 2007.

[22] R. Stocki, P. Tauzowski, and J. Knabel. Reliability analysis of a crashed thin-walled s-rail accounting for random spot weld failures. *International Journal of Crashworthiness*, 13:693–706, 2008.

[23] B Stroustrup. *C++ Programming Language (special Edition)*. Addison Wesley, 2000.

[24] B.H. Thacker, D.S. Riha, L.J. Fitch, S.H.K.; Huyse, and J.B. Pleming. Probabilistic engineering analysis using the NESSUS software. *Structural Safety*, 28:83107, 2006.

[25] Y. Zhang and A. Der Kiureghian. Finite element reliability methods for inelastic structures. Report No. UCB/SEMM-97/05, Department of Civil & Enviromental Engineering, University of California, Berkeley, CA, 1997.

[26] T. Zou, S. Mahadevan, Z.P. Mourelatos, and P. Meernik. Reliability analysis of automotive body–door subsystem. *Reliability Engineering and System Safety*, 78:315–24, 2002.