Notes in Reliability-based Design Optimization using Double Loop Approach

Nophi Ian Biton

January 16, 2022

1 Reliability-based Design Optimization (RBDO)

Uncertainty in engineering analysis and design is inevitable. It may be due to imprecise models used or inherent variability of measurements, these uncertainties should properly addressed when designing structures to attain certain degree of safety. The goal of structural design is to determine the most economical use of materials that will have an ample ammount of capacity to carry the structural demands. Designing a structure is always an iteratative procedure finding an "optimal" or sufficient design to attain subject to constraints (strength/stress related constraints based from building codes, geometric constraints, displacement constraints etc.). All structural design procedure follows some form optimzation process. However, the demands (loads to be applied in the structure) and capacity (determined from its material properties, sectional/geometric properties) have uncertainties. Thus, a structural design process which considers explicity the uncertainty in the demands and capacities in a structure is termed as Reliability-based Design Optimization (RBDO).

A more general formulation of structural optimization problems involving uncertainty was described by [5]. Specifically, for single objective optimization problems the reliability appears in the contsraint or as an objective. Due to explicit inclusion of uncertainty in the design process, structural reliability analysis is carried out. These problems are stated as

- Minimize the cost of design, subject to reliability and structural constraints
- Maximize the reliability of the design, subject to cost and structural constraints

In this document, the focus will be on the first type problem. An RBDO problem is further subdivided into two parts based on the type of probabilistic constraint: Component or System. When the reliability analysis is done for a single limit-state function then it is called a Component RBDO (CRBDO). If the reliability analysis is done for several components (can be series, parallel or general) then it is called System RBDO (SRBDO). This document will focus on CRBDO problems. Based from the formulation of [4], its mathematical statement is

$$\begin{aligned} \min_{\mathbf{d}} \quad & f(\mathbf{d}) \\ \text{s.t.} \quad & P[g_i(\mathbf{d}, \mathbf{X}) \leq 0] \leq P_i^t \quad i = 1, \ldots, n \\ & h_j(\mathbf{d}) \leq 0 \quad j = 1, \ldots, m \\ & \mathbf{d}^L \leq \mathbf{d} \leq \mathbf{d}^U \end{aligned} \tag{1}$$

where $\mathbf{d} \in R^k$ is the vector of deterministic/design variables; $\mathbf{X} \in R^l$ is the vector of random variables; $f(\cdot)$ is the objective function (usually the weight of the structure); $g_i(\cdot)$, $i = 1, \ldots, n$ is the ith limit state function with occurence of failure defined as $g_i(\cdot) \leq 0$; P_i^t is target or threshold failure probability of the ith limit state; $h_j(\mathbf{d}) \leq 0$ $j = 1, \ldots, m$ is the jth deterministic inequality constraint; $\mathbf{d}^L$ and $\mathbf{d}^U$ are lower and upper bound on $\mathbf{d}$ and n, m, k and l are the number of probabilistic constraints, deterministic constraints, deterministic variables and proabilistic variables, respectively.

Conventionally, the limit-state function is formulated in the form $g(\mathbf{d}, \mathbf{X}) \leq 0$ which denotes the occurence of failure. The probabilistic constraint of Eq. (1) asserts that the probability of the limit-state function is

less than some target probability of failure. An alternative form of this formulation is the use of cumulative distribution function (CDF) of the limit state function [4], which is

$$P[g_i(\mathbf{d}, \mathbf{X}) \leq 0] = F_{g_i}(0) \leq \Phi\left(-\beta_i^t\right) \tag{2}$$

where $F_{g_i}(\cdot)$ is the CDF of the limit-state function $g_i(\cdot)$; $\Phi(\cdot)$ is the CDF of the standard normal distribution; and β_i^t is the target reliability index. There are a couple of methods that can be used for reliability analysis of Eq. (2). The more popular one is using First Order Reliability Method (FORM) as discussed in [3]. This method requires some transformation of random variables $\mathbf{X}$ into a set of standard normal random varibales $\mathbf{U}$.

2 Double-Loop Approach

Finding the solution of Eq. (**??**) is a complex process due to two reasons: (a) finding the optimal design variables $\mathbf{d}$ itself (b) performing the reliability analysis on the probabilistic constraint. The most straigthforward approach is called the **Double-Loop Approach**. In this approach, there are two loops in the solution process. The 'outer loop' which is an optimization process in finding the optimal design $\mathbf{d}$ and an 'inner loop' which is also an optimization process in finding the reliability (or conversely the probability of failure). The term 'loop' here is used due to the iterative nature of these procedures.

2.1 Reliability Index Approach (RIA)

Under the Double loop approach, the more commonly used approach is the Reliability Index Approach (RIA) [2]. In this approach the reliability index is used to check the probabilistic constraint. The mathematical statement is

$$\begin{aligned} \min_{\mathbf{d}} \quad & f(\mathbf{d}) \\ \text{s.t.} \quad & \beta_i = -\Phi^{-1}\left[F_{g_i}(0)\right] \geq \beta_i^t, \;\; i = 1, 2, \ldots, n \end{aligned} \tag{3}$$

where β_i is the distance from the origin of the space of standard normal random variables $\mathbf{U}$=$\mathbf{U}(\mathbf{X})$ to the nearest point on the limit state surface $G_i(\mathbf{d}, \mathbf{U}) = 0$, in which $G_i(\cdot)$ is the limit-state function $g_i(\cdot)$ determined in terms of $\mathbf{U}$, that is, $g_i(\mathbf{d}, \mathbf{X}) = G_i(\mathbf{d}, \mathbf{U}(\mathbf{X}))$. The β_i is called the reliability index. FORM [3] can be used to determine reliability index. The l-dimensional limit-state surface for the design $\mathbf{d}$

$$S_i(\mathbf{d}) = \{\mathrm{U} \in R^l | G_i(\mathbf{d}, \mathbf{U}) = 0\} \tag{4}$$

separates the safe and failure domains. To find the value of $\mathbf{U}_i^*$ along the limit state surface $S_i(\mathbf{d})$ that is closest to the origin in standard normal space, the nonlinear constrained optimization problem is defined as

$$\begin{aligned} & \mathbf{U}_i^* = \arg\min \|\mathbf{U}\| \\ & \text{s.t.} \quad G_i(\mathbf{d}, \mathbf{U}) = 0 \end{aligned} \tag{5}$$

where $\mathbf{U}_i^*$ is called the design point or most probable point (MPP) or also called most probable failure point (MPFP) of the ith limit-state function and "arg min" denotes the argument of the minimum of the function $\|\cdot\|$.

2.2 Performance Measure Approach (PMA)

In the RIA, the reliability index β_i which is determined by finding the MPP ($\mathbf{U}_i^*$) is compared with the target reliability index to check whether the probabalistic constraint is satisfied. Alternatively, the Performance Measure Approach checks the probabalistic constraint in Eq. (**??**) in the form of performance function. Inverse Reliability Analysis (IRA) is done by searching for the failure point corresponding to the lowest performance that satisfies a target reliability index [1]. Essentially the value of the limit state function is varied or its minimum value is determined with a fixed target reliability index. The mathematical formulation is given by

$$\begin{aligned} \min_{\mathbf{d}} \quad & f(\mathbf{d}) \\ \text{s.t.} \quad & g_{p_i} \geq 0, \;\; i = 1, 2, \ldots, n \end{aligned} \tag{6}$$

where g_{p_i} is the performance function. The IRA focuses on finding a specific performance corresponding to a given reliability. The limit state is varied while the reliability index (which is the target reliability index) is fixed. This approach is based on principle that minimizing a complex function under simple constraints is more efficient than minimizing simple funcction under complex constraints [1]. The performance function g_{p_i} is obtained from

$$\begin{aligned} g_{p_i} = & \min \; G_i(\mathbf{d}, \mathbf{U}) \\ \text{s.t.} \quad & \|\mathbf{U}\| = \beta_i^t \end{aligned} \tag{7}$$

3 Double-loop RBDO using FERUM and Matlab

In this section, the double-loop approach in performing RBDO will be discussed. The **F**inite **E**lement **R**eliability **U**sing **M**atlab (FERUM) will be used to perform Reliability Analysis (RA) and Inverse Reliability Analysis (IRA).

FERUM is an open source Matlab code that is able to perform a couple of analysis in structural reliability. It was first developed by researchers from the University of California Berkeley. The FERUM version 3.0 can be downloaded at `http://projects.ce.berkeley.edu/ferum/`.

The programming language Matlab will be used. This programming tool has several optimization tool. Specifically, the function `fmincon|` which finds the minimum of constrained nonlinear multivariable function. A theoretical explanation on how `fmincon` finds the optimal solution is discussed here.

3.1 Numerical Problem with Nonlinear limit state

An RBDO problem adapted from [1] is given by:

$$\begin{aligned} \min_{\mathbf{d}} \quad & f(\mathbf{d}) = d_1^2 + d_2^2 \\ \text{s.t.} \quad & P\left[\frac{1}{5} d_1 d_2 X_2^2 - X_1 \leq 0\right] \leq P_f^t \\ & 0 \leq \mathbf{d} \leq 15 \end{aligned} \tag{8}$$

The problem is to find for the optimal values $\mathbf{d} = [d_1, d_2]$ that will satisfy the probabilistic constraints in Eq. (8) with target failure probability of $P_f^t = 1\%$ or $\beta^t = 2.32$. The random variables $\mathbf{X} = [X_1, X_2]$ are normally distributed with means $\mu_X = [5.0, 3.0]$ and coefficient of variations C.O.V. $= [0.3, 0.3]$. The random variables are statistically independent.

3.2 Matlab files for RBDO

There several matlab files and folders that will be defined when performing RBDO using Matlab and FERUM. These files will be

- `main.m` - This file should contain the outer loop of RBDO process meaning this will contain the commands for optimizing the design variables. The handles for the anonymous functions pointing to the objective and constraints should be defined here. All the input arguments for `fmincon` optimizer should also be defined. This file will be the same for both the RIA and PMA.

```
clc,clear; format compact; format shortG;

fun = @calc_obj;       % objective function
nonlcon = @calc_const; % constraint function

lb = [0,0];
ub = [15,15];

A = [];
b = [];
Aeq = [];
beq = [];

x0 = [7.5,7.5];   % intial guess

options = optimoptions('fmincon','Display','iter','Algorithm','sqp');
[x,fval,exitflag,output] = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)
```

- `calc_obj.m` - This file should contain the objective function $f(\mathbf{d})$ as defined in Eq. (8). This file will be the same for both the RIA and PMA.

```
function [obj] = calc_obj(d)

% objective function
obj = d(1)^2+d(2)^2;

end
```

- `calc_const.m` - This anonymous Matlab function should contain the constraints as defined in Eq. (8). There will be two different consideration of constraint function with respect to RIA and PMA. In RIA, constraint function is checked via the reliability index as in Eq. (3) or the probability of failure as in (8). In this specific example, the probability of failure will be compared here instead of the reliability index since target failure probability is an exact value of 1%. Take note if the reliability index is used then the results from FERUM is stored in `formresults.beta1` else if the probability of failure is used the result is stored in `formresults.pf1`. Two ouputs of the `calc_const` function will be defined which are for inequality $c(x) \leq 0$ and equality $ceq(x) = 0$ constraints. The input arguments will be the values of design variables $\mathbf{d}$. These values will be used in FERUM which will be defined in `ferum_inputfile.m`.

```
function [c,ceq] = calc_const(d)

%inequality nonlinear constraint
ferum_inputfile;
c = formresults.pf1 - 1/100;

% equality nonlinear constraint
ceq = [];
end
```

For the case of PMA, the constraint is in the form of performance (Eq. 7). The g_{p_i} is calculated using the Inverse Reliability Analysis (IRA) function of FERUM. The results of IRA is stored in `inverse_formresults`. Take note that the performance $g_{p_i} \geq 0$ or in the format of constraint definition of `fmincon` $c(x) \leq 0$, it should be $-g_{p_i} \leq 0$.

```
function [c,ceq] = calc_const(d)

%inequality nonlinear constraint
ferum_inputfile_inverse;
c = -inverse_formresults.theta;

% equality nonlinear constraint
ceq = [];
end
```

In the PMA formulation of Eq. (6) and (7), g_{p_i} is the value of limit state function evaluation. The value of the limit state function evaluation can be set equal to some parameter θ. In the `ferum_inputfile_inverse.m` file, FERUM will used to perform IRA.

- `g_func.m` - This file will contain the limit-state function definition. This function will be called by the FERUM code. The limit state function in FERUM must be in the form $g_i(X, \theta) \leq 0$, where X are the random variables and θ are parameters (which may include the design variables). In RIA, the definition of limit state function is straight forward as defined in Eq. (8).

```
function val = g_func(x1,x2,d1,d2)

% limit state function
val = (1/5)*d1*d2*x2^2-x1;
end
```

In the PMA formulation, the value of the limit-state function is the one being minimized. The value of limit-state function is set to θ; therefore, the `g_func` in FERUM will be of the form $g_i(X) - \theta \leq 0$. Effectively, IRA is finding for the value of θ that will minimize the limit-state function.

```
function val = g_func(x1,x2,d1,d2,theta)

% limit state function
val = (1/5)*d1*d2*x2^2-x1 - theta;
end
```

- `ferum_inputfile.m` or `ferum_inputfile_inverse.m` - This file include the input file to read by FERUM. This file should include the probability data information, analysis option, limit state function definition and other data needed to run FERUM. Take note that this file is NOT a function but a rather just a separate script in order to better organize the input data for FERUM. This file should also include the command that includes the path of where the FERUM codes are located. This is done by the command `addpath(genpath(strcat(pwd,'\ferumcore')))` as in line 4 of the script below. The folder `ferumcore` should be located on the same directory of this file. The probability data should be properly define based on the input file provided by FERUM. The default analysis options can be used. For the limit state function data or `gfundata` in FERUM should be properly defined. Since the limit state function involves parameter other than the random variables, the option `\gfundata(1).parameter` should be set to `yes`. The values of the parameters which is the design variables $\mathbf{d}$ should be saved to `gfundata(1).thetag`. The function name of the anonymous function that defines the limit state function `g_func` should be saved to the variable `gfundata(1).parameter` with all the necesarry and correct input arguments (`x(1),x(2),gfundata(1).thetag(1),gfundata(1).thetag(2)`). Lastly, the `gfundata(1).dgthetag` should be defined. Temporarily a cell array of ones can be used because an error or bug will be encountered if this is not defined. The derivative is actually calculated using finite difference method and not by `gfundata(1).dgthetag` since `FFD` is defined.

```
% EXAMPLE Nonlinear Limit state
clear probdata femodel analysisopt gfundata randomfield systems results output_filename

addpath(genpath(strcat(pwd,'\ferumcore')))

% Probability Data
probdata.marg(1,:) =  [ 1  5   5*0.3   5  0 0 0 0 0]; % dist of Var 1
probdata.marg(2,:) =  [ 1  3   3*0.3   3  0 0 0 0 0]; % dist of Var 2
probdata.correlation = eye(2); % Correlation matrix
probdata.parameter = distribution_parameter(probdata.marg);

% Analysis Options
analysisopt.ig_max    = 100;
analysisopt.il_max    = 5;
analysisopt.e1        = 0.001;
analysisopt.e2        = 0.001;
analysisopt.step_code = 0;
analysisopt.grad_flag = 'FFD';
analysisopt.sim_point = 'dspt';
analysisopt.stdv_sim  = 1;
```

```
analysisopt.num_sim   = 100000;
analysisopt.target_cov = 0.0125;

% Limit State function (gfun) data
gfundata(1).thetag = d; % design variables
gfundata(1).evaluator = 'basic';
gfundata(1).type = 'expression';
gfundata(1).parameter = 'yes';
gfundata(1).expression='g_func(x(1),x(2),gfundata(1).thetag(1),gfundata(1).thetag(2))';
gfundata(1).dgthetag = {'1','1'};

% Other Data
femodel = 0;
randomfield.mesh = 0;

% Run modified FORM analysis from FERUM
ferum_form;
```

For the PMA case, the only difference is to include an initial value for the deterministic parameter `gfundata(1).deterministic_parameter_start_value` as in line 27 of the script below. Also, the variable `theta(1)` should be include as an additional parameter for the function in the `gfundata(1).expression`

```
% EXAMPLE Nonlinear Limit state
clear probdata femodel analysisopt gfundata randomfield systems results output_filename

addpath(genpath(strcat(pwd,'\ferumcore')));

% Probability Data
probdata.marg(1,:) =  [ 1  5   5*0.3   5  0 0 0 0 0]; % dist of Var 1
probdata.marg(2,:) =  [ 1  3   3*0.3   3  0 0 0 0 0]; % dist of Var 2
probdata.correlation = eye(2);
probdata.parameter = distribution_parameter(probdata.marg);

% Analysis Options
analysisopt.ig_max    = 100;
analysisopt.il_max    = 5;
analysisopt.e1        = 0.001;
analysisopt.e2        = 0.001;
analysisopt.step_code = 0;
analysisopt.grad_flag = 'FFD';
analysisopt.e3 = 0.001;           % Tolerance on how close to beta target is the
    solution point
analysisopt.beta_target = 2.326;    % Target value for the index of reliability

% Limit State function (gfun) data
gfundata(1).thetag = d; % design variables
gfundata(1).evaluator = 'basic';
gfundata(1).type = 'expression';
gfundata(1).parameter = 'yes';
gfundata(1).deterministic_parameter_start_value = 100;
gfundata(1).expression = 'g_func(x(1),x(2),gfundata(1).thetag(1),gfundata(1).thetag(2),
    theta(1))';
gfundata(1).dgthetag = {'1','1'};

% Other Data
femodel = 0;
randomfield.mesh = 0;

% Run modified FORM analysis from FERUM
ferum_inv_form
```

For both FERUM input files, a command is included at at the end of the files to run FERUM: `ferum_form` for RIA and `ferum_inv_form` for PMA. These Matlab files are modification of the Matlab function `ferum.m`. These files will be discussed in the next item.

- `ferumcore` - This folder contains all the FERUM Matlab codes that can be downloaded from its official website. There are two functions added in this folder apart from the original files of Matlab: `ferum_form` for RIA and `ferum_inv_form` for PMA. These two functions are just modification from

the orginal `ferum.m`. The original `ferum.m` by commenting out all output results to the command window as well as preselecting the analysis type as `form` for RIA and `Inverse FORM Analysis` for PMA.

3.3 Results of RBDO

Upon running the `main.m` file, the results are summarized in the table below. Both approaches were able to arrive to the optimal design found in literature [1] of $\mathbf{d} = [5.65, 5.65]$.

Method	**Optimal Design**	**Object Function/Cost**	**Iterations**	**Function Evaluation**
RIA	[5.6517, 5.6617]	63.883	8	27
PMA	[5.6491, 5.6498]	63.839	4	15

The matlab files can be accessed in a github reposity here.

References

[1] Younes Aoues and Alaa Chateauneuf. Benchmark study of numerical methods for reliability-based design optimization. *Structural and Multidisciplinary Optimization 2009 41:2*, 41(2):277–294, Aug 2009.

[2] I. Enevoldsen and J. D. Sørensen. Reliability-based optimization in structural engineering. *Structural Safety*, 15(3):169–196, 1994.

[3] Armen Der Kiureghian. First- and Second-Order Reliability Methods. In *Engineering Design Reliability Handbook*, chapter 14. CRC press, Boca Raton, Florida, 2005.

[4] Tam H. Nguyen, Junho Song, and Glaucio H. Paulino. Single-loop system reliability-based design optimization using matrix-based system reliability method: Theory and applications. *Journal of Mechanical Design, Transactions of the ASME*, 132(1):0110051–01100511, dec 2010.

[5] J. O. Royset, A. Der Kiureghian, and E. Polak. Reliability-Based Optimal Design of Series Structural Systems. *Journal of Engineering Mechanics*, 127(6):607–614, Jun 2001.