FERUM 4.1 User's Guide

J.-M. Bourinet

July 1, 2010

Abstract

The development of FERUM (**F**inite **E**lement **R**eliability **U**sing **M**atlab) as an open-source Matlab© toolbox was initiated in 1999 under Armen Der Kiureghian's leadership at the University of California at Berkeley (UCB). This general purpose structural reliability code was developed and maintained by Terje Haukaas, with the contributions of many researchers at UCB.

The present document aims at presenting the main features and capabilities of a new version of this open-source code (FERUM 4.x) based on a work carried out at the Institut Français de Mécanique Avancée (IFMA) in Clermont-Ferrand, France. This new version offers improved capabilities such as simulation-based technique (Subset Simulation), Global Sensitivity Analysis (based on Sobol's indices), Reliability-Based Design Optimization (RBDO) algorithm, Global Sensitivity Analysis and reliability assessment based on Support Vector Machine (SVM) surrogates, etc. Beyond the new methods implemented in this code, an emphasis is put on the new architecture of the code, which now allows distributed computing, either virtually through vectorized calculations within Matlab or for real with multi-processor computers.

An important note about this User's Guide is that it does not contain a detailed description of FERUM usage and FERUM available methods. The user must have some prior knowledge about probability concepts, stochastic methods, structural reliability, sensitivity analysis, etc. More details about the FERUM inputs may be found in the template input file (template_inputfile.m) provided with the FERUM package. This document has been mainly elaborated from FERUM 4.0 reference paper [BMD09].

Contents

1 Introduction

FERUM (**F**inite **E**lement **R**eliability **U**sing **M**atlab) is a general purpose structural reliability code whose first developments started in 1999 at the University of Cafifornia at Berkeley (UCB) [DKHF06]. This code consists of an open-source Matlab© toolbox, featuring various structural reliability methods. As opposed to commercial structural reliability codes, see *e.g.* [PS06] for a review in 2006, the main objective of FERUM is to provide students with a tool immediately comprehensible and easy to use and researchers with a tool very accessible which they may develop for research purposes. The scripting language of Matlab is perfect for such objectives, as it allows users to give commands in a very flexible way, either in an interactive mode or in a batch mode through input files.

FERUM was created under Prof. Armen Der Kiureghian's leadership and was managed by Terje Haukaas at UCB until 2003. It benefited from a prior experience with CalRel structural reliability code, which features all the methods implemented in the last version of FERUM. It also benefited from the works of many researchers at UCB, who made valuable contributions in the latest available version (version 3.1), which can be downloaded at the following address: http://www.ce.berkeley.edu/FERUM/. Since 2003, this code is no longer officially maintained.

This document is a basic user's manual of a new version of this code (FERUM 4.x), which results from a work carried out at the Institut Français de Mécanique Avancée (IFMA) in Clermont-Ferrand. As previously achieved in the past, the main intention is to provide students and researchers with a developer-friendly computational platform which facilitates learning methods and serves as a basis for collaborative research works. FERUM should still be viewed as a development platform for testing new methods and applying them to various challenging engineering problems, either represented by basic analytical models or more elaborated numerical models, through proper user-defined interfaces.

The main architecture of FERUM was preserved in general, see Section 2 for more details. In order to improve its efficiency in terms of computational time, all algorithms have been revisited to extend FERUM capabilities to distributed computing. For example, in its new version, FERUM makes Monte Carlo Simulation (MCS) much faster thanks to limit-state functions defined in a vectorized form or real distributed computing, according that a proper interface is defined for sending multiple jobs to a multi-processor computer platform.

2 Problem definition and structure of FERUM 4.x

This section briefly presents the general formulation of time-invariant structural reliability problems. In addition to some very brief details about theoretical concepts, this section highlights how these concepts are translated to FERUM structure. This includes for instance the stochastic model, the transformation to standard normal variates, limit-state functions and more generally other points regarding computational aspects. It is important here to recall that the main structure of input data in FERUM is preserved compared to version 3.1 (same Matlab structure variables: `probdata`, `analysisopt`, `gfundata`, `femodel`). Changes brought to FERUM are applied to core m-functions and within the fields of the existing structure variables. Similarly to version 3.1, results are stored in structure variables with the following syntax: results keyword appended to the name of the method applied, such as *e.g.* `formresults`, `sormcfhresults`, etc.

2.1 Time invariant structural reliability

We consider here only time invariant structural reliability problems, see *e.g.* [DM07, Lem09]. The probability *w.r.t.* an undesired or unsafe state is expressed in terms of a n-dimensional vector $\mathbf{X}$ of random variables with continuous joint density function $f_{\mathbf{X}}(\mathbf{x}, \boldsymbol{\theta}_f)$, where $\boldsymbol{\theta}_f$ stands for a vector of distribution parameters. Failure is defined in terms of a limit-state function $g(\mathbf{x}, \boldsymbol{\theta}_g)$ where $\mathbf{x}$ is a realization of the random vector $\mathbf{X}$ and $\boldsymbol{\theta}_g$ denotes a vector of deterministic limit-state function parameters. We restrict here the analysis to component reliability with a single g function, but this function may represent multiple failure modes in subset simulation in Section 3.6, without lack of generality. This limit-state function divides the random variable space in a safety domain, $g(\mathbf{x}, \boldsymbol{\theta}_g) > 0$, and a failure domain, $g(\mathbf{x}, \boldsymbol{\theta}_g) \leq 0$. The probability of failure therefore reads:

$$p_f = \int_{g(\mathbf{x}, \boldsymbol{\theta}_g) \leq 0} f_{\mathbf{X}}(\mathbf{x}, \boldsymbol{\theta}_f)\, \mathrm{d}\mathbf{x} \tag{1}$$

2.2 Probability distributions and transformation to standard normal space

The joint density function $f_{\mathbf{X}}(\mathbf{x}, \boldsymbol{\theta}_f)$ is often unknown and replaced by its Nataf counterpart completely defined by specifying marginal distributions and the Gaussian correlation structure between random variables, see [LDK86]. This Nataf joint distribution is completely specified by variables `probdata.marg` and `probdata.correlation` in FERUM input files. FERUM has a rich library of probability distribution models, including extreme value distributions and a truncated normal distribution. These distributions can be specified through either their statistical moments or parameters. See hereafter the corresponding part of the inputfile_template.m file.

```
% Marginal distributions for each random variable
% probdata.marg =  [ (type) (mean) (stdv) (startpoint) (p1) (p2) (p3) (p4) (input_type); ... ];
%
% type: -1 = Parameter in reliability analysis (thetag)
%        0 = Deterministic parameter (cg)
%
%        1 = Normal distribution
%        2 = Lognormal distribution
%        3 = Gamma distribution
%        4 = Shifted exponential distribution
%        5 = Shifted Rayleigh distribution
%        6 = Uniform distribution
%        7 = Beta distribution
%        8 = Chi-square distribution
%
%       11 = Type I largest value distribution ( same as Gumbel distribution )
%       12 = Type I smallest value distribution
%       13 = Type II largest value distribution
%       14 = Type III smallest value distribution
%       15 = Gumbel distribution ( same as type I largest value distribution )
%       16 = Weibull distribution ( same as Type III smallest value distribution with epsilon = 0 )
%
%       18  (Reserved for Laplace distribution)
%       19  (Reserved for Pareto distribution)
%
%       51 = Truncated normal marginal distribution
%
% Notes:
%
%    - Each field type, mean, stdv, startpoint, p1, p2, p3, p4, input_type must be filled in.
%      If not used, input a dummy nan value.
%
%    - input_type = 0 when distributions are defined with mean and stdv only
%                   1 when distributions are defined with distribution parameters pi
%                     (i = 1, 2, 3 or 4, depending on the total number of parameters required)
%
%    - For the Type III smallest value marginal distribution, you must give the value of
%      epsilon parameter as p3 when using the mean and stdv input (input_type = 0).
%    - For the Beta marginal distribution , you must give the value of a parameter as p3 and
%      b parameter as p4 when using the mean and stdv input (input_type = 0).
%    - For the Truncated normal marginal distribution, you must set input_type = 1 and give :
%         * the mean and stdv of the untruncated marginal distribution as p1 and p2 respectively
%         * the lower bound xmin and the upper bound xmax as p3 and p4 respectively
%
%    - startpoint stands for the starting point of the FORM analysis in the physical space
%
%    - Refer to ferum_pdf.m, ferum_cdf.m and distribution_parameter.m functions for more information
%      on a specific distribution.


% Correlation matrix
% This matrix is a square matrix with dimension equal to size(probdata.marg,1).
% Lines/columns corresponding to parameters in reliability analysis(thetag) or deterministic
% parameters (cg) are removed in a pre-processing stage of FERUM, by means of the update_data.m
% function.
```

The structural reliability problem of Equation (1) expressed in the original space of physical random variables $\mathbf{X}$ is transformed to a standard normal space where $\mathbf{U}$ becomes an independent standard normal vector. This mapping is carried out in FERUM using the Nataf model [LDK86]. Physical random variables $\mathbf{X}$ are transformed to correlated standard normal variables $\mathbf{Z}$ whose correlation structure obeys integral relation of Equation (3) and $\mathbf{Z}$ is then transformed to uncorrelated standard normal variables $\mathbf{U}$.

$$\mathbf{X} \text{ with } \left| \begin{array}{c} f_{X_i} \ , \ i = 1, \ldots, n \\ \mathbf{R} = \left[\rho_{ij}\right]_{n \times n} \end{array} \right. \quad \mapsto \quad \mathbf{Z} \sim N\left(\mathbf{0}, \mathbf{R}_0\right) \quad \mapsto \quad \mathbf{U} \sim N\left(\mathbf{0}, \mathbf{I}\right) \tag{2}$$

For each ij-pair of variables with known correlation ρ_{ij}, Equation (3) should be solved to determine correlation $\rho_{0\,ij}$ between mapped z-variables:

$$\rho_{ij} = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \left(\frac{x_i - \mu_i}{\sigma_i} \right) \left(\frac{x_j - \mu_j}{\sigma_j} \right) \varphi_2(z_i, z_j, \rho_{0\,ij})\, \mathrm{d}z_i \mathrm{d}z_j \tag{3}$$

where μ_i and σ_i respectively stand for the mean and standard deviation of the ith component of $\mathbf{X}$, and $\varphi_2(\bullet, \bullet, \rho)$ is the 2D standard normal probability density function (pdf) with correlation coefficient ρ.

Independent standard normal variables $\mathbf{U}$ are then obtained from $\mathbf{Z}$ variables such as follows:

$$\mathbf{u} = \mathbf{L}_0^{-1}\mathbf{z} \tag{4}$$

where $\mathbf{L}_0$ is the lower-triangular Cholesky decomposition of $\mathbf{R}_0 = \left[\rho_{0\,ij}\right]$ matrix, such that $\mathbf{L}_0\mathbf{L}_0^{\mathrm{T}} = \mathbf{R}_0$.

Solutions of Equation (3) can be found by formulae of reference [LDK86] for most common statistical distributions. These formulae are most often obtained by least-square fitting and therefore approximate, except for a few pairs of distributions. FERUM 4.x is now based on accurate solutions obtained by 2D numerical Gauss integration of Equation (3). A particular attention is paid to strongly correlated random variables, where the number of integration points along each dimension in $z_i z_j$-space must be selected carefully, for accurate $\rho_{0\,ij}$ values. A practical rule adopted here consists in increasing these numbers of points with correlation, ranging from 32 points along each dimension for absolute values of correlation lower than 0.9 to 1024 points for absolute values larger than 0.9995.

2.3 Definition of limit-state functions

As in FERUM 3.1, the limit-state function is defined through the structure variable `gfundata` of the input file and called by the file named gfun.m. Various strategies are now offered in FERUM 4.x. The limit-state function can either be a simple expression directly written in the input file or a Matlab function. For both cases, gfun.m calls another function called gfunbasic.m. Another interesting option offered in FERUM 4.x is that the limit-state function can be defined through a user-provided Matlab function, which calls a third-party software, such as a Finite Element code. Such merging of FERUM with problem-specific external codes was made in various applications, such as probabilistic buckling [DNBF09] and crack propagation [NBGL06]. For controlling such external codes, extra variables are provided to FERUM through the structure variable `femodel` and the user must create an application-specific function. One more option available in FERUM 4.x is that it takes advantage of gradients *w.r.t.* all or parts of the basic variables, when available from the external code. This proves to be very useful when limit-state functions involve very computationally demanding numerical models, as it avoids tedious estimations by finite differences. FERUM 4.x is compatible with Code_Aster©, running on a Windows OS. Code_Aster version STA9.1 compiled on Windows was developed by NECS and is available at http://www.necs.fr/gb/telechargement.php.

See hereafter the corresponding part of the inputfile_template.m file.

```
% Type of limit-state function evaluator:
% 'basic': the limit-state function is defined by means of an analytical expression or a Matlab
%          m-function, using gfundata(lsf).expression. The function gfun.m calls gfunbasic.m,
%          which evaluates gfundata(lsf).expression.
% 'xxx':   the limit-state function evaluation requires a call to an external code.  The function
%          gfun.m calls gfunxxx.m, which evaluates gfundata(lsf).expression where gext variable is
%          a result of the external code.
```

Case of a limit-state function which is defined by means of an analytical expression or a Matlab m-function:

```
gfundata(1).evaluator  = 'basic';
gfundata(1).type       = 'expression';   % Do no change this field!

% Expression of the limit-state function:
gfundata(1).expression = 'c1 - X2./(1000*X3) - (X1./(200*X3)).^2 - X5./(1000*X6) - (X4./(200*X6)).^2';

% Expression of the limit-state function:
gfundata(1).expression = 'gfun_nl_oscillator(mp,ms,kp,ks,zetap,zetas,Fs,S0)';
```

Case of a limit-state function which requires a call to Code_Aster FE code:

```
gfundata(1).evaluator  = 'aster';
gfundata(1).type       = 'expression';   % Do no change this field!

% Expression of the limit-state function:
gfundata(1).expression = 'gext-u0';
```

2.4 Vectorized / distributed computing

A major change brought to FERUM 4.x is that calls to the limit-state function g can be evaluated in a distributive manner, as opposed to the sequential manner of the previous version. Every algorithm implemented in FERUM was revisited, so as to send multiple calls to g, whenever possible. If one thinks of FE-based Monte Carlo Simulation (MCS) on a multiprocessor computer, the strategy consists in sending calls to the FE code in batches, the number of jobs in each batch being equal to the number of available CPUs. This strategy is known as distributed computing, see *e.g.* examples of applications in [NBGL06, DNBF09]. The number of jobs sent simultaneously is tuned through the variable `analysisopt.block_size`. Such an option is available in FERUM, assuming that the user has a suitable computer platform and all the necessary tools to create, send and post-process multiple jobs (scripting language such as Perl, queuing systems such as OpenPBS on Linux, job schedulers, ...). The function in charge of the job allocation is obviously application-specific and is called by gfun.m.

```
analysisopt.multi_proc = 1;     % 1: block_size g-calls sent simultaneously
                                %    - gfunbasic.m is used and a vectorized version of
                                %      gfundata.expression is available.
                                %      The number of g-calls sent simultaneously (block_size) depends
                                %      on the memory available on the computer running FERUM.
                                %    - gfunxxx.m user-specific g-function is used and able to handle
                                %      block_size computations sent simultaneously, on a cluster
                                %      of PCs or any other multiprocessor computer platform.
                                % 0: g-calls sent sequentially

analysisopt.block_size = 50;    % Number of g-calls to be sent simultaneously
```

Based on the same developments of FERUM algorithms, it is also possible to send multiple calls to a user-defined Matlab limit-state function written in a vectorized manner. Vectorized calculations, in the Matlab sense, eliminate the need to cycle through nested loops and thus run much faster because of the way Matlab handles vectors internally. The principle is similar to distributed computing, the difference being that the multiprocessor computer is virtually replaced by a single computer which can handle a number of runs simultaneously (this maximum number being directly dependent on the memory available on the computer). Here again, the maximum number of runs sent simultaneously is controlled through `analysisopt.block_size` variable.

For illustration purpose, on an Intel T7800 2.6GHz dual core CPU with 4Gb RAM, a crude MCS takes 31 min with $1.5 \cdot 10^9$ samples for a basic $g = r - s$ problem, where R and S are normal random variables, in a vectorized manner (FERUM 4.x), as opposed to 6 days 15 hours in a sequential manner.

3 Overview of available methods

3.1 FORM and reliability sensitivities / importance measures

3.1.1 Basic FORM

First-Order Reliability Method (FORM) (option 10 of FERUM 4.x) aims at using a first-order approximation of the limit-state function in the standard space at the so-called Most Probable Point (MPP) of failure P^* (or design point), which is the limit-state surface closest point to the origin. Finding the coordinates $\mathbf{u}^*$ of the MPP consists in solving the following constrained optimization problem:

$$\mathbf{u}^* = \arg\min \left\{ \|\mathbf{u}\| \;\middle|\; g(\mathbf{x}(\mathbf{u}), \boldsymbol{\theta}_g) = G\left(\mathbf{u}, \boldsymbol{\theta}_g\right) = 0 \right\} \tag{5}$$

Once the MPP P^* is obtained, the Hasofer and Lind reliability index β is computed as $\beta = \boldsymbol{\alpha}^{\mathrm{T}}\mathbf{u}^*$ where $\boldsymbol{\alpha} = -\nabla_{\mathbf{u}} G(\mathbf{u}^*) / \left\|\nabla_{\mathbf{u}} G(\mathbf{u}^*)\right\|$ is the negative normalized gradient vector at the MPP P^*. It represents the distance from the origin to the MPP in the standard space. The first-order approximation of the failure probability is then given by $p_{f1} = \Phi(-\beta)$, where $\Phi(\bullet)$ is the standard normal cdf. As in FERUM 3.1, the new version is based on the iHLRF algorithm, see [ZDK94] for further details. In order to take advantage of distributed computing, g-calls required for gradient evaluations by finite differences at a specific point of the standard space are sent in a single batch. The same technique is applied to step size evaluation with Armijo rule, where all corresponding g-calls are sent simultaneously.

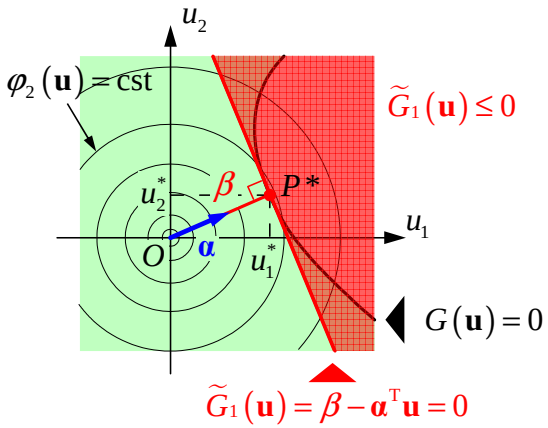


Figure 1: First-Order Reliability Method (FORM).

Parameters specific to FORM in the `analysisopt.block_size` structure variable are listed hereafter.

```
% FORM analysis options
analysisopt.i_max      = 100;      % Maximum number of iterations allowed in the search algorithm
analysisopt.e1         = 0.001;    % Tolerance on how close design point is to limit-state surface
analysisopt.e2         = 0.001;    % Tolerance on how accurately the gradient points towards
                                   % the origin
analysisopt.step_code  = 0;        % 0: step size by Armijo rule
                                   % otherwise: given value is the step size
analysisopt.Recorded_u = 1;        % 0: u-vector not recorded at all iterations
                                   % 1: u-vector recorded at all iterations
analysisopt.Recorded_x = 1;        % 0: x-vector not recorded at all iterations
                                   % 1: x-vector recorded at all iterations

% FORM, SORM analysis options
analysisopt.grad_flag  = 'ddm';    % 'ddm': direct differentiation
                                   % 'ffd': forward finite difference
analysisopt.ffdpara    = 1000;     % Parameter for computation of FFD estimates of gradients
                                   % Perturbation = stdv/analysisopt.ffdpara;
                                   % Recommended values: 1000 for basic limit-state functions,
                                   % 50 for FE-based limit-state functions
```

3.1.2 Reliability sensitivities / importance measures

In addition to the reliability index β and the MPP coordinates coming from a FORM analysis, the user may use FERUM 4.x to calculate the sensitivities of β (or of the failure probability p_f) to distribution parameters $\boldsymbol{\theta}_f$ or to deterministic limit-state function parameters $\boldsymbol{\theta}_g$.

For instance, the sensitivity of β *w.r.t.* $\boldsymbol{\theta}_f$ reads:

$$\nabla_{\boldsymbol{\theta}_f}\beta = \mathbf{J}_{\mathbf{u}^*,\boldsymbol{\theta}_f}\left(\mathbf{x}^*,\boldsymbol{\theta}_f\right)^{\mathrm{T}}\boldsymbol{\alpha} \tag{6}$$

where $\mathbf{J}_{\mathbf{u}^*,\boldsymbol{\theta}_f}\left(\mathbf{x}^*,\boldsymbol{\theta}_f\right) = \left[\partial u_i/\partial\theta_{f\,j}\right]$.

The Jacobian of the transformation is obtained by differentiating Equation (4) *w.r.t.* $\boldsymbol{\theta}_f$ parameters:

$$\frac{\partial\mathbf{u}}{\partial\theta_f} = \mathbf{L}_0^{-1}\frac{\partial\mathbf{z}}{\partial\theta_f} + \frac{\partial\mathbf{L}_0^{-1}}{\partial\theta_f}\mathbf{z} \tag{7}$$

In FERUM 4.x, sensitivities *w.r.t.* distributions parameters $\boldsymbol{\theta}_f$ are evaluated based on both terms of Equation (7), as opposed to FERUM 3.1 which only uses the first term. Sensitivities to correlation are based on the second term of this expression only, as the first one vanishes [BL08]. Sensitivities are evaluated numerically with the same integration scheme as the one used for obtaining $\mathbf{R}_0$ matrix and it is required to differentiate the Cholesky decomposition algorithm in a step-by-step manner. Examples of application are given in reference [BL08].

Parameters specific to FORM reliability sensitivities are listed hereafter. Thet are set up in the `probdata` structure variable for sensitivities *w.r.t.* distributions parameters $\boldsymbol{\theta}_f$ and in the `gfundata` structure variable for sensitivities *w.r.t.* deterministic limit-state function parameters $\boldsymbol{\theta}_g$.

```
% Flag for computation of sensitivities w.r.t. means, standard deviations, parameters and
% correlation coefficients
% 1: all sensitivities assessed, 0: no sensitivities assessment
probdata.flag_sens    = 1;

% Flag for computation of sensitivities w.r.t. thetag parameters of the limit-state function
% 1: all sensitivities assessed, 0: no sensitivities assessment
gfundata(1).flag_sens  = 1;

analysisopt.ffdpara_thetag = 1000;  % Parameter for computation of FFD estimates of dbeta_dthetag
                                    % perturbation = thetag/analysisopt.ffdpara_thetag if thetag ~= 0
                                    %              or 1/analysisopt.ffdpara_thetag if thetag == 0
                                    % Recommended values: 1000 for basic limit-state functions
                                    %                     100 for FE-based limit-state functions
```

3.1.3 FORM with search for multiple design points

Search for multiple MPPs such as described in [DKD98] is also implemented in FERUM 4.x (option 11). Figure 2 illustrates the use of this method, applied to a 2D example with a parabolic limit-state function [DKD98]:

$$g(\mathbf{x}) = g(x_1, x_2) = b - x_2 - \kappa(x_1 - e)^2 \tag{8}$$

where $b = 5$, $\kappa = 0.5$ and $e = 0.1$. Both variables x_1 and x_2 are independent and identically distributed (i.i.d.) standard normal random variables.

This problem is characterized by two MPPs at similar distances from the origin and basic FORM algorithm results are therefore not valid: $1.83 \cdot 10^{-3}$ instead of $3.02 \cdot 10^{-3}$ reference value for p_f. Results in Figure 2 are obtained with parameter values recommended in [DKD98], *i.e.* $\gamma = 1.1$, $\delta = 0.75$ and $\epsilon = 0.5$. These parameters are set in the form_multiple_dspts.m function. All valid MPP results obtained with FERUM 4.x are stored in an `ALLformresults` array which is added as an extra field to the `analysisopt` structure variable.

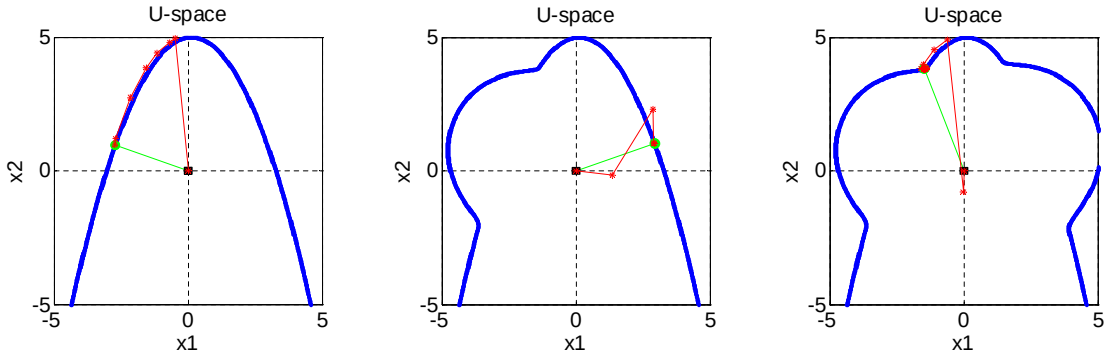


Figure 2: FORM with search for multiple design points.

3.2 SORM curvature-fitting and point-fitting

As in the previous version, FERUM 4.x offers two ways for computing a second-order approximation of the failure probability. In both methods, the SORM approximation of the failure probability p_{f2} is computed with Breitung or Tvedt formulae, as in FERUM 3.1.

The first method consists in determining the principal curvatures and directions, by solving an eigenproblem involving the Hessian of the limit-state function (option 12 of FERUM 4.x). The Hessian is computed by finite differences, the perturbations being set in the standard normal space. All calls to the limit-state function corresponding to perturbated points are potentially sent simultaneously, as being all independent from each other.

The second method consists in approximating the limit-state function by a piece-wise paraboloid surface [DKLH87] (option 13 of FERUM 4.x). This approximate surface must be tangent to the limit-state at the design point and coincides with the limit-state at two points on each axis selected on both sides of the origin, see Figure 4. It is built iteratively, with a limited number of iterations and all calls to the limit-state function, at each iteration, are potentially sent simultaneously as well. This second approach is advantageous for slightly-noisy limit-state functions (*e.g.* when a FE code is involved), for problem with a large number of random variables or when the computation of curvatures turns out to be problematic.

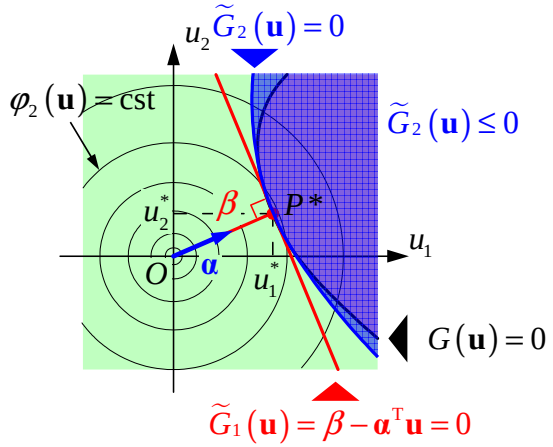


Figure 3: Second-Order Reliability Method (SORM).

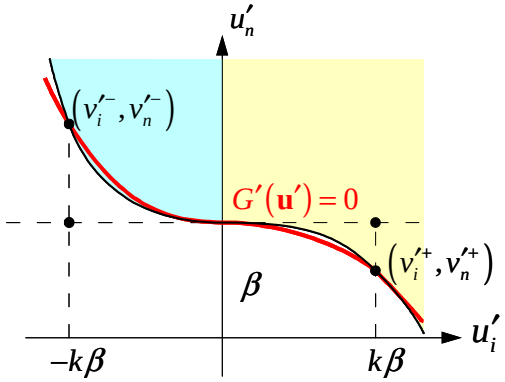


Figure 4: Second-Order Reliability Method - Point-fitting method (SORM-pf).

3.3 Distribution Analysis

FERUM 4.x offers a way to assess qualitatively (histograms) and quantitatively to some extent (only mean and variance of the model outputs in the current version) the distribution of the limit-state function g (option 20 of FERUM 4.x). This limit-state function may either return a scalar value if `gfundata(lsf).ng` field is set to 1 in the input file or a vector of values if `gfundata(lsf).ng` is set to a value strictly greater than 1 (please refer to the gfun.m function for more details).

Parameters specific to a distribution analysis are listed hereafter.

```
% Simulation analysis (MC,IS,DS,SS) and distribution analysis options
analysisopt.num_sim              = 1e4;      % Number of samples
analysisopt.rand_generator       = 1;        % 0: default rand matlab function
                                             % 1: Mersenne Twister (to be preferred)

% Simulation analysis (MC, IS) and distribution analysis options
analysisopt.sim_point            = 'origin'; % 'dspt': design point
                                             % 'origin': origin in standard normal space
analysisopt.stdv_sim             = 1;        % Standard deviation of sampling distribution
                                             % in the simulation analysis
```

3.4 Crude Monte Carlo Simulation, Importance Sampling

Equation (1) is rewritten as follows:

$$p_f = \int_{D_{f_{\mathbf{X}}}} I(\mathbf{x})\, f_{\mathbf{X}}(\mathbf{x})\, \mathrm{d}\mathbf{x} = E_{f_{\mathbf{X}}}[I(\mathbf{X})] \tag{9}$$

where $D_{f_{\mathbf{X}}}$ represents the integration domain of joint pdf $f_{\mathbf{X}}(\mathbf{x})$, $I(\bullet)$ is an indicator function which equals 1 if $g(\mathbf{x}) \le 0$ and 0 otherwise, and $E_{f_{\mathbf{X}}}[\bullet]$ denotes the mathematical expectation *w.r.t.* joint pdf $f_{\mathbf{X}}(\mathbf{x})$ ($\boldsymbol{\theta}_f$ and $\boldsymbol{\theta}_g$ parameters omitted for the sake of clarity).

The expectation in Equation (9) is estimated in a statistical sense for a crude Monte Carlo Simulation (crude MCS). The u-space is randomly sampled with N independent samples $\mathbf{u}^{(j)}$, $j = 1, \ldots, N$. These N samples are then transformed to the x-space $\mathbf{x}^{(j)} = \mathbf{x}(\mathbf{u}^{(j)})$ and an unbiased estimate of p_f is finally obtained from the sample mean of $q_j = I(\mathbf{x}^{(j)})$. Note that a standard deviation is also obtained for this sample, providing useful information regarding the accuracy of the estimated value of p_f.

It must be stressed out here that a crude MCS requires a high computational effort (large N) for small failure probabilities and a number of variance reduction techniques have been proposed in the past to lower this computational effort. One of these variance reduction techniques is known as Importance Sampling (IS). Equation (9) is rewritten now in the following form:

$$p_f = \int_{D_h} I(\mathbf{x}) \, \frac{f_{\mathbf{X}}(\mathbf{x})}{h(\mathbf{x})} \, h(\mathbf{x}) \, \mathrm{d}\mathbf{x} = E_h \left[I(\mathbf{X}) \, \frac{f_{\mathbf{X}}(\mathbf{X})}{h(\mathbf{X})} \right] \tag{10}$$

where h is called a sampling density. For an IS analysis, it is usual to take $h(\mathbf{x}) = h(\mathbf{x}(\mathbf{u})) = \varphi_n(\mathbf{u} - \mathbf{u}^*)$ where φ_n is the n-dimensional standard normal pdf and $\mathbf{u}^*$ is the vector of MPP coordinates coming from a previous FORM analysis. Note that p_f is now obtained from the sample mean of $q_j = I(\mathbf{x}^{(j)}) \, f_{\mathbf{X}}(\mathbf{x}^{(j)}) \, / \, h(\mathbf{x}^{(j)})$.

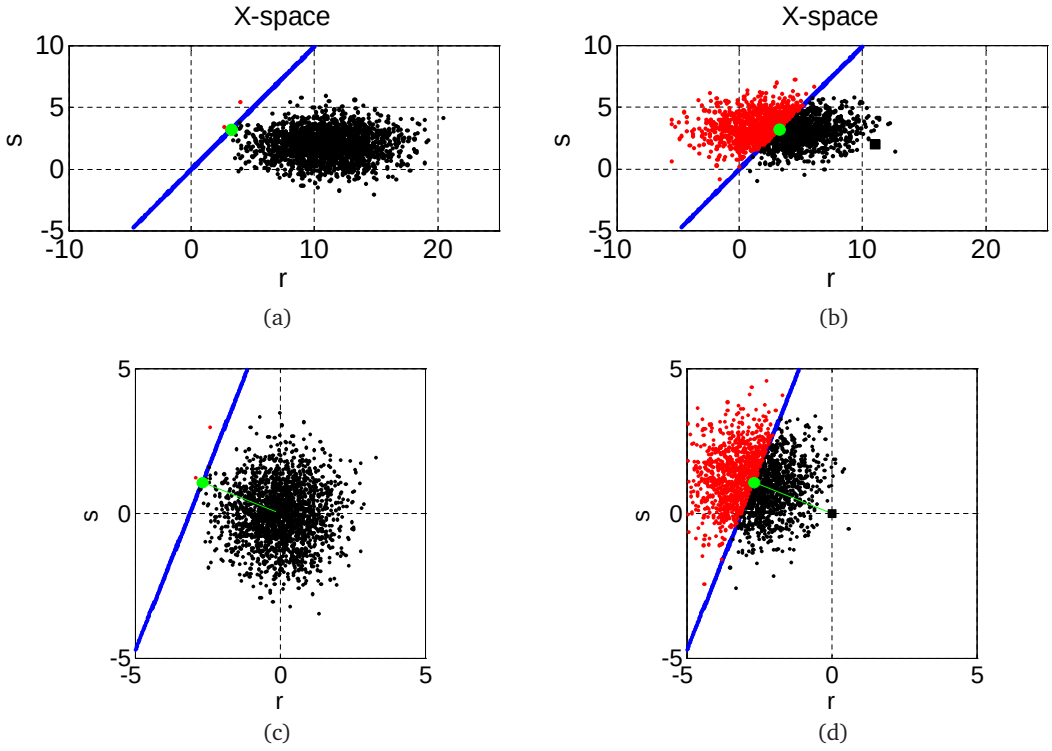


Figure 5: Crude Monte Carlo (CMC) *vs.* Importance Sampling (IS).

FERUM 4.x features both methods and calls to the limit-state function are sent in a distributed manner, the maximum number of jobs sent being adjusted by the variable `analysisopt.block_size`. For a crude MCS analysis in FERUM 4.x, set `analysisopt.sim_point` to `'origin'` and select option 21. For an IS analysis, set `analysisopt.sim_point` to `'dspt'` and select option 21.

Parameters specific to a crude MCS / IS analysis are listed hereafter.

```
% Simulation analysis (MC,IS,DS,SS) and distribution analysis options
analysisopt.num_sim              = 1e4;      % Number of samples
analysisopt.rand_generator       = 1;        % 0: default rand matlab function
                                             % 1: Mersenne Twister (to be preferred)

% Simulation analysis (MC, IS) and distribution analysis options
analysisopt.sim_point            = 'origin'; % 'dspt': design point
                                             % 'origin': origin in standard normal space
analysisopt.stdv_sim             = 1;        % Standard deviation of sampling distribution
                                             % in the simulation analysis
```

```
% Simulation analysis (MC, IS)
analysisopt.target_cov          = 0.05;     % Target coef. of variation for failure probability
analysisopt.lowRAM              = 0;        % 1: memory savings allowed
                                            % 0: no memory savings allowed
```

3.5 Directional Simulation

The n-dimensional normal vector $\mathbf{U}$ is expressed as $\mathbf{U} = R\mathbf{A}$, $R \geq 0$, where R^2 is a chi-square distributed random variable with n degrees of freedom (d.o.f.), independent of the random unit vector $\mathbf{A}$, which is uniformly distributed on the n-dimensional unit sphere Ω^n. The failure probability p_f can be written as follows, conditioning on $\mathbf{A} = \mathbf{a}$ [Bje88]:

$$p_f = \int_{\mathbf{a}\in\Omega^n} P\left[G(R\mathbf{A}) \leq 0 \mid \mathbf{A} = \mathbf{a}\right] f_{\mathbf{A}}(\mathbf{a})\, d\mathbf{a} \tag{11}$$

where $f_{\mathbf{A}}(\mathbf{a})$ is the uniform density of $\mathbf{A}$ on the unit sphere.

Practically, a sequence of N random direction vectors $\mathbf{a}^{(j)} = \mathbf{u}^{(j)} \,/\, \left\|\mathbf{u}^{(j)}\right\|$, $j = 1,\ldots,N$, is generated first, then $r_j = \{\, r \mid G(r\mathbf{a}^{(j)}) = 0 \,\}$ are found iteratively and p_f is finally estimated from the following expression:

$$p_f = \frac{1}{N}\sum_{j=1}^{N}\left[1 - \chi_n^2\left(r_j^2\right)\right] \tag{12}$$

where χ_n^2 is the chi-square cdf with n d.o.f..

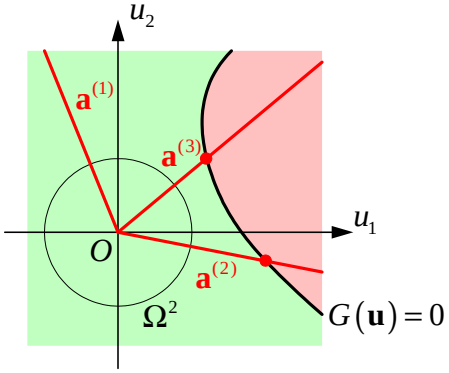


Figure 6: Directional Simulation (DS): Uniform distribution on the 2D unit sphere

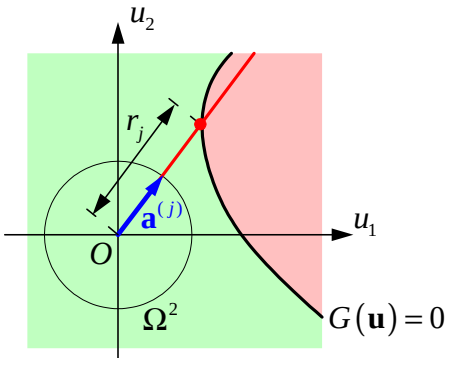


Figure 7: Directional Simulation (DS): Determination of r_j radius

In FERUM 4.x, the user must select option 22 for a Directional Simulation (DS) analysis. Beside a classical random generation of directions (`analysisopt.dir_flag` set to `'random'`), a slightly modified version of this algorithm is proposed. Instead of generating random directions on the unit sphere, it is proposed to divide it into N evenly distributed points, in a deterministic manner, in order to gain an improved accuracy at a given computational cost. This second option is selected by setting `analysisopt.dir_flag` to `'det'`. In the two methods, intersections with the limit-state function along each direction are obtained in a distributive manner, based on a vectorized version of fzero.m Matlab function. The maximum number of jobs sent is adjusted by the variable `analysisopt.block_size`. It is worth noting that DS looses efficiency as the number of random variables n increases.

Parameters specific to a DS analysis are listed hereafter.

```
% Simulation analysis (MC,IS,DS,SS) and distribution analysis options
analysisopt.num_sim              = 200;      % Number of directions (DS)
analysisopt.rand_generator       = 1;        % 0: default rand matlab function
                                             % 1: Mersenne Twister (to be preferred)

% Directional Simulation (DS) analysis options
analysisopt.dir_flag             = 'det';    % 'det': deterministic points uniformly distributed
                                             %        on the unit hypersphere using eq_point_set.m
                                             %        function
                                             % 'random': random points uniformly distributed
                                             %           on the unit hypersphere
analysisopt.rho                  = 8;        % Max search radius in standard normal space for
                                             % Directional Simulation analysis
analysisopt.tolx                 = 1e-5;     % Tolerance for searching zeros of g function
analysisopt.keep_a               = 0;        % Flag for storage of a-values which gives axes
                                             % along which simulations are carried out
analysisopt.keep_r               = 0;        % Flag for storage of r-values for which g(r) = 0
```

3.6 Subset Simulation

Starting from the premise that the failure event $F = \{\, g(\mathbf{x}, \boldsymbol{\theta}_g) \leq 0 \,\}$ is a rare event, S.-K. Au and J.L. Beck proposed to estimate $P(F)$ by means of more frequent intermediate conditional failure events F_i, $i = 1, \ldots, m$ (called subsets) so that $F_1 \supset F_2 \supset \ldots \supset F_m = F$ [AB01]. The m-sequence of intermediate conditional failure events is selected so that $F_i = \{\, g(\mathbf{x}, \boldsymbol{\theta}_g) \leq y_i \,\}$, where y_i's are decreasing values of the limit-state function and $y_m = 0$. As a result, the failure probability $p_f = P(F)$ is expressed as a product of the following m conditional probabilities:

$$p_f = P(F) = P(F_m) = P(F_m \mid F_{m-1}) P(F_{m-1}) = \ldots = P(F_1) \prod_{i=2}^{m} P(F_i \mid F_{i-1}) \tag{13}$$

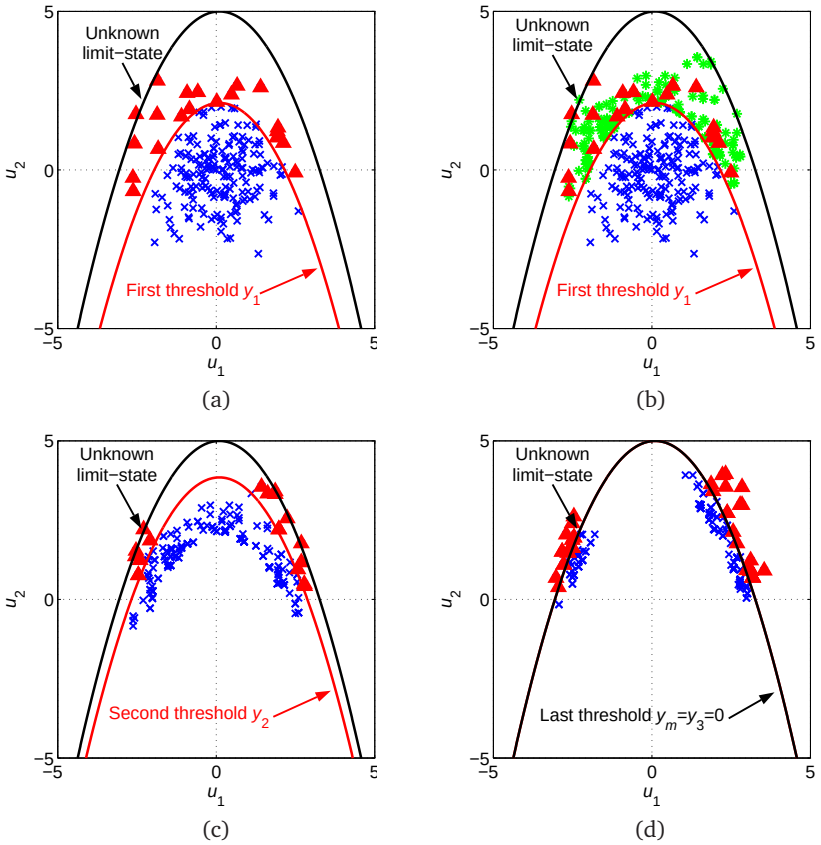


Figure 8: Main steps in Subset Simulation.

Each subset event F_i (and the related threshold value y_i) is determined so that its corresponding conditional probability equals a sufficiently large value α, in order to be efficiently estimated with a rather small number of samples (in practice $\alpha \approx 0.1$-0.2, set by `analysisopt.pf_target` parameter). In essence, there is a trade-off between minimizing the number m of subsets by choosing rather small intermediate conditional probabilities and maximizing the same probabilities so that they can be estimated efficiently by simulations. The first threshold y_1 is obtained by a crude MCS, so that $P(F_1) \approx \alpha$ (see Figure 8, subplot (a)). For further thresholds, new sampling points corresponding to $\{ F_i \mid F_{i-1} \}$ conditional events are obtained from Markov Chains Monte Carlo (MCMC), based on a modified Metropolis-Hastings algorithm, see green star points in figure 8, subplot (b) corresponding to $i = 2$ case. The process is repeated until a negative threshold y_i is found. This is therefore the last step ($i = m$) and y_m is set to 0. The corresponding probability $P(F_m \mid F_{m-1})$ is then evaluated. See Figure 8, subplot (d). The last step is reached for $m = i = 3$ in the present case. The coefficient of variation of the failure probability estimated from SS can be evaluated by intermediate coefficients of variation, weighted by the correlation that exists between the samples used for the estimation of intermediate conditional probabilities, please refer to reference [AB01] for more details.

For a Subset Simulation analysis, select option 23 in FERUM 4.x. Parameters specific to this method are listed hereafter.

```
% Simulation analysis (MC,IS,DS,SS) and distribution analysis options
analysisopt.num_sim              = 10000;    % Number of samples per subset step (SS)
analysisopt.rand_generator       = 1;        % 0: default rand matlab function
                                             % 1: Mersenne Twister (to be preferred)

% Subset Simulation (SS) analysis options
analysisopt.width                = 2;        % Width of the proposal uniform pdfs
analysisopt.pf_target            = 0.1;      % Target probability for each subset step
analysisopt.flag_cov_pf_bounds   = 1;        % 1: calculate upper and lower bounds of the
                                             %    coefficient of variation of pf
                                             % 0: no calculation
analysisopt.ss_restart_from_step = -inf;     % i>=0 : restart from step i
                                             % -inf : all steps, no record (default)
                                             % -1   : all steps, record all
analysisopt.flag_plot            = 0;        % 1: plots at each step (2 r.v. examples only)
                                             % 0: no plots
analysisopt.flag_plot_gen        = 0;        % 1: intermediate plots for each MCMC chain
                                             %    (2 r.v. examples only)
                                             % 0: no plots
```

3.7 ^{2}SMART

Basic random variables realizations are classified in two classes, either they fall in the safety domain or in the failure domain. The ^{2}SMART method hinges on the use of Support Vector Machines (SVMs) [Vap95, STC00, SS02] for classifying training points (called also learning points). The limit-state $g(\mathbf{x}) = 0$ must be seen here as the perfect classifier. We assume that such a classifier exists, whatever the complexity of its geometry. The ^{2}SMART method starts from the pionneering idea of J.E. Hurtado's work, who first introduced SVMs to reliability analysis [Hur04]. The key idea of ^{2}SMART is to build a SVM classifier for each Subset Simulation-like y_i-threshold intermediate limit-state, which turns out to be more easy to manage than tackling $g(\mathbf{x}) = 0$ directly. This learning process takes place in the standard space. The surrogate to each intermediate limit-state $g(\mathbf{x}) - y_i = 0$ is constructed by training on a set of initial learning points and a few new points added in an iterative manner (concept of active learning [Mac92]).
Table 1 gives an overview of the ^{2}SMART algorithm. More details about this method are available in a paper submitted to Structural Safety [BDL10].

1. An initial set of $(N_u + \overbrace{N_n})$ samples is used for the training of a first SVM-y_i classifier (Table 1, part 1), noted $\widetilde{g}_{y_i}(\mathbf{x}(\mathbf{u})) = g(\mathbf{x}(\mathbf{u})) - y_i$, as an approximation to the exact intermediate limit-state $g_{y_i}(\mathbf{x}(\mathbf{u})) = g(\mathbf{x}(\mathbf{u})) - y_i = 0$. Some of these samples (N_n samples) are used for determining a suitable threshold value y_i corresponding to a target probability expected to be close to $\alpha = 0.1$, similarly to Subset Simulation.
2. The strategy then consists in improving the initial SVM-y_i classifier, by adding new points to the learning database in an iterative manner (Table 1, part 2). These updates of the current SVM-y_i classifier hinge on a coarse to fine adjustment strategy (localization, stabilization and convergence stages, see Figure 9). The number of new points added at each iteration is increased with the number of random variables n. By a wise selection of the "most informative" new training samples, drastic improvements of the accuracy of the SVM classifier are obtained, with a rather limited amount of new samples.

3. After a few iterations noted $N_{\text{iter max}}$, the SVM-y_i classifier is expected to become accurate enough and we can evaluate $P(\tilde{F}_1)$ corresponding to y_1-threshold value at the first step of the algorithm ($i = 1$) or $P(\tilde{F}_i \mid \tilde{F}_{i-1})$ corresponding to y_i-threshold value at next steps ($i > 1$) (Table 1, part 3). For $i = 1, \ldots, m$, $\tilde{F}_i$ stands for the SVM approximation of F_i exact intermediate failure domain. For all steps except the last, it is required to keep points which satisfy $\widetilde{g}_{y_i}(\mathbf{x}(\mathbf{u})) \leq 0$ (g-values lower or equal than y_i based on the current SVM-y_i classifier) and which correspond to various numbers of points involved at each stage (N_1, N_2, N_3 for localization, stabilization and convergence stages respectively). These points called germs serve for generating the sampling population of the next step.
4. The P_f estimate is given by the following expression: $P_f = P(\tilde{F}_1)\prod_{i=2}^{m} P(\tilde{F}_i \mid \tilde{F}_{i-1})$.

Table 1: General flowchart of the ^{2}SMART algorithm.

$i = 0$, $y_0 = +\infty$
while $y_i > 0$ **[Loop on each intermediate threshold value** y_i **]**
$i = i + 1$
1. First set of training points → y_i-threshold. if $y_i < 0$, set $y_i = 0$ → Initial training of SVM-y_i classifier
for $k = 1$ **to** $N_{\text{iter max}}$ **[Update of the SVM classifier at each iteration** k **]**
2. Coarse to fine adjustment strategy for work populations: (1) Localization-L, (2) Stabilization-S, (3) Convergence-C (see Figure 9) → New points added to the learning database → New training of SVM-y_i classifier
end for k
3. Evaluation of $P(\tilde{F}_1)$ for $i = 1$ or $P(\tilde{F}_i \mid \tilde{F}_{i-1})$ for $i > 1$ Storage of $\approx \alpha N_1$, $\approx \alpha N_2$ and $\approx \alpha N_3$ points (called germs) for further applications of λr_j or r_j-mM algorithm if $y_i > 0$
end while
$m = i$
4. Evaluation of P_f estimate: $P_f = P(\tilde{F}_1)\prod_{i=2}^{m} P(\tilde{F}_i \mid \tilde{F}_{i-1})$

For a ^{2}SMART simulation, select option 33 in FERUM 4.x. Parameters specific to this method are listed hereafter.

```
analysisopt.num_sim_sdu  = [ 50 50 ];     % Nu number of points
                                          %( [ first subset-like step, next steps], same values by default )
analysisopt.num_sim_norm = [ 100 100 ];   % Nn number of points
                                          %( [ first subset-like step, next steps], same values by default )
analysisopt.svm_buf_size = 3500^2;        % RAM-size dependent parameter for eval_svm.m function
analysisopt.flag_var_rbf = 0;             % 0: cross validation at the 1st subset-like step only
                                          % 1: cross validation at all steps
```

3.8 Global Sensitivity Analysis

Global Sensitivity Analysis aims at quantifying the impact of the variability in each (or group of) input variates on the variability of the output of a model in apportioning the output model variance to the variance in the input variates. Sobol' indices [Sob07] are the most usual global sensitivity measures. They can be assessed in FERUM 4.x.

We consider here a model given by:

$$Y = g(\mathbf{X}) = g(X_1, X_2, \ldots, X_n) \tag{14}$$

where $\mathbf{X} = (X_1, X_2, \ldots, X_n)$ is a vector of n **independent** random input variates, g is a deterministic model and Y is a scalar random output.

In order to determine the importance of each input variate, we consider how the variance of the output Y decreases when variate X_i is fixed to a given x_i^* value:

$$V\left(Y \mid X_i = x_i{}^*\right) \tag{15}$$

where $V(\bullet)$ denotes the variance function.

Since $x_i{}^*$ value is unknown, we take the expectation of Equation (15) and, by virtue of the law of total variance, we can write:

$$V\left(E\left[Y \mid X_i\right]\right) = V(Y) - E\left[V\left(Y \mid X_i\right)\right] \tag{16}$$

The global sensitivity index of the first-order is defined as follows, for $i = 1, \ldots, n$:

$$S_i = \frac{V\left(E\left[Y \mid X_i\right]\right)}{V(Y)} = \frac{V_i}{V} \tag{17}$$

Indices of higher orders are defined in a similar manner, *e.g.* for the second-order:

$$S_{ij} = \frac{V\left(E\left[Y \mid X_i, X_j\right]\right) - V_i - V_j}{V(Y)} = \frac{V_{ij}}{V} \tag{18}$$

First-order indices inform about the influence of each variate taken alone whereas higher order indices account for possible influences between various parameters. Total sensitivity indices are also usually introduced. They express the total sensitivity of Y variance to X_i input, including all interactions that involve X_i:

$$S_{T_i} = \sum_{i \subset k} S_k \tag{19}$$

where $i \subset k$ denotes the set of indices containing i.

From a computational viewpoint, Sobol' indices can be assessed using a Crude Monte Carlo (CMC) or a Quasi-Monte Carlo (QMC) sampling procedure (`analysisopt.sampling` respectively set to 1 or 2). This latter technique is based on low-discrepancy sequences, which usually outperform CMC simulations in terms of accuracy, at a given computational cost. Both CMC and QMC sampling procedure are implemented in FERUM 4.x. Another available option consists in building a Support Vector surrogate function, by regression on a set of well-chosen sampling points (`analysisopt.SVR` set to `'yes'`). The size of the learning database (number of these well-choosen sampling points) is specified by the field `analysisopt.SVR_Nbasis`. This option based on statistical learning theory proves to be a rather cost efficient technique for evaluating sensitivities of models of moderate complexity.

For a Global Sensitivity Analysis, select option 50 in FERUM 4.x. Parameters specific to this analysis are listed hereafter.

```
% Global Sensitivity Analysis (GSA) analysis options
analysisopt.sampling      = 1;          % sampling = 1: Sobol' indices are assessed from
                                        %               Crude Monte Carlo (CMC) simulations
                                        %            2: Sobol' indices are assessed from
                                        %               Quasi Monte Carlo (QMC) simulations
analysisopt.Sobolsetopt   = 0;          % Sobolsetopt = 0: Sobol' sequences from Matlab
                                        %                  Statistic toolbox
                                        %               1: Sobol' sequences from Sobol'
                                        %                  ( Broda - http://www.broda.co.uk/ )
analysisopt.SVR           = 'no';       % SVR = 'yes': a SVR is built as a surrogate to
                                        %              the physical model. Sobol' indices
                                        %              are assessed on the SVR surrogate
                                        %       'no':  no SVR surrogate is built. Sobol' indices
                                        %              are assessed on the physical model,
                                        %              by means of gfundata data structure
analysisopt.first_indices = 1;          % first_indices = 1: assessment of first order indices
                                        %                 0: no assessment
analysisopt.total_indices = 1;          % total_indices = 1: assessment of total indices
                                        %                 0: no assessment
analysisopt.all_indices   = 1;          % all_indices = 1: assessment of first order, second order,
                                        %                  ..., all order indices
                                        %                  (analysisopt.first_indices must be set to 1)
                                        %               0: no assessment
analysisopt.NbCal         = 30;         % Number of replications of Sobols' indices assessments
                                        % (based on the same SVR surrogate, if analysisopt.SVR = 'yes')
                                        %    NbCal = 1: one single assessment
                                        %    NbCal > 1: assessment of mean and variance
                                        %               for each Sobols'indice
```

```
%  Global Sensitivity Analysis (GSA)- SVR options
analysisopt.SVRbasis      = 'CVT_unif'; % SVRbasis =
                                        %    'Sobol_norm': Points are deterministically generated
                                        %                  in the standard space (normal distribution)
                                        %                  (use of Sobol' sequences in [0 1] hypercube)
                                        %    'CVT_unif'  : Points are deterministically generated
                                        %                  in an hypersphere with a specific radius
                                        %                  (use of Voronoi cells)
analysisopt.SVR_Nbasis      = 200;      % Number of points in the design of experiments


analysisopt.gridsel_option  = 1;        % SVR parameter for cross validation search (do not modify)
% Parameters C, epsilon and sigma (RBF kernel is chosen) are determined in the train_SVR.m function.
% However, the user may have some knowledge of the problem in order to define a grid search
% for the hyperparameters (C, epsilon, sigma)
analysisopt.n_reg_radius    = 200000;   % Number of samples for assessing the radius of the
                                        % learning hypersphere, when analysisopt.SVRbasis = 'CVT_unif'
analysisopt.svm_buf_size    = 3500^2;   % RAM-size dependent parameter for eval_svm.m function
```

3.9 Reliability-Based Design Optimization

FERUM 4.x now offers Reliability-Based Design Optimization (RBDO) capabilities. The problem of interest reads, in its most basic and general formulation:

$$\min_{\boldsymbol{\theta}} c(\boldsymbol{\theta}) \text{ s.t. } \left| \begin{array}{c} f_1(\boldsymbol{\theta}), \ldots, f_{q-1}(\boldsymbol{\theta}) \leq 0 \\ f_q(\mathbf{x}, \boldsymbol{\theta}) = \beta_t - \beta(\mathbf{x}, \boldsymbol{\theta}) \leq 0 \end{array} \right. \tag{20}$$

where:

- $\boldsymbol{\theta}$ stands for the design variables of the problem, either purely deterministic variables $\boldsymbol{\theta}_g$ or distribution parameters $\boldsymbol{\theta}_f$,
- $c(\boldsymbol{\theta})$ is the cost function to be minimized,
- $f_1(\boldsymbol{\theta}), \ldots, f_{q-1}(\boldsymbol{\theta})$ is a vector of $(q-1)$ deterministic constraints over the design variables $\boldsymbol{\theta}$,
- $f_q(\mathbf{x}, \boldsymbol{\theta})$ is the reliability constraint enforcing the respect of the design rule referred to as the limit-state function and considering the uncertainty to which some of the model parameters $\mathbf{x}$ are subjected to. β_t is the targeted safety index.

One way to answer the problem in Equation (20) consists in a brute-force outer optimization loop over the reliability evaluation, here termed "Nested bi-level approach" (N2LA). This might be computational expensive in the case of simulation-based methods such as MCS and DS, as addressed in [RP04] and [RP07] respectively. However, if based on FORM, this brute-force method gives a solution within a reasonable amount of calls to the limit-state function. The outer optimization loop makes use of the Polak-He optimization algorithm [Pol97] and requires the gradients of both cost and constraints functions, which themselves require the gradient of the reliability index β *w.r.t.* design variables $\boldsymbol{\theta}$.

Previous RBDO applications of the Polak-He algorithm showed that its rate of convergence was highly dependent on the order of magnitude of the design parameters, cost and constraints functions. In FERUM 4.x, all these values are normalized at each Polak-He iteration, thus improving and ensuring convergence, whatever the initial scaling of the problem in Equation (20). Convergence to an optimum is assumed to be obtained when the cost function has reached a stable value and all the constraints are satisfied, *i.e.* $f_1(\boldsymbol{\theta}), \ldots, f_q(\boldsymbol{\theta}) \leq 0$.

In order to carry out a N2LA FORM-based RBDO analysis, select option 40 in FERUM 4.x. The current implementation is so far restricted to deterministic parameters as design parameters, *i.e.* $\boldsymbol{\theta} = \boldsymbol{\theta}_g$. The cost function (`rbdo_fundata.cost` field) is expressed in the form $c(\boldsymbol{\theta}) = c_0 + c_f p_f$, where p_f represents the failure probability. Deterministic constraints are defined by means of the `rbdo_fundata.constraint` field. Parameters specific to the RBDO analysis are listed hereafter.

```
                                                    % Cost function in the form:
                                                    % c_0 + c_f * p_f
rbdo_fundata.cost = {       '4/3.*pi.*(r1.^3 - r0.^3)'     % c_0 term
                      '0e2 * 4/3.*pi.*(r1.^3 - r0.^3)' };  % c_f term

rbdo_fundata.constraint = { '- r0 + 40'                    % Deterministic constraints:
                            'r1 - 150'                     % fi <= 0, i = 1,...,(q-1)
                            'r0 - r1'
                          };
```

```
rbdo_parameters.alpha                   = 0.5;      % ph_quadprog parameter
rbdo_parameters.beta                    = 0.6;      % ph_quadprog parameter
rbdo_parameters.gamma                   = 3;        % ph_quadprog parameter
rbdo_parameters.delta                   = 1;        % ph_quadprog parameter

rbdo_parameters.steplim                 = 50;       % Max number of steps in stepsize calculation
                                                    % (see ph_quadprog.m)

rbdo_parameters.max_iter                = 7;        % Max number of iterations of N2LA algorithm
rbdo_parameters.target_beta             = 5;        % Target beta reliability index
rbdo_parameters.method                  = 'FORM';
```

3.10 Random fields

(nothing available yet)

4 Getting started

You might want to check out how FERUM works by running one of the examples provided with the program package. To do so, proceed as follows:

1. Go to the FERUM 4.x homepage http://www.ifma.fr/FERUM/.
2. Download the FERUM4.0.zip archive file and extract all compressed files to a directory of your choice on your harddisk where you wish to save all FERUM files (extraction with pathnames).
3. Before you can start using the toolbox, you must to add all FERUM subdirectories (aster_files, dir_simulations, etc.) to your Search Path, so that Matlab knows where to look for them. This is done by using one of the three following options:
   - select "Set Path" in the "File" menu in Matlab,
   - select subdirectories in the Current Directory Browser, then right click, "Add to path", "Selected folder and Subfolders",
   - use the addpath.m function.
4. Read one of the example inputfiles into your Matlab workspace. This can be done by issuing the command `>> inputfile_xxx` in Matlab. You now have the necessary input parameters for a reliability analysis in your current Matlab workspace.
5. Issue the command `>> ferum` in the Matlab workspace.
6. Choose a specific option (*e.g.* option 10 for a FORM analysis).
7. The program now performs an analysis corresponding to the selected option and gives you intermediate results/information as it runs.
8. If the analysis is successful, you will finally see an overview of the available result parameters on the screen. These results are gathered in the fields of a specific structure in the Matlab workspace (*e.g.* `formresults` if a FORM analysis is run).

5 Organization of FERUM 4.x m-files

Main directory:

- ferum.m main file
- definition of distributions
- FERUM pre-processing files (update_data.m, distribution_parameter.m)
- Nataf model specific files including mod_corr_solve.m main file (files necessary for FORM sensitivities in the sensitivities directory)
- Mapping from physical space to standard space (and reverse)
- Limit-state function main file (gfun.m) and gfunbasic.m

inputfile directory:

- Examples of inputfiles
- Input file template (inputfile_template.m)

gfunction directory

- Examples of limit-state functions defined through Matlab .m files

FORM directory

- Search for a single design point with the iHL-RF algorithm (form.m)
- Search for possible multiple design points (form_multiple_dspts.m, via calls to form.m)
- Auxiliary files

sensitivities directory

- All files necessary to assess FORM sensitivities *w.r.t.* distribution parameters $\boldsymbol{\theta}_f$ (means, standard deviations, correlation, distribution parameters) and deterministic parameters $\boldsymbol{\theta}_g$ in the limit-state function

SORM directory

- SORM curvature-fitting method (sorm_cfh.m)
- SORM point-fitting method (sorm_pf.m)
- Auxiliary files

distributions directory

- Distribution of the limit-state function based on a Monte Carlo simulation (distribution_analysis.m)

simulations directory

- Crude Monte Carlo (CMC) simulation or Importance Sampling (IS) (simulation_single_dspt.m, simulation_single_dspt_lowRAM.m)
- Mersenne Twister pseudo-random number generator (C++ source files twister.cpp-32bit and twister.cpp-64bit respectively for 32-bit and 64-bit computational platforms, Windows .dll file, Linux 32-bit .mexglx file)

dir_simulations directory

- Directional Simulation (dir_simulation.m) with either random or deterministic directions (through calls to eq_point_set.m function and auxiliary files in eq_sphere_partitions directory)

subset_simulations directory

- Subset Simulation (subset_simulation.m)
- Auxiliary files

2SMART directory

- ^{2}SMART simulation (ssvm_simulation.m)
- Auxiliary files
- kmeans files for clustering (kmeans subdirectory)

Sobol directory

- Global Sensitivity Analysis based on Sobol' indices (Sobol_SA.m), either through calls to the original limit-state function or to a SVR surrogate (Support Vector Regression).
- Sobol' quasirandom sequences generated by means of the Matlab Statistical Toolbox or the sobolseq51 Windows .dll file from BRODA (BRODA subdirectory)
- Training samples possibly re-arranged by means of Centroidal Voronoi Tessellation (CVT subdirectory)

N2LA directory

- Nested bi-level RBDO analysis based on FORM and gradient of β index *w.r.t.* design variables (N2LA.m)
- Polak-He optimization algorithm (ph_quadprog.m)
- fun.m auxiliary file

SVM directory

- Spider toolbox (spider subdirectory)

- SVM auxiliary files (svm_init.m and svr_init.m: initialization for classification and regression respectively, train_SVR.m: training of a SVR surrogate, eval_svm.m: evaluation based on a SVM surrogate, used both for classification and regression, gfunsvr.m: SVR surrogate function)

ferum_in_loop directory

- Example script files showing how to call FERUM in a silent mode

aster_files directory

- Files required for external calls to Code_aster Finite Element code in a sequential way (inputfile, gfun-aster.m function called by gfun.m, Windows specific files, Code_aster template files, gawk binary, gawk file for post processing of FE results)

6 Release summary

FERUM 4.1 (July 1, 2010) vs. 4.0 (September 9, 2009):

- A new method is offered to assess small failure probabilities, based on Support Vector Machine (SVM) surrogates. This method referred as ^{2}SMART consists in essence to build a SVM classifier for each Subset Simulation-like y_i-threshold intermediate limit-state.
- Global Sensitivity Analysis based on Sobol' indices now works with models characterized by a vectorial output. The limit-state function may either return a scalar value if `gfundata(lsf).ng` field is set to 1 in the input file (default value) or a vector of values if `gfundata(lsf).ng` is set to a value strictly greater than 1. The Global Sensitivity Analysis works either with the physical model itself or SVR surrogates. If the limit-state function has a vectorial output, a SVR surrogate is built for each component of this vectorial output.
- FORM with search for multiple design points can now be run in silent mode (`analysisopt.echo_flag` set to 0).
- All SVM m-functions are gathered in the svm subdirectory (including the Spider toolbox). They may be useful either in a ^{2}SMART analysis or in a SVR-based Global Sensitivity analysis.
- A slightly modified version of twister.cpp file is given for 64-bit computational platforms. It prevents crashes which occur when the internal state of the generator needs to be saved by issuing the command `>> S = twister('state')`.

References

[AB01] S.-K. Au and J.L. Beck. Estimation of small failure probabilities in high dimensions by subset simulation. *Probabilistic Engineering Mechanics*, 16:263–277, 2001.

[BDL10] J.-M. Bourinet, F. Deheeger, and M. Lemaire. Assessing small failure probabilities by combined subset simulation and support vector machines. *Submitted to Structural Safety*, 2010.

[Bje88] P. Bjerager. Probability integration by directional simulation. *Journal of Engineering Mechanics*, 114(8):1285–1302, 1988.

[BL08] J.-M. Bourinet and M. Lemaire. FORM sensitivities to correlation: Application to fatigue crack propagation based on Virkler data. In *Proc. of the 4th International ASRANet Colloquium*, Athens, Greece, 2008.

[BMD09] J.-M. Bourinet, C. Mattrand, and V. Dubourg. A review of recent features and improvements added to FERUM software. In *Proc. of the 10th International Conference on Structural Safety and Reliability (ICOSSAR'09)*, Osaka, Japan, 2009.

[DKD98] A. Der Kiureghian and T. Dakessian. Multiple design points in first and second-order reliability. *Structural Safety*, 20(1):37–49, 1998.

[DKHF06] A. Der Kiureghian, T. Haukaas, and K. Fujimura. Structural reliability software at the University of California, Berkeley. *Structural Safety*, 28(1-2):44–67, 2006.

[DKLH87] A. Der Kiureghian, H.-Z. Lin, and S.-J. Hwang. Second-order reliability approximations. *Journal of Engineering Mechanics*, 113(8):1208–1225, 1987.

[DM07] O. Ditlevsen and H.O. Madsen. *Structural reliability methods*. Internet Edition 2.3.7, 2007.

[DNBF09] V. Dubourg, C. Noirfalise, J.-M. Bourinet, and M. Fogli. FE-based reliability analysis of the buckling of shells with random shape, material and thickness imperfections. In *Proc. of the 10th International Conference on Structural Safety and Reliability (ICOSSAR'09)*, Osaka, Japan, 2009.

[Hur04] J.E. Hurtado. *Structural reliability, statistical learning perspectives*. Lecture Notes in Applied and Computational Mechanics Vol. 17, Springer, 2004.

[LDK86] P.-L. Liu and A. Der Kiureghian. Multivariate distribution models with prescribed marginals and covariance. *Probabilistic Engineering Mechanics*, 1(2):105–112, 1986.

[Lem09] M. Lemaire. *Structural reliability*. John Wiley, 2009.

[Mac92] D. MacKay. Information-based objective functions for active data selection. *Neural Computation*, 4(4):590–604, 1992.

[NBGL06] L. Nespurek, J.-M. Bourinet, A. Gravouil, and M. Lemaire. Some approaches to improve the computational efficiency of the reliability analysis of complex crack propagation problems. In *Proc. of the 3rd International ASRANet Colloquium*, Glasgow, UK, 2006.

[Pol97] E. Polak. *Optimization: Algorithms and consistent approximations*. Springer-Verlag, 1997.

[PS06] M.F. Pellissetti and G.I. Schuëller. On general purpose software in structural reliability - an overview. *Structural Safety*, 28:3–16, 2006.

[RP04] J.O. Royset and E. Polak. Reliability-based optimal design using sample average approximations. *Probabilistic Engineering Mechanics*, 19(4):331–343, 2004.

[RP07] J.O. Royset and E. Polak. Extensions of stochastic optimization results to problems with system failure probability functions. *Journal of Optimization Theory and its Application*, 132(2):1–18, 2007.

[Sob07] I.M. Sobol'. Sensitivity estimates for nonlinear mathematical models. *Mathematical Modelling and Computational Experiments*, 1:407–414, 2007.

[SS02] B. Schölkopf and A. Smola. *Learning with kernels*. MIT Press, 2002.

[STC00] J. Shawe-Taylor and N. Cristianni. *An introduction to support vector machines and other kernel-based learning methods*. Cambridge University Press, 2000.

[Vap95] V. Vapnik. *The nature of statistical learning theory*. Springer, 1995.

[ZDK94] Y. Zhang and A. Der Kiureghian. Two improved algorithms for reliability analysis. In R. Rackwitz, G. Augusti, and A. Borri, editors, *6th IFIP WG 7.5 Working Conference on Reliability and Optimisation of Structural Systems, Reliability and Optimization of Structural Systems*. Chapman & Hall, 1994.

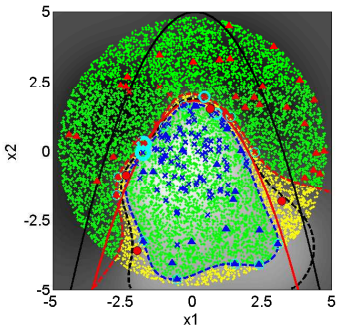
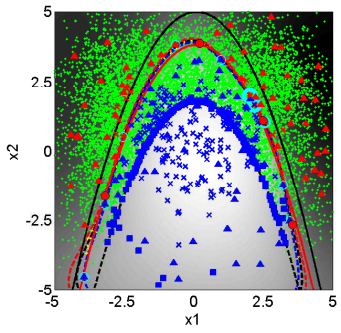
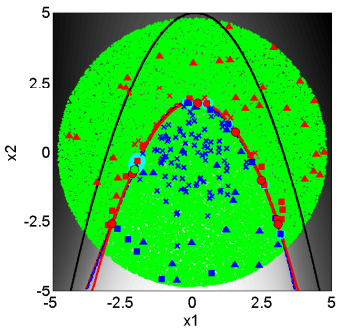
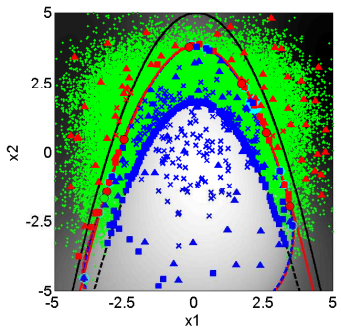
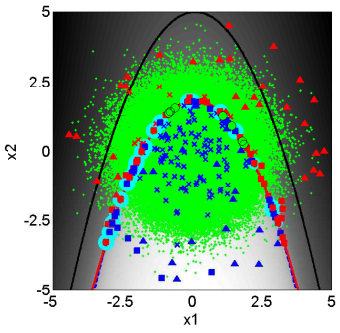
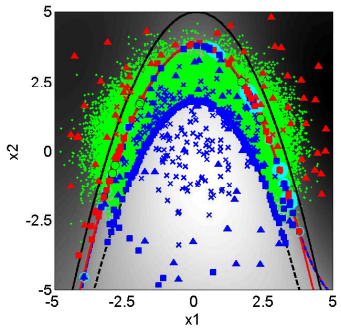
	Subset Simulation-like step	
	$i = 1$	$i > 1$
	(L1)	(L)
	Uniform work population	$\lambda_1 r_j$-mM work population
Localization stage (L) N_1 sampling points Iteration $1 \leq k < k_1$		
	(S1)	(S)
	Uniform work population	$\lambda_2 r_j$-mM work population
Stabilization stage (S) N_2 sampling points Iteration $k_1 \leq k < k_2$		
	(C1)	(C)
	Work/MC population	Work/r_j-mM population
Convergence stage (C) N_3 sampling points Iteration $k_2 \leq k \leq k_3$		

Figure 9: Coarse to fine adjustment of the sampling process [BDL10].