Struct Multidisc Optim (2016) 54:391–410
DOI 10.1007/s00158-016-1414-5

EDUCATIONAL ARTICLE

Matlab codes of Subset Simulation for reliability analysis and structural optimization

Hong-Shuang Li[1] · **Zi-Jun Cao**[2]

Received: 11 April 2015 / Revised: 18 September 2015 / Accepted: 18 October 2015 / Published online: 17 March 2016

Abstract This paper presents two efficient and compact Matlab codes of Subset Simulation for reliability analysis and structural optimization. The codes for reliability analysis and structural optimization comprise of the direct Monte Carlo and Markov Chain Monte Carlo. The theoretical and numerical elements of Subset Simulation are briefly presented in this paper, as well as the detailed instructions to implement the standard codes for solving reliability analysis and structural optimization problems. The paper also discusses simple extensions of argument check, post-processing, alternative stop criterion and constraint-handling. Four examples are presented to demonstrate these codes, two for reliability analysis and two for structural optimization. This paper will be helpful for the students and newcomers both in reliability analysis and structural optimization to understand and use Subset Simulation. The complete codes are included in Appendixes 1 and 2, and they can be downloaded from https://sites.google.com/site/rasosubsim/.

Keywords Subset Simulation · Matlab · Education · Reliability analysis · Structural optimization · Markov Chain Monte Carlo

✉ Hong-Shuang Li
hongshuangli@nuaa.edu.cn

[1] Key Laboratory of Fundamental Science for National Defense-Advanced Design Technology of Flight Vehicles, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China

[2] State Key Laboratory of Water Resources and Hydropower Engineering Science, Wuhan University, Wuhan 430072, China

1 Introduction

Subset Simulation provides a powerful tool for the assessment of small failure probability and optimization design in engineering design field. It is originally developed from the concept of conditional probability and Markov Chain Monte Carlo (MCMC) technique by Au and Beck (2001), and well-known as an efficient Monte Carlo technique for variance reduction in reliability analyses. Many variants of standard Subset Simulation have appeared in the literature due to its elegant idea, robustness in high dimensions, high efficiency, etc. Ching et al. introduced the splitting of a trajectory into MCMC algorithm to increase the acceptance rate of a candidate sample when there is a causal relationship between inputs and outputs (Ching et al. 2005a). They further developed a hybrid Subset Simulation method to improve their version (Ching et al. 2005b). By separating the calculation of linear elastic and inelastic structural response, Katafygiotis and Cheung presented a two-stage Subset Simulation for estimating the reliability of inelastic structural systems subjected to Gaussian random excitations (Katafygiotis and Cheung 2005). Along the spirit of decomposing the failure region into a series of sub-regions, Katafygiotis and Cheung also proposed a spherical Subset Simulation for high dimensional problems (Katafygiotis and Cheung 2007). Zuev et al. developed an optimal scaling strategy for the modified Metroplis-Hasting algorithm (MMH), provided a theoretical basis for the optimal value of the conditional failure probability, and proposed a Bayesian Subset Simulation which can produce the posterior probability density function (PDF) of the failure probability instead of a constant value (Zuev et al. 2012). Apart from abovementioned variants of Subset Simulation, some researchers aimed at reducing the correlation in conditional samples generated by MCMC techniques (Santoso et al. 2011; Zuev and Katafygiotis 2011), and other researchers

concentrated on combining Subset Simulation with surrogate models, e.g., artificial neural network (Papadopoulos et al. 2012), support vector machine (Bourinet et al. 2011), to further improve its efficiency.

Subset Simulation was originally developed for solving reliability analysis and risk assessment of civil structures subjected to uncertain earthquake ground motions (Au and Beck 2001; Ching et al. 2005a, b; Katafygiotis and Cheung 2005, 2007; Au and Beck 2003; Au and Wang 2014; Tee et al. 2014). It has been widely applied in many engineering fields, such as aerospace engineering (Pellissetti et al. 2006; Song et al. 2009), geotechnical engineering (Santoso et al. 2011; Wang et al. 2010, 2011; Li et al. 2015, 2016), nuclear engineering (Zio and Pedroni 2012; Wang et al. 2015), and also as simulation engine for improving efficiency in different stochastic simulation algorithms, for example (Chiachío et al. 2014a, b). Please refer to Au and Wang (Au and Wang 2014) for a comprehensive summary of the applications of Subset Simulation.

Subsequent studies by Li and Au (2010); (Li 2011); Li and Ma (2015), showed that Subset Simulation can be efficiently used for structural optimization. Based on the idea that an optimization problem can be formulated as a reliability problem by augmenting the design variables as random variables, an artificial reliability problem can be constructed, which follows a similar idea for solving reliability problem using Monte Carlo simulation (Au 2005). As a result, Subset Simulation is extended to solve constrained optimization problems (Li and Au 2010), unconstrained optimization problems (Li 2011) and discrete optimization problems (Li and Ma 2015) as a stochastic searching and optimization algorithm.

In this paper, two efficient and compact Matlab codes are presented to perform reliability analysis and structural optimization using Subset Simulation. The Matlab codes presented here are intended to facilitate engineering education and applications of Subset Simulation, which will provide instructions to students and newcomers to Subset Simulation. Matlab is a high-level computational language which has many outstanding characteristics, e.g., accessible syntax, excellent debugging tools and extensive graphics output. These allow users to concentrate on the physical and mathematical modeling of engineering problems without being distracted by how to implement it. Therefore, Matlab is the ideal environment for learning to program, solving computationally problems and writing prototype programs. Both codes include three parts: (1) algorithm parameter definition, (2) direct Monte Carlo (DMC), and (3) MCMC. These two codes are organized and written as a built-in function in Matlab so that no modification is required for new applications except for input information.

This paper starts with a brief theoretical background of Subset Simulation for reliability analysis and structural optimization. Section 3 shows the details and instructions of Matlab codes of Subset Simulation. The usage of Matlab codes are explained using two numerical examples. Section 4 discuses several extensions to the developed codes. Section 5 presents the numerical implementation procedures and results of two practical problems, one for structural reliability analysis and the other one for structural optimization design. Finally, major conclusions drawn from this paper are summarized in Section 5. The codes are provided in Appendix 1 and 2, and can also be downloaded from the website: https://sites.google.com/site/rasosubsim/.

2 Theoretical background

Subset Simulation is a stochastic simulation procedure for estimating small failure probabilities and solving optimization problems. Specifically, we consider some engineering systems subject to random input parameters. Let us define the failure region F as the subregion in the $\mathbf{x}$-space that exceeds a response function (or the system performance function) $g(\mathbf{x})$ below a specific threshold value b, as follows:

$$F = \{\mathbf{x} : g(\mathbf{x}) < b\} \tag{1}$$

where $\mathbf{x}$ is the input random vector which models all uncertain parameters in the system. In fact, $g(\mathbf{x})$ can be a nonlinear and implicit function of $\mathbf{x}$. The target failure probability P_F associated with the target failure event F may be very small ($P_F << 1$). Under the setting where a single system response analysis requires the solution of a numerical model (e.g., finite element model), an excessive number of simulations may be required to estimate the target failure probability with a desired accuracy. The basic idea of Subset Simulation is to convert a small probability into a product of a sequence of large conditional probabilities, each of which has a smaller coefficient of variation (c.o.v.). This conversion is achieved by dividing the input parameter space into subset failure domains. Thus, a sequence of intermediate events are required to be defined in the same way of the target failure event

$$F_j = \{\mathbf{x} : g(\mathbf{x}) < b_j\}, j = 1, \ldots, m \tag{2}$$

where b_j is a series of threshold values of the system response and m is the total number of intermediate events. Importantly, it is assumed that $F_1, F_2, \cdots, F_m$ are

a sequence of nested events, i.e., $F_1 \supset F_2 \supset \cdots \supset F_m = F$. Note that the values of b_j cannot be determined in advance. However, the determination of b_j and further the intermediate events can be achieved in an adaptive manner, i.e., setting the conditional probabilities $P(F_i|F_{i-1})$ equal to a specified value (Au and Beck 2001). To ensure the nestedness of F_j, $j=1,\ldots,m$, the threshold values are arranged such as $b_1 > b_2 > \ldots > b_m = 0$. Because of the nested nature of all intermediate events, the target failure probability can be rewritten as

$$P_F = P(F) = P(F_1)\prod_{j=1}^{m-1} P\left(F_j \middle| F_{j-1}\right) \tag{3}$$

By (3), the simulation of a rare event F is subdivided to the simulations of a series of frequently conditional events $F_j|F_{j-1}$. Therefore, generating conditional samples in $F_j|F_{j-1}$ is the pivotal of successful implementation of Subset Simulation. This can be achieved using MCMC techniques. In the following subsections, we briefly review Subset Simulation for reliability analysis and structural optimization, and also give an overview of MCMC methods.

2.1 Subset simulation for reliability analysis

For reliability analysis, Subset Simulation starts with DMC in the first step. The probability P_1 associated with the first intermediate event F_1 is estimated as

$$P_1 = P(F_1) \approx \frac{1}{N}\sum_{i=1}^{N} I_{F_1}(g(\mathbf{x}_i)) \tag{4}$$

where N is the number of samples in the first simulation level, $\{\mathbf{x}_i\}$ are independent and identically distributed (i.i.d.) samples generated according to the probability density function (PDF) $f(\mathbf{x})$ and $I_{F_1}(\cdot)$ is the indictor function

$$I_{F_1}(\cdot) = \begin{cases} 0 & \text{if} \quad g(\mathbf{x}_i) \geq b_1 \\ 1 & \text{if} \quad g(\mathbf{x}_i) < b_1 \end{cases} \tag{5}$$

In (5), however, b_1 is unknown, and so is the first intermediate event F_1. If a fixed value p_0 is set to P_1, one can employ (4) to determine the value of b_1 and the first intermediate event F_1. After generating $\{\mathbf{x}_i\}$, calculate the system response function $\{g(\mathbf{x}_i)\}$ for all i, and sort them in an ascending order such that $g(\mathbf{x}_1) \leq g(\mathbf{x}_2) \leq \cdots \leq g(\mathbf{x}_N)$. Let b_1 be the sample P_1-quantile of the system response, i.e., $b_1 = g\left(\mathbf{x}_{[P_1 N]}\right)$. Herein, the symbol $[\cdot]$ in the subscript denotes to round the argument. In this way, samples $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{[P_1 N]}$ belong to the first intermediate event F_1, and they automatically satisfy (4).

The subsequent conditional probabilities $P_j = P(F_j|F_{j-1})$ require the samples conditioning on F_{j-1} with implicit conditional PDF

$$f\left(\mathbf{x}\middle|F_{j-1}\right) = f(\mathbf{x}) I_{F_{j-1}}(\cdot)/P\left(F_{j-1}\right) \tag{6}$$

One may generate conditional samples based on (6). However, this method is in general inefficient because the acceptance probability of sample is proportional to the inverse of $P(F_{j-1})$. This means that it would reject about $1/P(F_{j-1})$ samples before obtaining one proper conditional samples on average. It is still a DMC estimator for $P(F_j|F_{j-1})$, being consistent with the estimator in (4). Given that one already have $[NP_{j-1}]$ samples belonging to F_{j-1}, a simulation procedure based on MCMC can be employed to obtain the required conditional samples $\{\mathbf{x}_i\}$ and then the estimator for $P(F_j|F_{j-1})$

$$P_j = P\left(F_j \middle| F_{j-1}\right) \approx \frac{1}{N}\sum_{i=1}^{N} I_{F_j}(g(\mathbf{x}_i)) \tag{7}$$

where $\mathbf{x}_i \sim f(\mathbf{x}|F_{j-1}), i=1,\ldots,N$ are generated by a modified Metropolis-Hasting type MCMC (Au and Beck 2001), which has been proved that its stationary distribution is equal to the desired conditional distribution. We will discuss this MCMC approach in detail in subsection 2.3. After generating new $N-[NP_{j-1}]$ conditional samples in F_{j-1} and combining the previously selected $[NP_{j-1}]$ samples, we can compute the system response function $\{g(\mathbf{x}_i)\}$ for all i, and sort them in an ascending order. Let b_j be the sample P_j-quantile of N system responses in F_{j-1}, i.e., $b_j = g\left(\mathbf{x}_{[P_j N]}\right)$. In this way $\left\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{[P_j N]}\right\}$ belong to the next intermediate event F_j, and are guaranteed to satisfy (7) for some fixed P_j.

Repeat the above procedure until the sample P_j-quantile of N system responses in F_{j-1} is less than b, i.e., $b_j = g\left(\mathbf{x}_{[P_j N]}\right) < b$. Then, by this point, the target failure domain F_m is already arrived by the algorithm, i.e., $j=m$ and $b_m = b$. The estimator for the last conditional probability $P(F_m|F_{m-1})$ is obtained as

$$P(F_m|F_{m-1}) \approx P_m = \frac{1}{N}\sum_{i=1}^{N} I_{F_m}(g(\mathbf{x}_i)) \tag{8}$$

Combining (4), (7) and (8), the failure probability of the target event is expressed as

$$P_F = \prod_{j=1}^{m} P_j \tag{9}$$

In practical engineering applications of Subset Simulation, a typical value of P_j, $j=1,\ldots,m-1$ is some fixed $p_0 \in 0.1 \sim 0.3$ (Zuev et al. 2012).

The following pseudo-code implements Subset Simulation for reliability analysis.

```
j=1
For i = 1 to N
    Randomly generate x_i
    Compute the response function value g(x_i)
End
Sort x_i (i=1,…, N) according to g(x_i) (i=1,…,N)
Obtain the first Np_0 samples from the ascending sequence and the threshold value b_1
j = 2
Do
    l = [1/p_0]
    For i = 1 to N p_0
        For k= 1 to l
            Obtain a new conditional sample using the MMH algorithm
        End
    End
    Sort x_i (i=1,…, N) according to g(x_i) (i=1,…,N)
    Obtain the first Np_0 samples from the ascending sequence which belong to F_j and the
    threshold value b_j
    j = j+1
While ( g(x_[Np_0]) > 0 )
```

2.2 Subset simulation for structural optimization

Consider solving the following unconstrained optimization problem

$$\min_{\mathbf{d}^L \le \mathbf{d} \le \mathbf{d}^U} W(\mathbf{d}) \tag{10}$$

where $W(\mathbf{d})$ is the objective function, $\mathbf{d}$ is the design vector that contains design variables, $\mathbf{d}^L$ and $\mathbf{d}^U$ are the lower bound and upper bound for the design vector. The analogy between an optimization problem and a reliability problem is presented firstly, which allows us to solve an optimization problem potentially using reliability analysis methods (Li and Au 2010; Li 2011). The aim of reliability analyses is to evaluate the probability of a target failure event F, while the aim of an optimization problem is to search a point or region where the objective function is minimized under the problem setting in (10), i.e., taking extreme values. We employ a one-dimensional example, as shown in Fig. 1, to explain this analogy. In this illustrative example, only one variable d is involved, and h is a function of d. The optimization problem is defined as finding the minimal value of h in Fig. 1a, i.e., $\min h(d)$. By augmenting the design variable d to be a random variable, a reliability problem can also be defined to estimate the probability of h less than a given threshold h_0, as shown in Fig. 1b. This means that the target event is $F=\{h<h_0\}$ and the corresponding probability is $P_F=P(h<h_0)$. It is well-known that engineering reliability problems are often rare event simulation problems since very small failure probabilities are generally involved in practice. Geometrically, the region of $F=\{h<h_0\}$ of a reliability problem absolutely covers the minimum points of an optimization problem. In other words, the minimum points to be found of an optimization problem is a reduced region of a reliability problem. Therefore, we can deal with optimization problems under the framework of the reliability analysis because an extreme event is viewed as a special case of a rare event concerned in reliability analyses.

Based on the analogy between an optimization problem and a reliability problem, we can construct an artificial reliability problem by randomizing the design variables, i.e.,

$$P_F = P(F) = P\left(W(\mathbf{d}) \le W_{opt}\right) \tag{11}$$

where W_{opt} is the minimum value of the objective function, $F=\{W(\mathbf{x})\le W_{opt}\}$ is the artificial target event, and

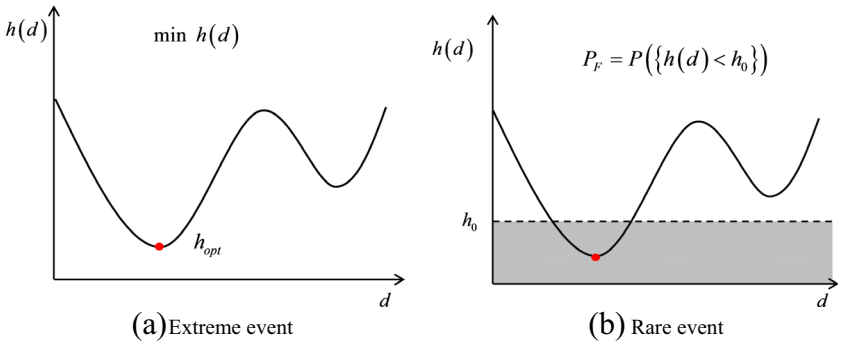

Fig. 1 The analogy between an optimization problem and a reliability problem

P_F is the corresponding failure probability of F. It is clear that P_F has a zero value because W_{opt} is the minimum value of the objective function. However, in an optimization problem, we are interested in the point $\mathbf{d}_{opt}$, where the zero failure probability can be attained, rather than the zero failure probability itself. Since the design vector $\mathbf{d}$ is augmented to be random, the type of probability distribution of design vector may have influences on the performance of the transformation in (11). Experiences shows that the truncated normal distributions is sufficient and convenient to characterize design variables themselves and their respective bounds, as suggested by Li and Au (2010); Li (2011).

Equation (11) maps a multidimensional objective function $W(\mathbf{d})$ into a random variable W. According to the definition of cumulative distribution function (CDF) for a random variable, it is a monotonic, non-decreasing, and right-continuous function, such that it displays values lying in the interval of $[0, 1]$. Therefore, the CDF value at W_{opt} is zero. This is consistent with the zero failure probability discussed above. It is worthwhile to point out that, based on the transformation in (11), local optimums can be avoided, at least, from a theoretical perspective view. The objective function may be a many-to-one relationship between multiple design variables and itself, while the CDF function of W is a one-to-one relationship after mapping manipulation.

In Subset Simulation for structural optimization, the governing equation is still given by (3). Its implementation starts with the initialization of distributional parameters for the design vector $\mathbf{d}$ in the first step. The following steps for structural optimization are similar to these for reliability analyses described in subsection 2.1. The optimization iterations are not terminated until a stopping criterion is reached. Two groups of stopping criterion have been developed in the literature. The idea of first group stems from other stochastic optimization method, employing a maximum iteration number or the difference of objective function values between two consecutive iterations smaller than a specified tolerance to enhance for convergence. While the idea of the other group is based on sample statistics, the stopping criterion is defined as (Li and Au 2010; Li 2011)

$$\max\left(\hat{\sigma}_k\right) \quad or \quad \max\left(\left|\hat{\sigma}_k - \hat{\sigma}_{k-1}\right|\right) \le \varepsilon \tag{12}$$

where $\hat{\sigma}_k$ is the estimator of standard deviation of the samples in the $k-th$ simulation level and ε is the specified tolerance.

2.3 Markov Chain Monte Carlo

The main challenge for implementing Subset Simulation is to generate conditional samples in (7) for the estimation of conditional probability. Difficulties arise because the conditional PDF $f(\cdot|F_j)$ for an intermediate event has an implicit expression. Markov Chain Monte Carlo (MCMC) is a stochastic simulation technique for generating samples from an arbitrary PDF, and it provides a powerful tool to generate conditional samples in the implementation of Subset Simulation.

The working principle of MCMC is that starting with an arbitrary sample $\mathbf{x}_0$, a Markov Chain is generated using a transition kernel whose stationary distribution is equal to the desired distribution. Therefore, the most important component of MCMC is the transition from current state sample $\mathbf{x}_i$ to the next state $\mathbf{x}_{i+1}$ of the chain. MCMC method was originated from (Metropolis et al. 1953), and was further modified by Hastings to allow nonsymmetrical proposal PDF (Hastings 1970). Today, the Metropolis-Hastings (M-H) algorithm is still the standard implementation of MCMC because it imposes minimal requirements on the desired distribution. The transition $\mathbf{x}_i \rightarrow \mathbf{x}_{i+1}$ in an M-H algorithm is achieved by two steps. First, a candidate state $\mathbf{x}'$ is generated from a proposal PDF $q(\cdot|\mathbf{x}_i)$ which describes how to go from $\mathbf{x}_i$ to $\mathbf{x}_{i+1}$ and is easier to simulate. Here, it is clear that the proposal PDF is a conditional PDF, however, some variants of the standard version may not require this constraint. Second, the candidate state $\mathbf{x}'$ is either accepted or rejected with a certain acceptance probability $\rho(\mathbf{x}, \mathbf{x}')$. These two steps can be viewed as a local random walk in the neighborhood of the current state $\mathbf{x}_i$. For the use in Subset Simulation, there is another step to make sure that the accepted candidate $\mathbf{x}'$ also lies in the current intermediate failure domain F_{l-1}. If $\mathbf{x}' \in F_{l-1}$, $\mathbf{x}'$ is accepted as the next state; otherwise, $\mathbf{x}'$ is rejected and $\mathbf{x}_i$ is repeated in the Markov Chain. This step ensures the correct conditioning in the samples. The final acceptance probability is the product of acceptance probabilities in the last steps, i.e., $\rho(\mathbf{x}, \mathbf{x}') \cdot I_{F_{l-1}}(\mathbf{x}')$.

From above three steps, it can be seen that the MCMC samples are not statistical independent because repeated states occur in a Markov Chain. However, the ergodic theorem can guarantee that statistical inference based on the MCMC samples with stationary distribution $f(\cdot|F_{l-1})$ converges to the one based on i.i.d. samples from $f(\cdot|F_{l-1})$.

Typically, a Markov Chain starting with an arbitrary sample $\mathbf{x}_0$ may has the burn-in issue for assessing the convergence of a Markov Chain, i.e., the first few samples generated by the M-H algorithm cannot be used for statistical inference. However, the nested setting of Subset Simulation perfectly avoids this issue. Since the seed samples for all Markov Chains follow the desired distribution $f(\cdot|F_{l-1})$, all the subsequent samples on the same Markov Chains naturally follow the same desired distribution. This technique is termed perfect simulation in the literature (Robert and Casella 2004).

Another issue associated with the standard M-H type MCMC is 'the curse of dimension'. As the dimension of problem increases, the acceptance probability $\rho(\mathbf{x}, \mathbf{x}')$ in the second step exponentially decreases to zero. For a high dimension n, the candidate $\mathbf{x}'$ may be always rejected, leading to a large number of repeated samples. Au and Beck (Li and Au 2010); Au and Wang (2014) have examined the issue in detail. They also proposed a component-wise scheme to overcome the high dimension issue and the degeneration of $\rho(\mathbf{x}, \mathbf{x}')$, which is attributed to the fact that $\rho(\mathbf{x}, \mathbf{x}')$ is a product of PDF ratios and tends to a small number as n increasing. In the modified Metropolis-Hastings (MMH) algorithm, one-dimensional proposal PDF replaces the n-dimensional proposal PDF so that the acceptance probability of a component in the random vector is only a ratio of one-dimension PDFs. This scheme dramatically decreases the probability of having repeated values simultaneously for all the components during the simulation, making it feasible in high dimension problems.

After solving the high-dimensional issue, the next implementation issue may be the choice of the type of one-dimensional proposal PDF and the spread around the current sample since they seem to control the simulation efficiency of MCMC. Simulation experiences show that the efficiency of the MMH algorithm is not sensitive to the type of proposal PDF, while the spread of the one-dimensional proposal PDF is important for the balance of efficiency and robustness. Using large spreads lead to the reduction of the acceptance probability, repeated component samples and slow convergence. In contrast, using small spreads introduces large dependence among component samples due to their proximity and also slows down convergence. Au and Beck (2001) suggested an adaptive manner which uses some fraction of sample standard deviation calculated from all samples or only the $[p_0N]$ seed samples generated in the previous simulation level. Chiachio et al. (2014a) also provided a sensitivity analysis and proposed to adaptively choose the variance of the jth intermediate level so that the monitored acceptance rate belong to the interval [0.2, 0.4].

3 Matlab implementation

The Matlab codes given in Appendixes 1 and 2 can be used to solve reliability analysis problems and structural optimization problems, respectively. Both Matlab implementations for reliability analysis and structural optimization are written as Matlab build-in functions so that they are considered as standard codes and can be easily used without the need to be a Matlab expert.

3.1 Subset simulation for reliability analysis

In this section, we explain the usage and important details of the 78-line Matlab code in Appendix 1 for reliability analysis. The code is divided into three parts: (1) input parameter definition, (2) DMC, and (3) MCMC, and can be used as a standard code of Matlab. Modification is not needed for different applications. In the following subsections, detailed explanations are given through a high-dimensional example with analytical solution.

3.1.1 Function arguments and outputs

For simplification, the function name is shorten as SS, and the SS function is called from the Matlab command prompt by the line

```
[pfss, gRe, cdf, zRe] = SS(FUN, n, paras)
```

where `FUN` is a Matlab function handle which is used to calculate limit state values of interest, `n` is the dimension of the problem, *paras* is the collection of all algorithm parameters, `pfss` is the estimator of failure probability, `gRe` is a record of limit state values that have been used in Subset Simulation, `cdf` is the corresponding CDF values to limit state values stored in `gRe`, and `zRe` is the record of samples corresponding to limit state values stored in `gRe`.

The data type of SS function arguments have been specified and cannot be overridden. Argument `FUN` is a Matlab function handle which is bound to the Matlab function for limit state functions and argument `n` is a specified value. Argument `paras` is a structure data type with three fields: `NumSam`, `ConPro` and `MaxTry`. Field `NumSam` is the number of samples in each simulation level. Field `ConPro` is the conditional probability and Field `MaxTry` is the maximum number of simulation levels which is used to terminate the simulation procedure so as to avoid infinite loop when the target domain cannot be reached. It should be pointed out that Subset Simulation for reliability analysis has only two algorithm parameters, i.e., `NumSam` and `ConPro`, while `MaxTry` is added just for avoiding infinite loop.

To illustrate the use of SS function, consider estimating the failure probability of the following limit state function

$$h(\mathbf{x}) = b - \frac{1}{\sqrt{n}} \sum_{i=1}^{n} \mathbf{x}_i \tag{13}$$

where $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ are i.i.d. standard normal variables, and b is a constant value. It can be easily reasoned that the target failure region is a linear half-space, i.e., $F = \{h(\mathbf{x}) < 0\}$, and h is a normal variable so that the failure probability is analytically given by

$$P_F = P(h(\mathbf{x}) < 0) = \Phi(-b) \tag{14}$$

where $\Phi(\cdot)$ is the CDF of standard normal distribution. It is obvious that (14) is independent of the problem dimension n. In the case of $b = 3$ and $n = 1000$, the exact failure probability is $\Phi(-3) = 1.35 \times 10^{-3}$. The following lines represent the evaluation of (13) in Matlab.

```
n = 1000;
b = 3;
hx =@(x)b-sum(x)/sqrt(n);
```

For this problem, the arguments for SS function are initialized as follows:

```
def = struct('NumSam',300,'CondPro',0.1,'MaxTry',10);
```

This line means that the number of samples in each simulation level is 300, the conditional probability is taken as 0.1 and the maximum number of simulation level is equal to 10. Then, this problem is solved by calling with the input line

```
[pfss, gre,cdf, Zre] = SS(hx, n, def);
```

Finally, the calculated results are stored in `pfss`, `gre`, `cdf` and `zre`.

3.1.2 Algorithm parameter definition (lines 3–6)

Symbol `N` is used to represent the number of samples in each simulation level and it receives the passing value from `para.NumSam`. Symbol `p0` denotes the conditional probability and it receives the passing value from `para.ConPro`. Symbols `nt` and `ns` are used for the sample quantile of system responses and the length of Markov Chain, respectively. Note that the product of `N` and `p0` may not be an integer. A built-in function in Matlab, i.e., `round()`, is employed to round it to the nearest integer (line 5). The similar operation is carried on the length of a Markov Chain (line 6).

3.1.3 Direct Mont Carlo simulation (lines 7–20)

DMC simulation starts with generating samples from standard normal distribution (line 8). Then, the system responses for these samples are evaluated using the evaluation function `feval()` in Matlab (line 11). Lines 14–15 are the Matlab implementation of sorting and determining the sample quantile of the system response. Lines 16–20 represent the record of SS function outputs.

3.1.4 Markov Chain Monte Carlo (lines 21–78)

Lines 23–25 are some basic preparation for the following MCMC simulation procedure. As mentioned before, the spread of proposal distribution is estimated from the seed samples (line 23).

The Matlab code for MCMC is divided into four parts: (1) selecting seed samples (lines 27–28), (2) generating conditional samples by the MMH (lines 31–62), (3) storing and recording the calculated results (lines 64–69), and (4) terminating the simulation procedure (lines 71–77).

Variables `w` and `g` denote the seed samples and their corresponding system responses, and are used to generate the conditional samples. Line 31 is a counting command beginning with 1 since the first nt samples are selected from the previously simulation level. Additional $N - nt$ conditional samples are supplied to keep the number of samples in a simulation level constant. Lines 34–47 represent the implementation of the first two steps in the MMH algorithm with component-wise strategy. A loop over all components is performed here (lines 34–47). The system response at a candidate sample is calculated in line 48. Then, comparing it with the threshold value `gSort(nt+1)` (line 50), the corresponding candidate is accepted if it is less than the threshold value (line 51). Otherwise, a repeated sample appears in the current Markov Chain (lines 53–54). The sampling center is updated to the current state in line 56. It should be noted that there is an inner loop over `ns` samples in a Markov Chain starting in line 33 and an outer loop over `nt` Markov Chains starting in line 31. The same sorting and recording operations as in DMC stage are carried out in lines 64–69. Lines 71–77 are used to terminate the main loop if the threshold value is less than zero, or

the number of iteration is larger than the maximum permissible number stored in `MaxTry`.

For completeness, the estimator of failure probability of (13) is 2.6×10^{-3}, which is stored in the variable `pfss`, using the following prompt lines:

```
1  clear all;
2  seed = 10;
3  rand('twister',seed);
4  randn('state',seed+1);
5  def = struct('NumSam',300,'CondPro',0.1,'MaxTry',10);
6  n = 1000;
7  b = 3;
8  hx =@(x)b-sum(x)/sqrt(n);
9  [pfss, gre,cdf, zre] = SS(hx, n, def);
```

The purpose of lines 2–4 is to set a fixed starting point for the random number generator in Matlab so that one can reproduce the calculated results reported here. The other results, such as the CDF values stored in variable `cdf` and the corresponding values of LSF stored in variable `gre`, can be converted into a CDF plot of $h(x)$, which shows the probability of $h(x)$ smaller than a threshold value h_0, i.e., $p(h<h_0)$. Figure 2 shows such a CDF plot for this problem by using the following prompt lines:

```
10  semilogy(gre,cdf,'-k','LineWidth',2)
11  xlabel('\ith')
12  ylabel('CDF')
```

3.2 Subset simulation for structural optimization

In this section, we explain the usage and important details of the 98-line Matlab code in Appendix 2 for structural optimization. Similar to the code for reliability analysis shown in Appendix 1, the code also has three parts. We concentrate on the differences between them.

3.2.1 Function arguments and outputs

For simplification, the name of Matlab function for structural optimization is shorten as SSO. The SSO function can be simply called by the line.

```
[minimum,solu,numTol,gRe,zRe] = SSO(objFun, n, boundaries, paras)
```

Compared with SS function, one argument, i.e., `boundaries`, is added to SSO function to pass the boundaries of design variables to the main program. In the structure `paras`, a new field `Tol` is added to define a specified tolerance for the termination of simulation procedure. For the function outputs, `minimum` and `solu` are used for storing the optimal value of objective function and the corresponding solution vector, respectively. The total number of the objective function evaluation is denoted by `numTol`. Function outputs `gRe` and `zRe` are employed to store the history of the optimal value of objective function and its corresponding solution vector.

To illustrate the usage of SSO function, consider the problem of minimizing the six-hump camel-back function

$$\min h(x) = 4x_1^2 - 2.1x_1^4 + x_1^6/3 + x_1x_2 - 4x_2^2 + 4x_2^4 \quad \text{s.t. } \mathbf{x}\in[-3,3]$$

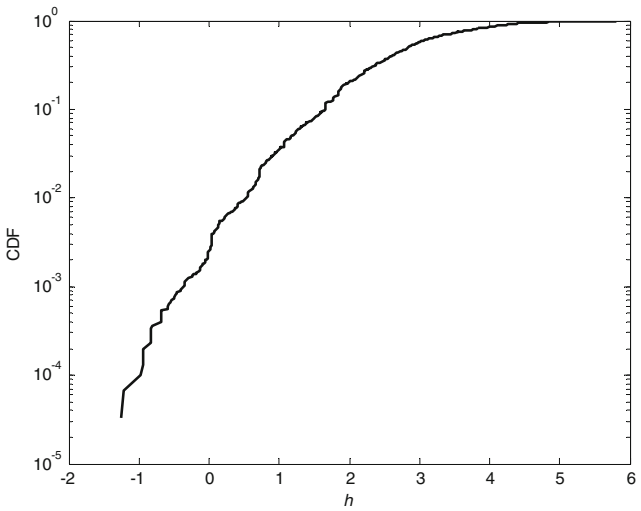

Fig. 2 CDF plot for high dimensional problem

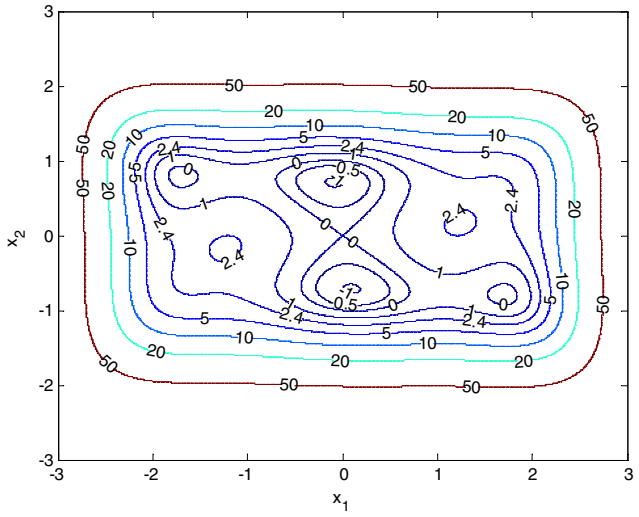

Fig. 3 The six-hump camel-back function

In the bounded region, this function has six minima, while four of six minima are local ones (Fig. 3). The global minimum is -1.0316, and the corresponding solutions are $\mathbf{x}^* = (-0.0898, 0.7126)$ and $\mathbf{x}^* = (0.0898, -0.7126)$.

3.2.2 Algorithm parameter definition

The distributional parameters for design variables are written in lines 8–9 after they are augmented to be truncated normal variables.

3.2.3 Direct Monte Carlo simulation

Lines 13–19 represent the sampling method for truncated normal variables.

3.2.4 Markov Chain Monte Carlo

Compared with its counterpart in Appendix 1 (line23), a slight modification is made in line 33, calculating the standard deviation using all samples in a simulation level instead of the first Np_0 samples. This leads to a larger spread of the proposal distribution in Subset Simulation for structural optimization.

Since the design variables have boundaries in the parameter space, lines 47–53 are added to ensure that the generated candidate satisfies those constraints. In the MMH code, two PDF calculators are revised according to the definition of truncated distribution (lines 56–57). For structural optimization, the simulation procedure stops when the maximum sample standard deviation (line 83 and line 89) is less than the specified tolerance stored in `paras.Tol` or the number of simulation levels is greater than the specified maximum number stored in `paras.MaxTry`. At the end of the Matlab code, lines 97–98 determine the minimum of the objective function and the corresponding solution.

For the completeness of the illustrated example, we performed Subset Simulation to solve the problem shown in Fig. 3, and the input lines are

```
1   clear all
2   ranseed = 10;
3   rand('twister',ranseed);
4   randn('state',ranseed+1);
5   hx =
@(x)(4-2.1*x(1).^2+x(1).^4/3)*x(1).^2+x(1).*x(2)+(-4+4*x(2).^2)*x(2).
^2;
6   def = struct('NumSam',200,'CondPro',0.5,'MaxTry',100,'Tol',1e-4);
7   b1 = -3*ones(2,1);
8   b2 = 3*ones(2,1);
9   b = [b1,b2];
10  [min,solu,nt,gre,zre] = SSO(hx, 2, b, def);
11  plot(gre,'-k','LineWidth',2)
12  xlabel('The number of simulation levels')
13  ylabel('Objective function')
```

The optimization history is given in Fig. 4. After 28 simulation levels, SSO function finds a minimum of -1.0136 with 2900 function evaluations. However, this minimum value is achieved at the 13th simulation level and after that this value does not change. This is caused by the adoption of sample statistics, i.e., sample standard deviation, for termination.

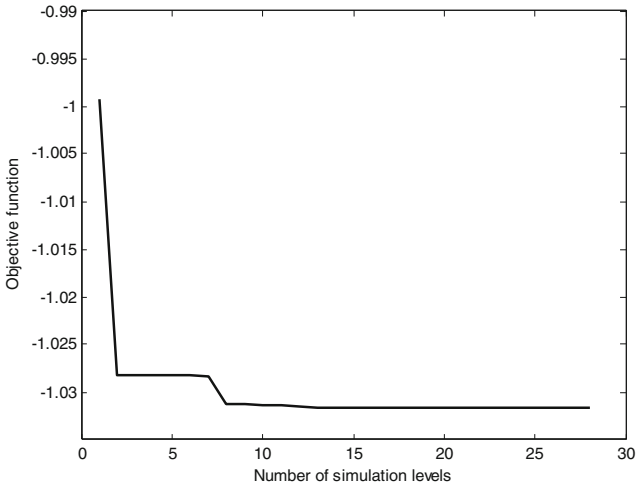

Fig. 4 The optimization history of the six-hump camel-back function

4 Extensions

A number of extensions in the algorithms and codes can be thought of, a few of which are described in the following subsections.

4.1 Argument check

In order to make the code in Appendix 1 operate like a 'professional' function in Matlab, the following additional lines may be added just after the definition of SS function (line 1). These lines provide default parameter setting and throw errors and prompts for the user. For SSO function, it can be done by making small changes to the above lines.

4.2 Post-processing

It is very simple to extend the code to provide some basic post-processing operations, such as the CDF plot for reliability analysis. In fact, this can be done by adding three additional lines just after line 78.

```
79 semilogy(gRe,cdf,'-k','LineWidth',2)
80  xlabel('\ith')
81  ylabel('CDF')
```

For structural optimization, the following three lines draw a history of optimization.

```
99  plot(gRe,'-k','LineWidth',2)
100 xlabel('The number of simulation levels')
101 ylabel('Objective function')
```

4.3 Alternative stop criteria for structural optimization

The stop criterion here is based on a combination of the sample standard deviation and the maximum number of simulation levels. Two alternative stop criteria can be adopted. The first one is based on the idea that when the CDF of the objective function approaches zero, the sequence of objective function also approaches its minimum point. Thus, the line 90 in Appendix 2 is changed to

```
90  if pfss < paras.Tol || iter >= paras.MaxTry
```

The second stop criterion is based on the difference between the minima of objective function in two consecutive simulation levels.

```
90  if abs(gRe (1)- gRe(2)) < paras.Tol || iter >= paras.MaxTry
```

4.4 Constraint-handling for structural optimization

Admittedly, constrained optimization algorithms are more preferable to unconstrained ones because most of optimization problems in engineering and science fields have a variety of restriction. It is also quite convenient to extend the SSO algorithm to account for multiple general constraints. Here, consider the following constrained optimization problem

$$\begin{array}{ll} \text{Min} & W(\mathbf{d}) \\ s.t. & g_i(\mathbf{d}) < 0 \quad i = 1, \ldots, n_e \end{array} \tag{15}$$

where n_e is the number of constraints. In order to include the effects of constraints, a constraint fitness function is defined as

$$F_{con}(\mathbf{d}) = -\max_i \mathbf{v}_i \tag{16}$$

where $\{\mathbf{v}_i\}$ are the violations of constraints, which are given by

$$\mathbf{v}_i = \begin{cases} 0 & , \quad g_i(\mathbf{d}) < 0 \\ g_i(\mathbf{d}) & , \quad g_i(\mathbf{d}) > 0 \end{cases} \tag{17}$$

Four lines `Fitness` function, a sub-function of SSO function, can be easily added to realize this definition.

```
99  %% Constraint fitness function
100 function [objF,consF] = Fitness(fun,d)
101 [objF,cons] = feval(fun,d);
102 consF = min(-max(cons),0);
```

In the case of considering constraints, the sorting sequence is not unique, which is very important to provide seed samples for the next simulation level. Li and Au proposed a double-criterion ranking method to solve this issue successfully. Please refer to Ref. (Li and Au 2010) for more details about this method. The corresponding code is written in lines 103–112, just appended to SSO function as a subfunction.

```
103 %% Double-criterion ranking function
104 function [g1sort, g2sort, index] = DSort(g1,g2)
105 [g2sort,index] = sort(g2,'descend');
106 g1sort = g1(index);
107 feRegion = find(g2sort == 0);
108 if ~isempty(feRegion)
109    [g1sort(feRegion),subIndex] = sort(g1sort(feRegion));
110    g2sort(feRegion) = g2sort(subIndex);
111    index(feRegion) = index(subIndex);
112 end
```

Both the objective function and the constraint fitness function now are calculated using Fitness subfunction so that line 22 and line 65 are substituted by the following lines.

```
22  [g(i),cons(i)] = Fitness(objFun,z(i,:));
65  [gTemp,consTemp] = Fitness(objFun,w(len,:));
```

To sort the samples considering the objective function and the constraint fitness function, both line 25 and 81 become

```
25  [gSort,consSort, index]=DSort(g,cons);
81  [gSort,consSort,index]=DSort(g,cons);
```

To generate conditional samples, the constraint fitness function must be induced in the MMH algorithm, which means the lines 67–70 are changed to

```
67  if gTemp <= gSort(nt+1) && consTemp >= consSort(nt+1);
68     g(len) = gTemp;cons(len) = consTemp;
69  else
70     g(len) = g(i);cons(len) = cons(i);
```

5 Practical use

These two codes can be conveniently applied by users in practice. In this section, two applications are considered, one for structural reliability analysis and one for structural optimization design. Since the purpose of two applications is to illustrate the applications of Subset Simulation, we do not focus on comparing it with other methods for accuracy or efficiency.

5.1 Structural reliability analysis

The one-bay one-storey frame is taken from Ref.(Bucher 2009) and is concerned with its collapse subjected to static loads, including a horizontal load P_1 and a vertical load P_2, as shown in Fig. 5. Both P_1 and P_2 are random and normally distributed. The failure of the frame is defined by the first-order rigid-plastic hinge theory, and then three dominant collapse mechanisms can be identified, as shown in Fig. 5 as failure modes. The three failure modes are given by the following relations

$$h_1(X) = 4\frac{M_{pl}}{L} - P_1 \tag{18}$$

$$h_2(X) = 4\frac{M_{pl}}{L} - P_2 \tag{19}$$

$$h_3(X) = 6\frac{M_{pl}}{L} - P_1 - P_2 \tag{20}$$

where M_{pl} is the plastic moment, which is a deterministic quantity. The frame fails when any one of these three collapse mechanisms occurs. This means that it is a series system, i.e., the LSF of this frame $h(x) = -\min(h_1(x), h_2(x), h_3(x))$. Assuming that the mean value of

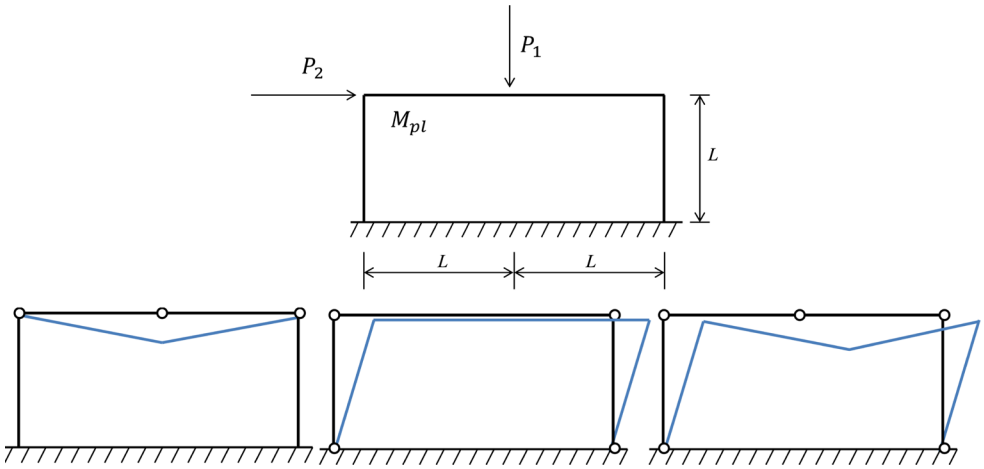

Fig. 5 One-bay one-storey frame structure and failure modes

P_1 and P_2 is $1.7\frac{M_{pl}}{L}$ and their coefficient of variation is 0.5. The above LSF is transferred into standard normal space, as shown in Fig. 6.

Based on the above given information, the following input lines are used to perform structural reliability analysis of this frame structure.

```
clear all;
seed = 10;
rand('twister',seed);
randn('state',seed+1);
def = struct('NumSam',500,'CondPro',0.1,'MaxTry',10);
n = 2;
[pfss, gre ,cdf,Zre] = SS('frame', n, def);
```

The CDF plot obtained from the code is shown in Fig. 7. Furthermore, the failure probability given by Subset Simulation is 5.660×10^{-5} with 2,300 samples, which can be known by checking the size of `gre` in the command window.

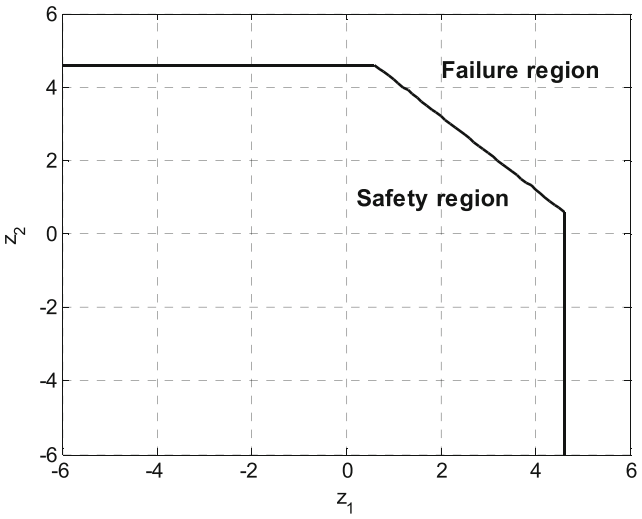

Fig. 6 Limit state in the standard normal space

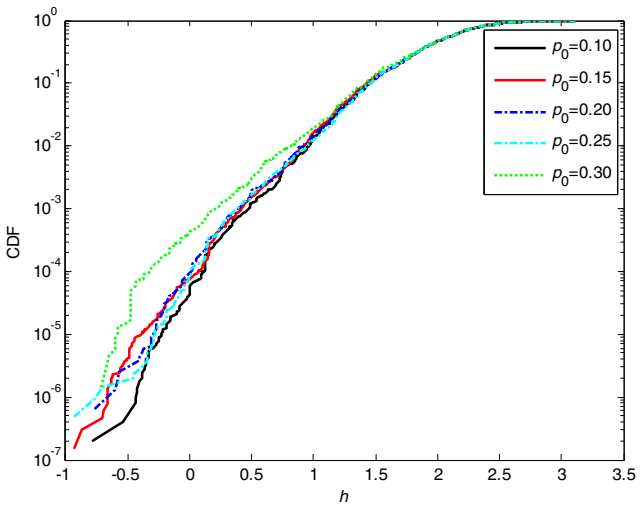

Fig. 7 CDF plot for the one-bay one-storey frame problem

The CDF plots for p_0=0.15, 0.20, 0.25 and 0.30 are also given in Fig. 7. The estimated failure probabilities and the number of required samples are listed in Table 1. The purpose of this comparison is to see how Subset Simulation behaves differently by adopting different values of p_0 for the example provided here. It seems the CDF plot with $p_0=0.3$ significantly deviates from the rest. As reported in Ref.(Zuev et al. 2012), the values of p_0 selected from the interval [0.1, 0.3] would produce similar efficiency with similar accuracy. It should be pointed out that this conclusion was drawn based on the specified values of target failure probability and the total number of samples. In fact, the practical interval of p_0 may be narrower or larger than that reported by Zuev et al. (2012). In other words, the optimal value of p_0 is still problem-dependent somewhat although they provided a theoretical basis for the selection of p_0. For the currently investigated problem, $p_0=0.1$, 0.15, 0.20 and 0.25 are suitable for reliability analysis, while $p_0=0.3$ is not proper.

5.2 Structural design optimization

Because constrained optimization problems are more common than unconstrained ones in engineering practice, we provide an example with 4 constraints in this section. It should be noted that, in order to solve this problem, the code in Appendix 2 for structural optimization must include all the extension changes described in the subsection 4.4 entitled "**Extensions**".

Consider the tension-compression string design problem taken from Ref. (Coello Coello 2000), as shown in Fig. 8. The objective is to minimize the string weight under constraints on deflection, shear stress, surge frequency, limits on outside diameter and on design variables. There are three design variables in this problem: the wire diameter d, the mean coil diameter D, and the number of active coils P, i.e., the design vector $[x_1, x_2, x_3]=[d, D, P]$ (Fig. 8).

Table 1 Estimated failure probabilities and the number of samples with different p_0

p_0	0.10	0.15	0.20	0.25	0.30
Failure probability (10^{-4})	0.5660	0.7563	0.9280	0.7861	4.3303
Number of samples	2300	2625	2500	2750	2600

Fig. 8 The tension-compression string design problem

The problem can be formulated as

$$\min f(\mathbf{x}) = (x_3+2)x_2x_1^2 \tag{21}$$

Subject to

$$g_1(\mathbf{x}) = 1-\frac{x_2^3x_3}{71785x_1^4}\le 0 \tag{22}$$

$$g_1(\mathbf{x}) = \frac{4x_2^2-x_1x_2}{12566\left(x_2x_1^3-x_1^4\right)}+\frac{1}{5108x_1^2}-\le 0 \tag{23}$$

$$g_3(\mathbf{x}) = 1-\frac{140.45x_1}{x_2^2x_3}\le 0 \tag{24}$$

$$g_4(\mathbf{x}) = \frac{x_2+x_1}{1.5}-1\le 0 \tag{25}$$

$$0.05\le x_1\le 2.0, 0.25\le x_2\le 1.3, 2.0\le x_3\le 15.0 \tag{26}$$

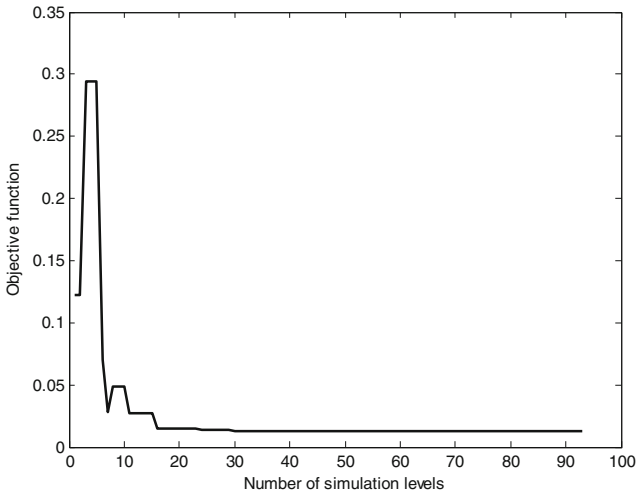

Fig. 9 The optimization history of the tension-compression string design problem

The objective function and all constraints are calculated by a Matlab function named `tcsp`.

```
function [obj, cons] =tcsp(x)
obj = x(1)^2*x(2)*(x(3)+2);
cons(1) = 1-(x(2)^3*x(3))/(71785*x(1)^4);
cons(2) =
(4*x(2)^2-x(1)*x(2))/(12566*x(1)^3*(x(2)-x(1)))+1/(5108*x(1)^2)-1;
cons(3) = 1-140.45*x(1)/(x(3)*x(2)^2);
cons(4) = (x(1)+x(2))/1.5-1;
```

Based the above information, the input lines for this optimization problem are

```
clear all
ranseed = 10;
rand('twister',ranseed);
randn('state',ranseed+1);
def = struct('NumSam',500,'CondPro',0.5,'MaxTry',100,'Tol',1e-4);
b1 = [0.05,0.25,2.0]';
b2 = [2.0,1.3,15]';
b = [b1,b2];
[min,solu,nt,gre,zre] = SSOC('tcsp', 3, b, def);
```

Figure 9 shows the optimization history of this problem. After 93 simulation levels, SSO finds a minimum of 0.0127 at design vector $[d, D, P] = [0.0528, 0.3849, 9.8076]$. The total number of function evaluation is 23,500. This optimization result is a feasible solution by re-examining all constraints' values.

6 Summary and conclusions

This paper presents two very simple Matlab codes of Subset Simulation for reliability analysis and structural optimization. The two codes for reliability analysis and structural optimization are comprised of 78 lines and 98 lines, respectively. The codes can be easily extended to include more considerations. For instance, constraint-handling method is included to deal with constrained optimization problems. We made attempts to organize the codes in the way of Matlab standard functions so that users can conveniently use them without modifications for their specific applications. This will be very helpful for the beginners in reliability engineering and structural optimization using Subset Simulation for their own applications. The usages of SS and SSO function are demonstrated through four numerical examples, including a high-dimensional LSF, structural reliability analysis of the one-bay one-storey frame, minimizing the six-hump, camel-back function, and structural optimization design of a tension-compression spring. To facilitate future communication with the readers who may be interested in this research, we have casted a website. Both codes and examples can be downloaded from the webpage: https://sites.google.com/site/rasosubsim/.

Acknowledgments The authors are grateful for the support by Fundamental Research Funds for the Central Universities (Project No.3082015NS2015007).

Appendix 1: subset simulation for reliability analysis

```
function [pfss, gRe, cdf, zRe] = SS(Fun, n, paras)
% Algorithm parameters
N = paras.NumSam;
p0 = paras.CondPro;
nt = round(N*p0);
ns = ceil(1/p0-1);
%% Step 1 (Monte Carlo simulation)
z = normrnd(0,1,N,n); % Generate standard normal samples
% Calculate LSF
for i=1:N;
    g(i) = feval(Fun,z(i,:));
end;
% Sort samples and LSF values
[gSort,index]=sort(g);
Z = z(index,:);
% Record the calculated results
pfss = nt/N;
gRe = gSort;
cdf = (N-[N:-1:1]+1)/N;
zRe = Z;
%% Step 2 (Markov Chain Monte Carlo)
% preparation for MCMC
sigma = std(Z(1:nt,:),0,1);
stopFlag = 0;
iter = 1;
while (stopFlag == 0)
    w = Z(1:nt,:);
    g = gSort(1:nt);
    iter = iter+1;
    len = nt+1;
    for i=1:nt
        seed = w(i,:);
        for j = 1:ns % A Markov chain
            for k = 1:n % Component by component
                % Generate an candidate
                u(k) = seed(k)+(2*rand()-1)*sigma(k);
                % Calculate the acceptance probability
                pdf2 = exp(-0.5*seed(k).^2);
                pdf1 = exp(-0.5*u(k).^2);
                alpha = pdf1/pdf2;
                % Accept u(k) with a probability of alpha
                if  alpha > rand();
                    w(len,k)= u(k);
                else
                    w(len,k)= seed(k);
```

```
            end;
        end
        gTemp = feval(Fun,w(len,:));
        % Accept or reject
        if gTemp <= gSort(nt+1);
            g(len) = gTemp;
        else
            g(len) = g(i);
            w(len,:) = seed;
        end
        seed = w(len,:);
        len = len+1;
        % terminate the simulation of a Markov chain
        if len > N break; end
        end
    if len > N break; end
end
% Sort samples and LSF values
[gSort,index]=sort(g);
Z = w(index,:);
% Record the calculated results
gRe = [gSort,gRe(nt+1:end)];
cdf = [(N-[N:-1:1]+1)/N*(nt/N)^(iter-1),cdf(nt+1:end)];
zRe = [Z;zRe(nt+1:end,:)];
% Terminate simulation
if iter >= paras.MaxTry || gSort(nt+1) <= 0
    pfss = pfss*(size(find(g<0),2)/N);
    stopFlag = 1;
    break;
else
    pfss = pfss*(nt/N);
end;
end;
```

Appendix 2: subset simulation for structural optimization

```
function [minimum,solu,numTol,gRe,zRe] = SSO(objFun, n, boundaries,
paras)
% Algorithm parameters
N = paras.NumSam;
p0 = paras.CondPro;
nt = round(N*p0);
ns = ceil(1/p0-1);
```

```
% Distributional parameters for truncated normal distribution
means = mean(boundaries,2);
stds = abs(boundaries(:,1)-boundaries(:,2))/6;
%% Step 1 (Monte Carlo simulation)
% Generate samples of truncated normal variables
z = [];
for i=1:n
    U(i) = normcdf(boundaries(i,2),means(i),stds(i));
    L(i) = normcdf(boundaries(i,1),means(i),stds(i));
    uu = unifrnd(0,1,N,1)*(U(i)-L(i))+L(i);
    tempz = norminv(uu,means(i),stds(i));
    z = [z,tempz];
end
% Calculate objective function
for i=1:N;
    g(i) = feval(objFun,z(i,:));
end;
% Sort samples and and objective function values
[gSort,index]=sort(g);
Z = z(index,:);
% Record the calculated results
numTol = N;
gRe = gSort(1);
zRe =Z(1,:);
%% Step 2 (Markov Chain Monte Carlo)
% preparation for MCMC
sigma = std(Z,0,1);
pfss = 1;
stopFlag = 0;
iter = 1;
while (stopFlag == 0)
    w = Z(1:nt,:);
    g = gSort(1:nt);
    iter = iter+1;
    len = nt+1;
    for i=1:nt
        seed = w(i,:);
        for j = 1:ns % A Markov chain
            for k = 1:n % Component by component
                % Generate an candidate
                done = false;
                while ~done
                    u(k) = seed(k)+(3*rand()-1)*sigma(k);
                    if u(k)>= boundaries(k,1) && u(k)<= boundaries(k,2)
```

```
                    done = true;
                end
            end
            % Calculate the acceptance probability
            pdf2 = exp(-0.5*((seed(k)-means(k))/stds(k)).^2);
            pdf1 = exp(-0.5*((u(k)-means(k))/stds(k)).^2);
            alpha = pdf1/pdf2;
            % Accept u(k) with a probability of alpha
            if  alpha > rand();
                w(len,k)= u(k);
            else
                w(len,k)= seed(k);
            end;
        end
        gTemp = feval(objFun,w(len,:));
        % Accept or reject
        if gTemp <= gSort(nt+1);
            g(len) = gTemp;
        else
            g(len) = g(i);
            w(len,:) = seed;
        end
        seed = w(len,:);
        len = len+1;
        % terminate the simulation of a Markov chain
        if len > N break; end
      end
    if len > N break; end
  end
  % Sort samples and objective function values
  [gSort,index]=sort(g);
  Z = w(index,:);
  sigma = std(Z,0,1);
  % Record the calculated results
  gRe = [gRe;gSort(1)];
  zRe = [zRe;Z(1,:)];
  numTol = numTol+N-nt;
  % Terminate simulation
  max_sigma = max(sigma);
  if max_sigma < paras.Tol || iter >= paras.MaxTry
      stopFlag = 1;
      break;
  else
      pfss = pfss*nt/N;
  end;
end;
minimum = gSort(1);
solu = Z(1,:);
```

References

Au SK (2005) Reliability-based design sensitivity by efficient simulation. Comput Struct 83(14):1048–1061

Au SK, Beck JL (2001) Estimation of small failure probabilities in high dimensions by subset simulation. Probab Eng Mech 16(4):263–277

Au SK, Beck JL (2003) Subset simulation and its application to seismic risk based on dynamic analysis. J Eng Mech ASCE 129(8):901–917

Au SK, Wang Y (2014) Engineering risk assessment with subset simulation. John Wiley & Sons

Au SK, Ching J, Beck JL (2007) Application of subset simulation methods to reliability benchmark problems. Struct Saf 29(3):183–193

Bourinet JM, Deheeger F, Lemaire M (2011) Assessing small failure probabilities by combined subset simulation and support vector machines. Struct Saf 33(6):343–353

Bucher C (2009) Computational analysis of randomness in structural mechanics. CRC Press

Chiachío M et al (2014a) Approximate Bayesian computation by subset simulation. SIAM J Sci Comput 36(3):A1339–A1358

Chiachío M, et al (2014) An efficient simulation framework for prognostics of asymptotic processes-a case study in composite materials. in Proceedings of the European conference of the prognostics and health management society

Ching J, Au SK, Beck JL (2005a) Reliability estimation for dynamical systems subject to stochastic excitation using subset simulation with splitting. Comput Methods Appl Mech Eng 194(12-16):1557–1579

Ching J, Beck JL, Au SK (2005b) Hybrid Subset Simulation method for reliability estimation of dynamical systems subject to stochastic excitation. Probab Eng Mech 20(3):199–214

Coello Coello CA (2000) Use of a self-adaptive penalty approach for engineering optimization problems. Comput Ind 41(2):113–127

Hastings WK (1970) Monte Carlo sampling methods using Markov chains and their applications. Biometrika 57(1):97–109

Katafygiotis L, Cheung SH (2005) A two-stage subset simulation-based approach for calculating the reliability of inelastic structural systems subjected to gaussian random excitations. Comput Methods Appl Mech Eng 194(12-16):1581–1595

Katafygiotis LS, Cheung SH (2007) Application of spherical subset simulation method and auxiliary domain method on a benchmark reliability study. Struct Saf 29(3):194–207

Li HS (2011) Subset simulation for unconstrained global optimization. Appl Math Model 35(10):5108–5120

Li HS, Au SK (2010) Design optimization using subset simulation algorithm. Struct Saf 32(6):384–392

Li HS, Ma YZ (2015) Discrete optimum design for truss structures by subset simulation algorithm. J Aerosp Eng 28(4):04014091

Li DQ, Xiao T, Cao Z, Zhou C B, and Zhang LM (2015). Enhancement of random finite element method in reliability analysis and risk assessment of soil slopes using Subset Simulation. Landslides, p. 1-11.

Li DQ, Xiao T, Cao Z, Phoon KK and Zhou CB (2016). Efficient and Consistent Reliability Analysis of Soil Slope Stability Using both Limit Equilibrium Analysis and Finite Element Analysis. Applied Mathematical Modelling, in press.

Metropolis N et al (1953) Equation of state calculations by fast computing machines. J Chem Phys 21(6):1087–1092

Papadopoulos V et al (2012) Accelerated subset simulation with neural networks for reliability analysis. Comput Methods Appl Mech Eng 223-224:70–80

Pellissetti MF et al (2006) Reliability analysis of spacecraft structures under static and dynamic loading. Comput Struct 84(21):1313–1325

Robert CP, Casella G (2004) Monte Carlo statistical methods, 2nd edn. Springer, New York

Santoso AM, Phoon KK, Quek ST (2011) Modified metropolis–hastings algorithm with reduced chain correlation for efficient subset simulation. Probab Eng Mech 26(2):331–341

Song S, Lu Z, Qiao H (2009) Subset simulation for structural reliability sensitivity analysis. Reliab Eng Syst Saf 94(2):658–665

Tee KF, Khan LR, Li H (2014) Application of subset simulation in reliability estimation of underground pipelines. Reliab Eng Syst Saf 130:125–131

Wang Y, Cao Z, Au S-K (2010) Efficient Monte Carlo simulation of parameter sensitivity in probabilistic slope stability analysis. Comput Geotech 37(7–8):1015–1022

Wang Y, Cao Z, Au S-K (2011) Practical reliability analysis of slope stability by advanced Monte Carlo simulations in a spreadsheet. Can Geotech J 48(1):162–172

Wang B et al (2015) Efficient functional reliability estimation for a passive residual heat removal system with subset simulation based on importance sampling. Prog Nucl Energy 78:36–46

Zio E, Pedroni N (2012) Monte Carlo simulation-based sensitivity analysis of the model of a thermal–hydraulic passive system. Reliab Eng Syst Saf 107:90–106

Zuev KM, Katafygiotis LS (2011) Modified metropolis–hastings algorithm with delayed rejection. Probab Eng Mech 26(3): 405–412

Zuev KM et al (2012) Bayesian post-processor and other enhancements of subset simulation for estimating failure probabilities in high dimensions. Comput Struct 92–93:283–296