ON THE USE OF SUPPORT VECTOR MACHINES IN STRUCTURAL RELIABILITY

AN UNDERGRADUATE THESIS

BY JUAN CAMILO OSORIO OVIEDO

Supervisor:
Dr. Techn. Diego Andrés Álvarez Marín

Submitted in partial fulfillment of the requirements
for the degree of

Civil Engineer

Department of Civil Engineering
Faculty of Engineering and Arquitecture
Universidad Nacional de Colombia - Sede Manizales

Manizales - Colombia
February - 2022

Who is the happiest of men? He who values the merits of others, and in their pleasure takes joy, even as though it were his own.

— Johann Wolfgang von Goethe

You only live once, but if you do it right,
once is enough.

— Mae West

ACKNOWLEDGMENTS

I want to thank Professor Diego Andrés Álvarez Marín for his continuous advice and teaching. I also want to thank my parents for their love, patience and support.

Juan Camilo Osorio Oviedo
Manizales, Colombia
February - 2022

CONTENTS

ACRONYMS

ASD	Allowable Stress Design
CDF	Cumulative Distribution Function
FORM	First Order Reliability Method
LRFD	Load and Resistance Factor Design
MCM	Monte Carlo Method
MCS	Monte Carlo Simulation
PDF	Probability Density Function
PRNG	Pseudo Random Number Generator
RBF	Radial Basis Function
RSM	Response Surface Method
SORM	Second Order Reliability Method
SLS	Serviceability Limit State
SVM	Support Vector Machines
SVR	Support Vector for Regression
ULS	Ultimate Limit State

1 INTRODUCTION

The fundamental problem of structural reliability is to find the probability that the structure exceeds a critical condition, which is given by a limit state function. Different methods are used to find this probability of failure, since it is often impossible to calculate it analytically and some numerical methods only allow it to be feasible in up to four dimensions, which is rarely the case in real life. Some of these methods are First Order Reliability Method (FORM) and Second Order Reliability Method (SORM).

On the statistical side are Monte Carlo Simulation (MCS) methods which require a large number of samples when calculating small probabilities and is therefore not the most efficient alternative for this case.

Regression methods can also be applied, such as the Response Surface Method (RSM), widely used, but is a rigid method that allows little flexibility. Another alternative are classification methods such as Multi-Layer Perceptrons and Support Vector Machines (SVM), which are used for pattern recognition but are also useful for the structural reliability problem as demonstrated by Hurtado and Alvarez (2003).

It should be clarified that several of the techniques in the field of Statistical Learning can be used for regression or classification. In the case of SVMs, when applied to regression, they are referred to as Support Vector for Regression (SVR). In this regard, Guo and Bai (2009) introduced the Least Square Support Vector for Regression (LSSVR) which has improvements in terms of time and space requirements. Later, Dai et al. (2012) used SVR with adaptive Markov chain simulation and the adaptive Metropolis algorithm and Bourinet (2016) also applied SVR in a way that focuses on the boundary state surface, in addition, to the Metropolis-Hastings algorithm it generates additional training points.

On the other hand, Hurtado (2007) employed SVM with importance sampling and a Markov chain method to determine the point with the highest probability of failure. Later, Hurtado and Alvarez (2010) generated an initial quasi-random population using Sobol sequence for triggering a Particle Swarm Optimization (PSO).

Other significant advances in the area were made by Li, Lu, and Yue (2006) who used two developments: SVM-based FORM and SVM-based MCS. Besides, Song et al. (2013) developed a method called virtual support vector machine (VSVM) that generates virtual samples from real ones and uses a sequential sampling method.

Also, Alibrandi, Alani, and Ricciardi (2015) developed the suitable sampling cone for a response surface based on SVM and Dai and Cao (2017) developed a wavelet SVM based neural network metamodel.

Usually, having the limit state function implicitly generates drawbacks, so several of the techniques mentioned above also deal with this issue.

As has been seen, the Hurtado and Alvarez (2003) article has had a significant influence in the area for almost two decades. It is therefore important to revisit it in order to understand key aspects of the initial proposal and those that have been developed in the meantime. Other crucial aspects to review are related to structural reliability, SVM theory and MCS, being that the objective of this document, along with coding some examples in the python programming language.

IMPORTANT NOTE: The Python codes mentioned in this document as well as the source code of this document in LaTeX can be consulted at the following link:

https://github.com/osvo/SVM_2003

2

THEORETICAL FRAMEWORK

2.1 STRUCTURAL RELIABILITY

Reliability is the probability that a structure will perform adequately during the time and for the purpose for which it was designed. To evaluate structural reliability there are several alternatives. Some are more deterministic based on safety factors while others use probabilities based on *limit states*.

Among the deterministic techniques is the Allowable Stress Design (ASD) which is based on safety factors with which a maximum or yield stress is taken and divided by a safety factor $FS > 1$, which is usually based on engineering judgment, to reduce the nominal resistance R_n and convert it into an allowable stress which is a fraction of its real potential capacity (Chen and Lui, 2005). A general formula for ASD is

$$\frac{R_n}{FS} \geq \sum_{i=1}^{m} Q_{ni} \tag{2.1}$$

where i is the load type, m is the number of load types, Q_{ni} is the acting stress from the acting load of type i.

A disadvantage of safety factors is that they cannot be compared with each other, for example, a beam with $FS = 2.0$ is not necessarily safer than a slope with $FS = 1.5$.

In contrast, there is the Load and Resistance Factor Design (LRFD) which is semi-probabilistic in nature. Here, a reduction factor ϕ multiplies the nominal resistance R_n while an amplification factor γ is applied to each load type i and its value varies according to the uncertainty present in the type of load i. All load combinations should be tested and the worst performing combination should be used for the design (Chen and Lui, 2005). A general formula for LRFD is

$$\phi R_n \geq \sum_{i=1}^{m} \gamma_i Q_{ni}. \tag{2.2}$$

This means that the design resistance already reduced (on the left) must be greater than or equal to the resistance required as an effect of a specific load combination that increases according to the type of load (on the right).

There is also a fully probabilistic method, but to understand it requires first delving into the limit states.

2.1.1 *Limit states*

A general and basic form of a limit state function can be written as follows

$$g(\mathbf{X}) = R(\mathbf{X}) - S(\mathbf{X}) \tag{2.3}$$

where $\mathbf{X} = (x_1, x_2, \cdots, x_n)$ is a vector that contains random variables such as wind speed, surface forces, live load, dead load, material properties, bending moments, axial and shear force, etc. whose randomness is defined by a Probability Density Function (PDF). From the knowledge of these variables, failure criteria such as stresses at certain points of the structure, critical displacements, etc. can be estimated.

On the other hand, $R(\mathbf{X})$ is the resistance of the structure and $S(\mathbf{X})$ is the solicitation on the structure. From there it is understood that when the resistance is greater than the stress $(R(\mathbf{X}) > S(\mathbf{X}))$ the structure is safe and $g(\mathbf{X}) > 0$; when they are equal $(R(\mathbf{X}) = S(\mathbf{X}))$, it is just on the limit state surface in the space of the random variables $\mathbf{X}$ and $g(\mathbf{X}) = 0$. Finally, when the resistance is less than the stress $(R(\mathbf{X}) < S(\mathbf{X}))$, it is possible to speak of failure and $g(\mathbf{X}) < 0$ (Hurtado, 2004).

The limit state surface $g(\mathbf{X}) = 0$ divides the realization space of the vector $\mathbf{X}$ into two: the safe domain $\Omega_s = \{\mathbf{X} : g(\mathbf{X}) > 0\}$ and the failure domain $\Omega_f = \{\mathbf{X} : g(\mathbf{X}) \leq 0\}$.

There are several types of limit states: Ultimate Limit State (ULS) where the structure is close to catastrophic failure and Serviceability Limit State (SLS) which show those limits that must be imposed in order for the structure to be comfortable, habitable and avoid fatigue processes, e.g. vibration, deflection, etc. (Vrijling and Gelder, 2004). There is even the accidental limit state for unforeseen situations such as explosions.

2.1.2 *Probability of failure*

The calculation of the probability of failure is the fundamental objective of structural reliability since there is a complementary relationship between reliability and probability of failure, i.e. Reliability $= 1 - P_f$. Therefore, in some cases it may be more convenient to P_f directly.

It is possible to estimate such probability according to whether the response of the structure exceeds a certain threshold or not under the presence of random variables. The following expression is used

$$P_f = P\left[g(\mathbf{X}) \leq 0\right] = P\left[\mathbf{X} \in \Omega_f\right] = \int_{\Omega_f} f_{\mathbf{X}}(\mathbf{x}) d\mathbf{X} \tag{2.4}$$

where $f_{\mathbf{X}}(\mathbf{x})$ is a joint probability density function of $\mathbf{X}$.

It is often impossible to solve the integral of equation 2.4 analytically and numerical integration only is efficient up to four dimensions

(Stocki et al., 2017), which is too few in a realistic scenario. In addition, the evaluation of the limit state function is computationally expensive, then it is an objective to minimize the number of times it is evaluated.

There are different techniques to deal with this problem, one of them is the MCS, which has the disadvantage that when calculating small probabilities ($P_f \leq 10^{-3}$) it requires many samples, so it is not the most efficient method in these cases (Uribe, 2011).

2.2 MONTE CARLO

2.2.1 *Introduction*

The Monte Carlo method is used to estimate mathematical expressions that due to their complexity are computationally expensive to evaluate because random variables are involved (Kenton, 2021). Approximate output ranges can be predicted from knowledge of the PDFs that model the input values.

It is necessary to repeatedly use random sampling (usually *pseudo-random number sampling*) to obtain the output values and repeat as many times as necessary to clearly define the output range. This repetition is made necessary by the law of large numbers which shows how when performing the same experiment, the average converges to the expected value as the number of times the experiment is performed is increased, as long as it is performed a sufficiently large number of times. In this way it guarantees stable long-term results in random events.

2.2.2 *Brief history*

It was developed primarily by Stanislaw Ulam and John Von Neumann while they were working on the Manhattan Project in Los Alamos. The name of the method was suggested by Nicholas Metropolis and comes from the Monte-Carlo casino located in the principality of Monaco, this due to the fact that Ulam came up with the idea while playing a game of Canfield solitaire and wanted to calculate the probability of winning a game (IBM, 2020). He tried using combinatorics but it was getting overly complex, so he thought that numerous games could be played (or simulated) and the proportion of those that were successful could be calculated.

The use of this method was crucial to the success of the Manhattan Project and Pseudo Random Number Generator (PRNG) began to be used for it.

2.2.3 *Pseudo-random number generators (PRNG)*

Generating true random numbers is not possible using an algorithm that follows deterministic commands; to do so would require hardware that measures physical values that are unpredictable such as atmospheric or thermal noise. This process is neither economical nor efficient, especially when everyday applications of these generators require huge amounts of numbers (Veaux, 2020).

Another drawback of true random numbers is their lack of reproducibility, which is useful, for example, when trying to find an error in a computer code. Moreover, in many cases it is not absolutely necessary to generate such sequences of true random numbers but rather pseudo-random numbers are sufficient (Wikipedia contributors, 2022c).

Pseudo-random number generators are algorithms that generate numerical sequences that are close to those of a truly random sequence. Although they appear random, they are in fact completely deterministic, since each value depends on the previous one and the first one is generated from a seed that determines the whole sequence. They can have weaknesses such as short periods (sequences are repeated over and over again) and poor distributions (Healy, 2015).

Frequently, PRNGs deliver a number of the form $u = m/M$, where m is an integer that is uniformly distributed over the interval $[0, M-1]$ to generate uniform samples $U(0,1)$. The quality of the approximation is directly proportional to the value of M, which gives better results if it is of the form $M = 2^b$, where b is an integer, since computers find it easier to work in powers of the number 2 Yates and Goodman (2005).

According to the above, even to generate numbers in continuous domains it is necessary to initially generate uniformly distributed integers. With $U(0,1)$ it is possible to extend the domain between two integer numbers a and b in this way

$$U(a,b) = a + (b-a) \cdot U(0,1). \tag{2.5}$$

Some methods to generate such integers are the Linear Congruential Generator which is one of the pioneers and the *Mersenne Twister* which comes by default in python and therefore will be the one used in this document.

2.2.3.1 *Mersenne Twister*

Its name is due to the fact that its period is a Mersenne prime number, which has the form $2^n - 1$, where n is an integer. It caused a great excitement after its appearance because of its long period, which in its most extended version (MT19937) is $2^{19937} - 1$ (Matsumoto and Nishimura, 1998), a number that has about six thousand decimal digits.

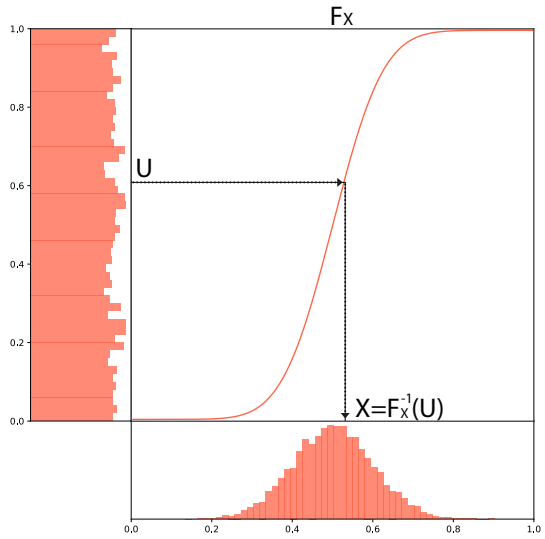

Figure 2.1: This is an example of inverse transform sampling. In the upper left part is a uniform distribution U in $[0,1)$. In the main frame is the Cumulative Distribution Function (CDF) F_X, *associated* in this example to the normal distribution (The CDF of the normal distribution is not well-defined but there are several approximations), so consequently, in the lower part are the values of X that follow the normal distribution as a result of using U in the inverse function of the CDF.

This algorithm passes many statistical randomness tests, however, there are others that it does not pass with important consequences that make it not recommended today for general use (Vigna, 2019), since it is neither cryptographically secure nor very fast. Despite the latter, it is sufficient for the purposes of this paper.

2.2.4 *Pseudo-random number sampling*

It is a strategy for obtaining pseudo-random numbers that are distributed according to a given PDF. Usually a PRNG is needed to produce uniformly distributed numbers and from them the sampling of the objective function is done, as in the case of Inverse Transform Sampling.

2.2.5 *Inverse Transform Sampling*

It is a pseudo-random sampling method that allows samples to be drawn from any PDF whose associated CDF and its inverse are known (Moufad, 2021).

Given a random variable U that follows a uniform distribution in $[0,1)$ and has an invertible CDF F_x it is possible to make the values of X to be distributed according to the distribution F_x if $X = F_X^{-1}(U)$

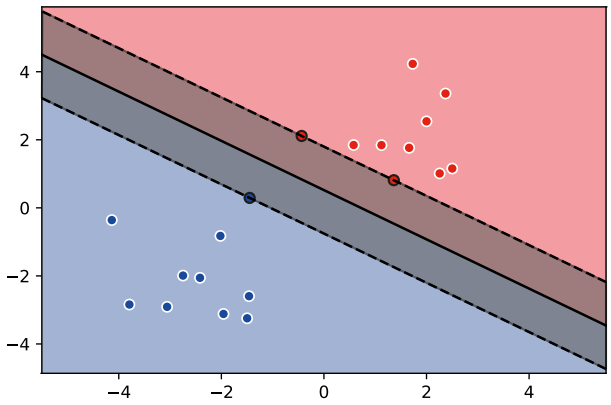

Figure 2.2: The solid line is the optimal hyperplane $\langle \mathbf{w}, \mathbf{x} \rangle + b = 0$ that divides the space into two categories: blue (negative) and red (positive). The dashed lines $\langle \mathbf{w}, \mathbf{x} \rangle + b = \pm 1$ limit the margin (which is shaded) and pass through the support vectors, which have a black outline unlike the other vectors which have a white outline.

(Nguyen, 2020). It is necessary to remember that the relationship between the PDF f_x and the CDF F_x is

$$F_X(x) = \int_{-\infty}^{x} f_X(t)dt. \tag{2.6}$$

2.3 SUPPORT VECTOR MACHINES

2.3.1 *Brief history*

Between 1962 and 1964, Vladimir Vapnik and Alexey Chervonenkis developed the Generalized Portrait Method (Chervonenkis, 2013), originally written in Russian and of which nothing was known in the West until much later. In the decade of the 1990s Vapnik left the Soviet Union and moved to the USA to work at AT&T Bell Labs. The above created the conditions for the paper *A training algorithm for optimal margin classifiers* by Boser, Guyon, and Vapnik (1992).

2.3.2 *Idea*

If it is desired to obtain a simple and efficient classifier from two linearly separable categories it is reasonable to think that the best separating hyperplane is the one that is halfway, simultaneously moving away from both categories so as not to be biased. This is called maximizing the *margin*, defining the margin as the distance between the decision boundary and some of the nearest points of each category measured along the smallest perpendicular line. These latter points are called *support vectors* (Nefedov, 2016).

It is important to note that once a support vector classifier has been defined, it will only change if samples that are within the margin are added, i.e. closer to the decision boundary than the previous support vector. The hyperplane depends on the support vectors and the outlying vectors have no influence.

2.3.3 *Development*

If there is a linearly separable set of ℓ samples in dimension n each linked to a class $y_i \in \{-1, 1\}$, it is possible to separate them by at least one hyperplane given by

$$f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b = 0 \tag{2.7}$$

of dimension $n - 1$, where $\mathbf{w}$ is the normal vector of the hyperplane, b is the intercept of the hyperplane and $\langle \cdot, \cdot \rangle$ is the inner product. There are many combinations of $\mathbf{w}$ and b that do the job considering that any length of the vector $\mathbf{w}$ works. Therefore it is necessary to place additional constraints that allow to choose unique values, then it is assumed that any positive sample $\mathbf{x}_+$ satisfies that

$$\langle \mathbf{w}, \mathbf{x}_+ \rangle + b \geq +1 \tag{2.8}$$

and any negative sample $\mathbf{x}_-$ satisfies that

$$\langle \mathbf{w}, \mathbf{x}_- \rangle + b < -1. \tag{2.9}$$

These last two inequalities can be converted into a single inequality by using the variable y_i which is 1 when the sample $\mathbf{x}_i$ is positive and it is -1 when $\mathbf{x}_i$ is negative (Berwick, 2010). Therefore, it remains that

$$y_i \langle \mathbf{w}, \mathbf{x} \rangle + b \geq 1. \tag{2.10}$$

In particular, the support vectors fulfill

$$y_i \langle \mathbf{w}, \mathbf{x} \rangle + b = 1. \tag{2.11}$$

This ensures that there is no vector inside the margin and that the support vectors are just at the margin boundary. The margin is defined as the distance between the two classes measured along the $\mathbf{w}$ vector (Nefedov, 2016). Ideally the margin should be as wide as possible because points close to the hyperplane are problematic and difficult to classify. Also, to avoid overfitting.

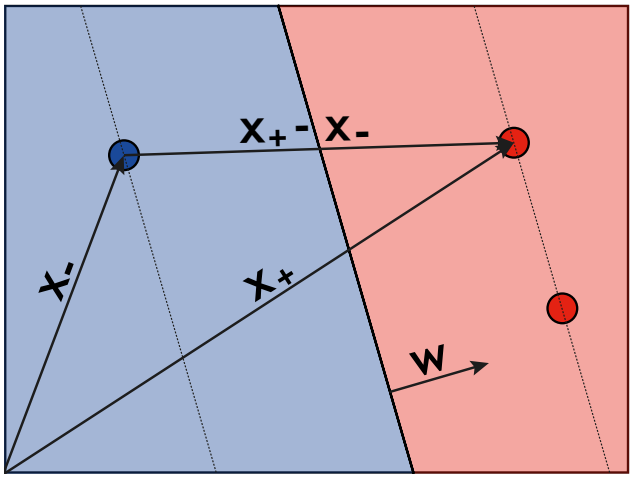

Figure 2.3: Since the margin is the distance between classes measured along the vector $\mathbf{w}$ that is normal to the decision boundary, it is possible to find the length of the margin Δ with the inner product of normalized $\mathbf{w}$ and the difference of the support vectors $\mathbf{x}_+$ and $\mathbf{x}_-$.

To know the length of the margin, the inner product can be made between the difference of support vectors $\mathbf{x}_+$ and $\mathbf{x}_-$ with the normalized vector $\mathbf{w}$, this means

$$\Delta = \left\langle (\mathbf{x}_+ - \mathbf{x}_-), \frac{\mathbf{w}}{\|\mathbf{w}\|} \right\rangle. \tag{2.12}$$

Now a positive sample $\mathbf{x}_+$ is inserted in equation 2.11, from which it is obtained that

$$\langle \mathbf{w}, \mathbf{x}_+ \rangle = 1 - b. \tag{2.13}$$

Similarly, with a negative sample $\mathbf{x}_-$ is obtained that

$$\langle -\mathbf{w}, \mathbf{x}_- \rangle = 1 + b. \tag{2.14}$$

Substituting the two previous results in equation 2.12, it follows that (Winston, 2010)

$$\Delta = \frac{1}{\|\mathbf{w}\|} \left(\langle \mathbf{w}, \mathbf{x}_+ \rangle - \langle \mathbf{w}, \mathbf{x}_- \rangle \right) \tag{2.15}$$

$$= \frac{1}{\|\mathbf{w}\|} \left((1 - b) + (1 + b) \right) \tag{2.16}$$

$$= \frac{2}{\|\mathbf{w}\|}. \tag{2.17}$$

As mentioned above, the aim is to maximize the margin; however, for mathematical convenience, it is possible to work with an equivalent

problem whose solution also maximizes the margin, related to each other in this way:

$$\max_{\mathbf{w},b} \frac{2}{\|\mathbf{w}\|} \rightarrow \max_{\mathbf{w},b} \frac{1}{\|\mathbf{w}\|} \rightarrow \min_{\mathbf{w},b} \|\mathbf{w}\| \rightarrow \min_{\mathbf{w},b} \frac{1}{2} \|\mathbf{w}\|^2 \tag{2.18}$$

Therefore, the optimization problem is

$$\min_{\mathbf{w},b} \frac{1}{2} \|\mathbf{w}\|^2 \tag{2.19}$$

subject to

$$y_i \langle \mathbf{w}, \mathbf{x} \rangle + b - 1 = 0. \tag{2.20}$$

The problem written in this way is known as a primal optimization problem, where $\mathbf{w}$ and b are known as primal variables.

Lagrange multipliers $\alpha_i \geq 0$ will be used to apply the method of the same name, which converts a constrained problem of p variables and q restrictions to an unconstrained problem with $p + q$ variables whose equations are easier to solve.

Now it is necessary to minimize the Lagrangian, which is

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^{\ell} \alpha_i \left[y_i \left(\langle \mathbf{w}, \mathbf{x}_i \rangle + b \right) - 1 \right], \tag{2.21}$$

for this purpose it is necessary to minimize with respect to the primal variables while maximizing with respect to the $\alpha_i \geq 0$ Lagrange multipliers that contribute to the minimization of the lagrangian (Veganzones, 2009).

As in any other optimization problem, the derivatives are equated to zero. From the derivative

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{\ell} \alpha_i y_i \mathbf{x}_i = 0 \tag{2.22}$$

it is possible to obtain an expression for $\mathbf{w}$

$$\mathbf{w} = \sum_{i=1}^{\ell} \alpha_i y_i \mathbf{x}_i \tag{2.23}$$

which indicates that the vector $\mathbf{w}$ is a linear sum of the samples.

The other derivative results in

$$\frac{\partial \mathcal{L}}{\partial b} = - \sum_{i=1}^{\ell} \alpha_i y_i = 0. \tag{2.24}$$

To eliminate the dependence of $\mathbf{w}$ on the Lagrangian, equation 2.23 can be replaced in equation 2.21. This results in

$$\mathcal{L} = \frac{1}{2}\left\langle \sum_{i=1}^{\ell} \alpha_i y_i \mathbf{x}_i, \sum_{j=1}^{\ell} \alpha_j y_j \mathbf{x}_j \right\rangle - \sum_{i=1}^{\ell} \left[y_i \left(\sum_{j=1}^{\ell} \alpha_j y_j \left\langle \mathbf{x}_j, \mathbf{x}_i \right\rangle + b \right) - 1 \right]. \tag{2.25}$$

Expanding the terms and using equation 2.24 to simplify, results in

$$\mathcal{L}(\boldsymbol{\alpha}) = \sum_{j=1}^{\ell} \alpha_j - \frac{1}{2} \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \alpha_i \alpha_j y_i y_j \left\langle \mathbf{x}_i, \mathbf{x}_j \right\rangle. \tag{2.26}$$

This representation is known as the dual lagrangian problem. The primal variables do not appear in it, so the expression must be maximized since it depends only on the Lagrange multipliers. Precisely $\boldsymbol{\alpha}$ represents the vector of such multipliers.

The maximization of this lagrangian is a quadratic problem with linear constraints, in which case the fastest convergence is obtained by quadratic optimization (or quadratic programming). Also, It can be proved that this is working in a convex space, so the algorithm will not stall at local maxima (Winston, 2010). The respective scikit-learn implementation is based on libsvm.

So far, a powerful algorithm is already available, however, this SVM is not suitable for nonlinear problems. Therefore, an ingenious strategy called the *kernel trick* must be introduced.

2.3.4 *Kernel trick*

Initially, a *kernel function* as a function $K(\mathbf{u}, \mathbf{v}) : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}$ that satisfies

$$K(\mathbf{u}, \mathbf{v}) = \langle \phi(\mathbf{u}), \phi(\mathbf{v}) \rangle \tag{2.27}$$

through a mapping function $\phi : \mathbb{R}^n \mapsto \mathbb{R}^m$.

A kernel function allows to measure the similarity between two vectors and represents an inner product in an output space that is usually of a higher dimension than the input space. This is done because although in a space the data may not be linearly separable, a mapping can be made to a space where it is and there the optimal hyperplane can be found (Dobilas, 2021). This process is called the kernel trick.

It might be thought that it is necessary to evaluate the vectors in ϕ and then operate the inner product, which can be time consuming, especially if it is in a high dimension, and in the end obtain a simple scalar. Actually if a suitable kernel is chosen this is not necessary

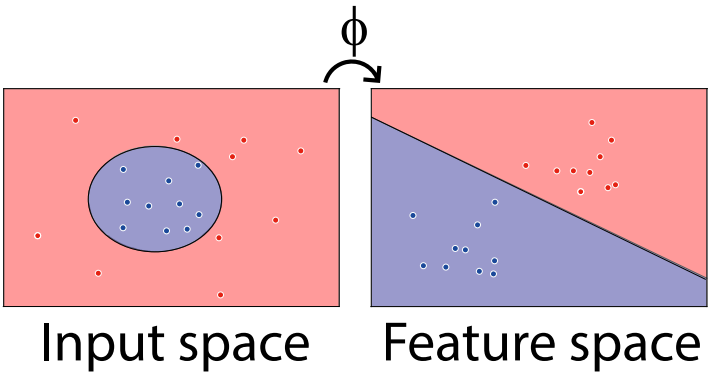

Figure 2.4: The kernel trick takes advantage of the inner product dependence of the Lagrangian in order to work with nonlinear problems. To do so, a ϕ transformation is applied from an input space where the vectors are not linearly separable to one where they are, without the need to know ϕ but the function K that makes the inner product of the transformed vectors.

(Afonja, 2017). The beauty of the kernel trick is that neither ϕ nor the output space m need to be known.

On the other hand, when different kernels are tested, a large number of times common support vectors are found, so their selection is not crucial for the classification, although it is still important.

A disadvantage of kernels is that they may not be intuitive due to the fact that it is generally difficult to properly interpret what they do. Precisely because it is not necessary to know ϕ, it may appear to work as a black box.

Finally, using the kernel trick it is possible to write the dual problem loss function 2.26 as (Winston, 2010)

$$\mathcal{L}(\boldsymbol{\alpha}) = \sum_{j=1}^{\ell} \alpha_j - \frac{1}{2} \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \alpha_i \alpha_j y_i y_j K\left(\mathbf{x}_i, \mathbf{x}_j\right). \tag{2.28}$$

2.3.4.1 *Radial Basis Function (RBF)*

Denotes the proximity between two vectors due to its use of the Euclidean norm so that nearby vectors will have a higher value of the RBF kernel than distant vectors (Bernstein, 2017). It is a general-purpose kernel and it is defined as

$$K(\mathbf{u}, \mathbf{v}) = \exp\left(-\gamma \left\|\mathbf{u} - \mathbf{v}\right\|^2\right) \tag{2.29}$$

where $\gamma > 0$ is a free parameter indicating the spread of the kernel. It is common to use $\gamma = 0.5\sigma^{-2}$, where σ is the variance of the data. In such case it is known as Gaussian RBF.

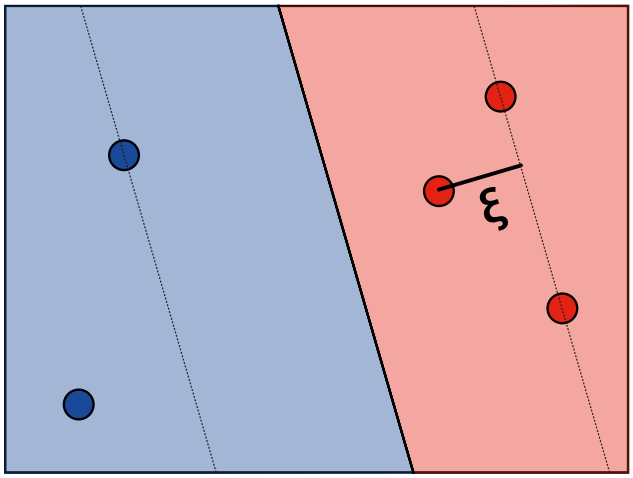

Figure 2.5: When there is a soft margin it is possible that there are vectors within the margin. In this case there are two vectors that still act as support even though they are not the closest to the other class. On the other hand, it is necessary to introduce slack variables ξ that show the magnitude of the error. A decision must be made about the relevance to be given to these errors by means of the C variable.

On the other hand, the RBF kernel is used by default in scikit-learn and is appropriate for the development of this document.

2.3.5 *Soft margin SVM*

One might well think that using the kernel trick is sufficient, however, in everyday applications it is common that error-free separating hyperplanes cannot be found because if there are no linearly separable classes the problem may be infeasible since there is no $\mathbf{w}$ or b that satisfy all the constraints simultaneously.

However, a fitting parameter can be introduced to control how much error is admissible. In this case there may be training vectors on the margin and vectors on the wrong side of the hyperplane. This is known as soft margin SVM (Ahlawat, 2020), as opposed to the one discussed above which is consequently known as hard margin SVM.

The recently mentioned parameter is called C. When C is very small only a few classification errors are allowed making the margin wide and spanning many training vectors, becoming time consuming. In contrast, when C is very large many classification errors are allowed and therefore the margin is smaller (*Scikit-learn 1.0.2 documentation* 2021).

Errors will be denoted as ξ_i. So problem 2.19-2.20 can be rewritten in its soft-margin version as

$$\min_{\mathbf{w},b} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^{\ell} \xi_i \tag{2.30}$$

subject to

$$y_i \langle \mathbf{w}, \mathbf{x} \rangle + b \geq 1 - \xi_i, \qquad \xi_i \geq 0 \tag{2.31}$$

The dual problem 2.26 remains the same, but the following condition is added (Berwick, 2010)

$$C \geq \alpha_i \geq 0. \tag{2.32}$$

3

ALGORITHM

The following algorithm was proposed by Hurtado and Alvarez (2003):

A databank D is generated as a set of samples of uniformly distributed basic variables (note that it is neither necessary nor appropriate to use the associated PDFs due to the fact that the uniform distribution scans the space better).

Two samples should be taken, either from D or extras. Of those two points there should be one from each category. The choice of these points can be made under different criteria such as good engineering judgment, a useful one is to place extremely unfavorable conditions that will surely lead to failure for $\mathbf{y_f}$ while for the other point $\mathbf{y_s}$ too favorable conditions are placed.

With the two existing samples a first decision boundary is calculated.

The set T is sequentially increased by testing the points that are closer to the current margin. These points are removed from D. The structural solver is only called in case the sample is inside the margin. Otherwise, it is not taken into account and its output is not calculated, although it could eventually be if it is in a new margin, which is unlikely because it is becoming progressively smaller. It is worth remembering that only the new points that are within the margin have any influence on the classifier, for the rest it is not worth incorporating them into the model.

Eventually all samples in the database have been checked and D is empty. At that time, the training is finished.

If the size of D is sufficiently large, the margin should be very small and the support vectors should be very close to the decision function on both sides.

It is necessary to mention that scikit-learn assumes that the data is scaled and centered, so in case it is not, the corresponding preprocessing must be done.

4 IMPLEMENTATION

4.1 TWO-DIMENSIONAL DECISION FUNCTION

The limit state function

$$g(x_1, x_2) = 2 - x_2 + \exp\left(-\frac{x_1^2}{10}\right) + \left(\frac{x_1}{5}\right)^4 \tag{4.1}$$

has two random variables distributed as shown in Table 4.1.

VARIABLE	PDF	MEAN	STD
x_1	Normal	0	1
x_2	Normal	0	1

Table 4.1: Random variables considered in section 4.1

Figure 4.1 shows the 10^6 samples used for the MCS. Each ordered pair was evaluated in the limit state function 4.1. If less than or equal to zero it was classified as unsafe, otherwise it was classified as safe. This MCS with these samples resulted in a probability of failure $P_f = 1.805 \times 10^{-3}$.

In the same figure 4.1 it appears at first glance that the proportion of failed samples is much higher. However, it should be noted that a large number of points are overlapping in a very dense area around $(0, 0)$. Therefore, conclusions should not be hastily drawn just by looking at the graph.

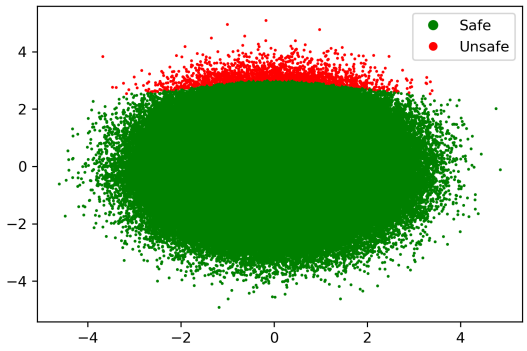

Figure 4.1: Standard normal random samples and their classification after a MCS. The red dots fulfill $g(\mathbf{X}) \leq 0$ and the green dots $g(\mathbf{X}) > 0$.

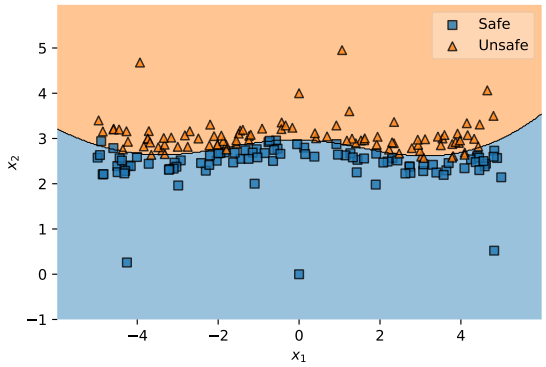

Figure 4.2: SVM implementation with actual categories and decision function. It is observed that the margin is small and that the support vectors tend to be very close to the decision boundary.

On the other hand, 10^3 uniformly distributed samples were used to train the SVM algorithm although the majority of them were not taken into account as they were not close enough to the margin to be useful.

To initialize the SVM algorithm, one point from each of the two categories was required. The points $(0,0)$ and $(0,4)$ which are safe and unsafe, respectively, were chosen. In addition, a RBF kernel was used (see equation 2.29) with $\sigma = 0.07$. This value was chosen after testing initially with logarithmic scale values until a value was found that gives good accuracy without the need to use too many support vectors, which would make it too slow.

Figure 4.2 shows the points that trained the model. They are identified with their actual category and additionally the decision boundary can be seen. The model ended up with 114 support vectors: 58 safe and 56 unsafe.

Finally, to calculate the probability of failure, the same 10^6 points used for the MCS (i.e. the same seed was employed) were used and evaluated in the resulting classifier. The result was $P_f = 1.831 \times 10^{-3}$ which means a relative error of 1.44%.

METHOD	NO. OF CALLS	P_f
MCS	10^6	1.805×10^{-3}
SVM	129	1.831×10^{-3}

Table 4.2: Comparison of the number of calls to the limit state function and the probability of failure for MCS and SVM methods.

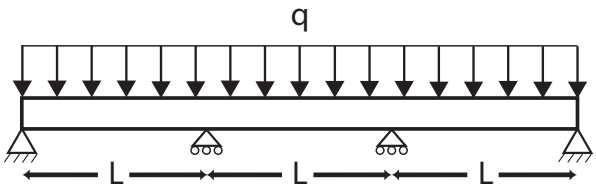

Figure 4.3: Schematic of three-span continuous beam.

4.2 RELIABILITY ANALYSIS OF A THREE-SPAN CONTINUOUS BEAM

Consider the three-span beam in figure 4.3. The associated limit state function is

$$g(q, E, I) = \frac{L}{360} - 0.0069\frac{qL^4}{EI} \tag{4.2}$$

where the minuend represents the allowable deflection and the subtrahend the maximum deflection. In table 4.3 are the random variables used, which are normally distributed and uncorrelated. It is also necessary to consider that the length L is equal to 5 meters.

VARIABLE	PDF	MEAN	STD	UNITS
q	Normal	10	0.4	$\frac{kN}{m}$
E	Normal	2×10^7	0.5×10^7	$\frac{kN}{m^2}$
I	Normal	8×10^{-4}	1.5×10^{-4}	m^4

Table 4.3: Random variables considered in section 4.2

The MCS delivers a probability of failure $P_f = 8.82 \times 10^{-4}$ using 10^6 points. On the other hand, 10^3 points were used to train the SVM model with $\gamma = 8$ (chosen after testing with logarithmic scale values first), leaving 165 support vectors: 82 safe and 83 unsafe. Resulting in a probability of failure $P_f = 8.77 \times 10^{-4}$, which means a relative error of 0.57%.

METHOD	NO. OF CALLS	P_f
MCS	10^6	8.82×10^{-4}
SVM	182	8.77×10^{-4}

Table 4.4: Comparison of the number of calls to the limit state function and the probability of failure for MCS and SVM methods.

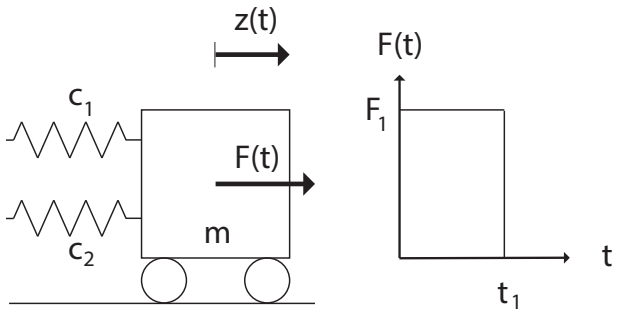

Figure 4.4: Non-linear oscillator – system definition and applied load.

4.3 DYNAMIC RESPONSE OF A NON-LINEAR OSCILLATOR

Consider the nonlinear oscillator shown in figure 4.4, which has as limit state function

$$g(c_1, c_2, m, r, t_1, F_1) = 3r - \left| \frac{2F_1}{m\omega_0^2} \sin\left(\frac{\omega_0 t_1}{2}\right) \right| \tag{4.3}$$

where

$$\omega_0 = \sqrt{\frac{c_1 + c_2}{m}}, \tag{4.4}$$

r is the displacement at which one of the springs yields, the subtrahend z is the maximum displacement response of the system.

The random variables are generated according to table 4.5.

VARIABLE	PDF	MEAN	STD
m	Lognormal	1.0	0.05
c_1	Lognormal	1.0	0.10
c_2	Lognormal	0.1	0.01
r	Lognormal	0.5	0.05
F_1	Lognormal	1.0	0.20
t_1	Lognormal	1.0	0.20

Table 4.5: Random variables considered in section 4.3 (Bucher and Bourgund, 1990)

The MCS was performed with 10^6 points and gave a probability of failure $P_f = 3.24 \times 10^{-2}$. The SVM model was trained with 10^3 uniformly distributed points and the points (x_1, x_2) chosen to start the algorithm were taken from the extremes of the MCS. A total of 153 support vectors were used: 81 safe and 72 unsafe.

In this case a $\gamma = 4.79$ was used (chosen after testing with logarithmic scale values first). Finally, 10^6 points were evaluated in the

decision function and $P_f = 3.23 \times 10^{-2}$ was obtained, which means a relative error of 0.49%.

METHOD	NO. OF CALLS	P_f
MCS	10^6	3.2447×10^{-2}
SVM	177	3.2289×10^{-2}

Table 4.6: Comparison of the number of calls to the limit state function and the probability of failure for MCS and SVM methods.

4.4 FINAL THOUGHTS AND FUTURE INVESTIGATIONS

SVM are simple, elegant and deliver remarkable results using much less computational load than the Monte Carlo Method (MCM). The algorithm tested here makes efficient use of samples and avoids unnecessary calls to the limit state function which requires intensive computation.

It is worthwhile to delve into the developments that have been made since the publication of the paper of Hurtado and Alvarez (2003), such as the combination of different methods at distinct stages. From there, it is possible to think of new alternatives or the improvement of existing ones during a postgraduate research.

BIBLIOGRAPHY

Afonja, Tejumade (2017). *Kernel Functions. Lately, I have been doing some reading... | Towards Data Science*. https://towardsdatascience.com/kernel-function-6f1d2be6091. Accessed: 2022-02-11.

Ahlawat, Randeep (2020). *SVM from scratch using Quadratic Programming | Medium*. https://medium.com/@ahlawat.randeep/svm-from-scratch-using-quadratic-programming-90b4dbc5e1d2. Accessed: 2022-02-11.

Alibrandi, Umberto, Amir M. Alani, and Giuseppe Ricciardi (July 2015). "A new sampling strategy for SVM-based response surface for structural reliability analysis." In: *Probabilistic Engineering Mechanics* 41, pp. 1–12. ISSN: 02668920. DOI: 10.1016/j.probengmech.2015.04.001.

Bernstein, Matthew (2017). *The Radial Basis Function Kernel*. URL: https://pages.cs.wisc.edu/~matthewb/pages/notes/pdf/svms/RBFKernel.pdf.

Berwick, R. (2010). *An Idiot's guide to Support vector machines (SVMs) R. Berwick: A New Generation of Learning Algorithms*. URL: http://web.mit.edu/6.034/wwwbob/svm-notes-long-08.pdf.

Boser, Bernhard E, Isabelle M Guyon, and Vladimir N Vapnik (1992). *A Training Algorithm for Optimal Margin Classifiers*.

Bourinet, J. M. (June 2016). "Rare-event probability estimation with adaptive support vector regression surrogates." In: *Reliability Engineering and System Safety* 150, pp. 210–221. ISSN: 09518320. DOI: 10.1016/j.ress.2016.01.023.

Bucher, C G and U Bourgund (1990). *A fast and efficient response surface approach for structural reliability problems*, pp. 57–66.

Chen, Wai-Fah and E. M. Lui (2005). *Handbook of structural engineering*. CRC Press. ISBN: 0849315697.

Chervonenkis, Alexey (2013). "Early History of Support Vector Machines." In: DOI: 10.1007/978-3-642-41136-6__3.

Dai, Hongzhe and Zhenggang Cao (Apr. 2017). "A Wavelet Support Vector Machine-Based Neural Network Metamodel for Structural Reliability Assessment." In: *Computer-Aided Civil and Infrastructure Engineering* 32 (4), pp. 344–357. ISSN: 14678667. DOI: 10.1111/mice.12257.

Dai, Hongzhe et al. (Oct. 2012). "Structural Reliability Assessment by Local Approximation of Limit State Functions Using Adaptive Markov Chain Simulation and Support Vector Regression." In: *Computer-Aided Civil and Infrastructure Engineering* 27 (9), pp. 676–686. ISSN: 10939687. DOI: 10.1111/j.1467-8667.2012.00767.x.

Das, P.K and Wei Zhang (2003). *Guidance on Structural Reliability Analysis of Marine Structures*.

Dobilas, Saul (2021). *SVM Classifier and RBF Kernel — How to Make Better Models in Python | Towards Data Science*. https://towardsdatascience.com/svm-classifier-and-rbf-kernel-how-to-make-better-models-in-python-73bb4914af5b. Accessed: 2022-02-11.

Guo, Zhiwei and Guangchen Bai (Apr. 2009). "Application of Least Squares Support Vector Machine for Regression to Reliability Analysis." In: *Chinese Journal of Aeronautics* 22 (2), pp. 160–166. ISSN: 10009361. DOI: 10.1016/S1000-9361(08)60082-5.

Healy, Jherek (2015). *Why is the Mersenne Twister regarded as good? - Computer Science Stack Exchange*. https://cs.stackexchange.com/questions/50059/why-is-the-mersenne-twister-regarded-as-good. Accessed: 2022-02-11.

HolyPython (2019). *Support Vector Machine - HolyPython.com*. https://holypython.com/svm/. Accessed: 2022-02-11.

Hurtado, Jorge E. (July 2004). "An examination of methods for approximating implicit limit state functions from the viewpoint of statistical learning theory." In: *Structural Safety* 26 (3), pp. 271–293. ISSN: 01674730. DOI: 10.1016/j.strusafe.2003.05.002.

Hurtado, Jorge E. (Jan. 2007). "Filtered importance sampling with support vector margin: A powerful method for structural reliability analysis." In: *Structural Safety* 29 (1), pp. 2–15. ISSN: 01674730. DOI: 10.1016/j.strusafe.2005.12.002.

Hurtado, Jorge E. and Diego A. Alvarez (2003). "Classification Approach for Reliability Analysis with Stochastic Finite-Element Modeling." In: *Journal of Structural Engineering* 129 (8), pp. 1141–1149. ISSN: 0733-9445. DOI: 10.1061/(asce)0733-9445(2003)129:8(1141).

Hurtado, Jorge E. and Diego A. Alvarez (Jan. 2010). "An optimization method for learning statistical classifiers in structural reliability." In: *Probabilistic Engineering Mechanics* 25 (1), pp. 26–34. ISSN: 02668920. DOI: 10.1016/j.probengmech.2009.05.006.

Hurtado, Jorge E. and Diego A. Alvarez (2013). "A method for enhancing computational efficiency in Monte Carlo calculation of failure probabilities by exploiting FORM results." In: *Computers and Structures* 117, pp. 95–104. ISSN: 00457949. DOI: 10.1016/j.compstruc.2012.11.022.

IBM (2020). *¿Qué es la simulación de Monte Carlo? - Colombia | IBM*. https://www.ibm.com/co-es/cloud/learn/monte-carlo-simulation. Accessed: 2022-02-11. In Spanish.

Jelic, Andrés Wellmann et al. (2003). *Distributed computing of failure probabilities for structures in civil engineering*, pp. 52–79.

Kenton, Will (2021). *Monte Carlo Simulation Definition*. https://www.investopedia.com/terms/m/montecarlosimulation.asp. Accessed: 2022-02-11.

Li, Hong Shuang, Zhen Zhou Lu, and Zhu Feng Yue (2006). "Support vector machine for structural reliability analysis." In: *Applied Mathematics and Mechanics (English Edition)* 27 (10), pp. 1295–1303. ISSN: 02534827. DOI: 10.1007/s10483-006-1001-z.

Matsumoto, Makoto and Takuji Nishimura (1998). "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator." In: *ACM Transactions on Modeling and Computer Simulation* 8 (1), pp. 3–30.

Melchers, Robert and André Beck (2018). *Structural reliability analysis and prediction*. Third. Wiley. ISBN: 9781119266075.

Moufad, Badr (2021). *Deriving the inverse transform sampling method from first principles | Towards Data Science*. https://towardsdatascience.com/an-insight-on-generating-samples-from-a-custom-probability-density-function-d0a06c290c54. Accessed: 2022-02-11.

Nefedov, Alexey (2016). *Support Vector Machines : A Simple Tutorial*.

Nguyen, Khanh (2020). *How to generate Gaussian samples. Part 1: Inverse transform sampling | MTI Technology | Medium*. https://medium.com/mti-technology/how-to-generate-gaussian-samples-347c391b7959. (Visited on 02/11/2022).

Pedregosa, F et al. (2011). "Scikit-learn: Machine Learning in Python." In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Schueremans, Luc and Dionys Van Gemert (2005). "Benefit of splines and neural networks in simulation based structural reliability analysis." In: *Structural Safety* 27 (3), pp. 246–261. ISSN: 01674730. DOI: 10.1016/j.strusafe.2004.11.001.

Scikit-learn 1.0.2 documentation (2021). https://scikit-learn.org/stable/modules/svm.html. Accessed: 2022-02-11.

Song, Hyeongjin et al. (Apr. 2013). "Adaptive virtual support vector machine for reliability analysis of high-dimensional problems." In: *Structural and Multidisciplinary Optimization* 47 (4), pp. 479–491. ISSN: 1615147X. DOI: 10.1007/s00158-012-0857-6.

Stocki, R et al. (2017). "FE based structural reliability analysis using STAND environment." In: *Computer Assisted Mechanics and Engineering Sciences*, 16, pp. 35–58.

Uribe, Felipe (2011). "Implementation of simulation methods in structural reliability." Universidad Nacional de Colombia.

Veaux, Franklin (2020). *Why are pseudo-random number generators used instead of real ones? - Quora*. https://www.quora.com/Why-are-pseudo-random-number-generators-used-instead-of-real-ones. Accessed: 2022-02-11.

Veganzones, Miguel A (2009). *Introduction Support Vector Machines (SVM) Appendix Support Vector Machines*.

Vigna, Sebastiano (2019). *It Is High Time We Let Go Of The Mersenne Twister* *. Università degli Studi di Milano.

Vrijling, J.K. and van Gelder (2004). *Life time management of structures*. European Safety, Reliability and Data Association, pp. 148–172.

Wikipedia contributors (2022a). *Mersenne Twister — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Mersenne_Twister&oldid=1071140937. [Online; accessed 12-February-2022].

Wikipedia contributors (2022b). *Monte Carlo method — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Monte_Carlo_method&oldid=1066704511. [Online; accessed 12-February-2022].

Wikipedia contributors (2022c). *Pseudorandom number generator — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Pseudorandom_number_generator&oldid=1070634528. [Online; accessed 12-February-2022].

Winston, Patrick (2010). *Lecture 16: Learning: Support Vector Machines | Artificial Intelligence | Electrical Engineering and Computer Science | MIT OpenCourseWare*. https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-034-artificial-intelligence-fall-2010/lecture-videos/lecture-16-learning-support-vector-machines/. Accessed: 2022-02-11.

Yates, Roy D. and David J. Goodman (2005). *Probability and stochastic processes : a friendly introduction for electrical and computer engineers*. John Wiley and Sons, p. 519. ISBN: 9780471452591. DOI: 10.1002/9783527683147.

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede and Ivo Pletikosić. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

https://bitbucket.org/amiede/classicthesis/

Final Version as of February 12, 2022 (classicthesis v4.6).