Adaptive Learning for Reliability Analysis using Support Vector Machines

Nick Pepper
UQlab, Dept of Aeronautics, Imperial College London, UK, E-mail: nicholas.pepper16@imperial.ac.uk

Luis Crespo
NASA Langley Research Center, Hampton, VA, 23681, USA

Francesco Montomoli
UQlab, Dept of Aeronautics, Imperial College London, UK

A novel algorithm is presented for adaptive learning of an unknown function that separates two regions of a domain. In the context of reliability analysis these two regions represent the failure domain, where a set of constraints or requirements are violated, and a safe domain where they are satisfied. The Limit State Function (LSF) separates these two regions. Evaluating the constraints for a given parameter point requires the evaluation of a computational model that may well be expensive. For this reason we wish to construct a meta-model that can estimate the LSF as accurately as possible, using only a limited amount of training data. This work presents an adaptive strategy employing a Support Vector Machine (SVM) as a meta-model to provide a semi-algebraic approximation of the LSF. We describe an optimization process that is used to select informative parameter points to add to training data at each iteration to improve the accuracy of this approximation. A formulation is introduced for bounding the predictions of the meta-model; in this way we seek to incorporate this aspect of Gaussian Process Models (GPMs) within a SVM meta-model. Finally, we apply our algorithm to two benchmark test cases, demonstrating performance that is comparable with, if not superior, to a standard technique for reliability analysis that employs GPMs.

Keywords: Reliability Analysis, Active learning, Support Vector Machines, Failure Probability

1. Introduction

A key requirement for engineering designs is that they offer good performance across a range of uncertain conditions while exhibiting an admissibly low probability of failure. In reliability analysis we want to assess the probability of the system violating a set of performance requirements cast as inequality constraints. Denote the following: \( x \) as a number of uncertain parameters with joint density \( f_x(x) \) with \( x \in X \subseteq \mathbb{R}^n \); \( y = M(x) \) as the system response with \( y \in Y \subseteq \mathbb{R}^m \); and \( g(x) = M(x) - y_0 \leq 0 \) with \( g : \mathbb{R}^n \rightarrow \mathbb{R}^m \) as a set of requirements imposed upon the system. The physical space \( X \) is divided into a safe domain, \( \{ x : g(x) \leq 0 \} \), and a failure domain \( \{ x : g(x) > 0 \} \), with the two separated by the limit state function (LSF) \( \{ x : g(x) = 0 \} \). The probability of failure, \( P \), is defined as:

\[
P[g(x) > 0] = \int_{g(x)>0} f_x(x) dx. \tag{1}
\]

Evaluating this expression is often difficult since it entails performing a multi-dimensional integral over a complex integration domain, thereby requiring an approximation. In many cases \( M \) is cheap to evaluate so the probability of failure can be readily estimated through Monte Carlo sampling. A Monte Carlo sampling approximation to the probability of failure is:

\[
P = \frac{1}{n_{mc}} \sum_{i=1}^{n_{mc}} I(g(x^{(i)}) > 0) \tag{2}
\]

where \( I \) is an indicator function (1 if \( g(x) > 0 \), 0 otherwise), \( P \) the estimated probability of failure, and \( n_{mc} \) represents the number of Monte Carlo samples. \( x^{(i)} \) denotes the \( i \)th Monte Carlo sample, drawn from \( f_x \). The resulting approximation is subjected to sampling error because \( n_{mc} \) is finite. However, when \( M \) is expensive to evaluate and \( n_{mc} \) is large, the cost of \( (2) \) becomes inadmissibly high. This has motivated the development of methods for reliability analysis which replace \( M \) with a meta-model based on a training set of expensive model evaluations. The meta-model, which is cheap to evaluate, is then used to estimate \( P \) through Monte Carlo sampling. This model, which is trained using a limited number of function evaluations of \( g(x) \), is given by \( \hat{g}(x) \). Therefore, a Monte Carlo approximation to the failure probability based on \( \hat{g}(x) \) will suffer from both sampling error and response error. Gaussian Processes, Artificial Neural Networks, Polynomial
Chaos, and response surface methods have been commonly used in the literature as surrogates to obtain $g(x)$.

A surrogate model for reliability analysis will perform well as long as the LSFs $g(x) = 0$ and $\hat{g}(x)$ are sufficiently close. Large offsets in the response away from these functions will be immaterial. This consideration enables casting the surrogate modeling problem as the problem of creating a 2-classifier. These two categories correspond to points falling in either the safe domain or the failure domain. This approach has several advantages: firstly, training a meta-model as a classifier, rather than directly learning $M(x)$ or $g(x)$, allows discontinuous or binary responses to be efficiently handled. Secondly, multiple constraints or failure modes, i.e. $n_g \geq 1$, may be reduced to a single decision function.

Naturally, the use of a meta-model will sacrifice some accuracy in favour of reducing the computational cost regardless of which strategy is used. For this reason there has been much interest in developing adaptive methods for reliability analysis, which seek to improve the accuracy of a meta-model iteratively, by adding additional model evaluations to the dataset. This article proposes a strategy for adaptively training a Support Vector Machine (SVM) classifier, according to a limited number of evaluations of $g(x)$, while accounting for uncertainty in the learned function.

There are several challenges inherent to adaptive learning with SVMs that the algorithm presented in this paper aims to address. Firstly, it is well recognised in the literature that the most informative candidates to add to the training set lie on the estimated LSF (Basudhar and Missoum (2010); Pan and Dias (2017)). However, the non-linear nature of the LSF can make it challenging to efficiently restrict the search to such a subspace. Secondly, while SVMs may generally scale more efficiently to higher dimensional physical spaces and larger training sets than Gaussian processes, an estimate from a Gaussian Process is accompanied by the predicted variance of the Gaussian Process Model (GPM) uncertainty function is analogous to the predicted variance of the Gaussian Process Model (GPM) (see, e.g. Echard et al. (2011)). In this way we seek to combine the flexibility and efficiency of SVMs when applied to reliability analysis, with the desirable attributes of Gaussian Processes when applied to adaptive learning.

The remainder of this paper is structured as follows: in the next section we discuss the theoretical background of SVMs; the proposed algorithm for adaptive learning for reliability analysis using SVMs is then described; and finally the algorithm is implemented for two benchmark test cases in which the performance of the algorithm is compared with an established adaptive learning method for reliability analysis that employs GPMs.

2. Support Vector Machines for Reliability Analysis

Approaches that employ Support Vector Machines (SVMs) to estimate the LSF formulate the problem as a two class classification problem, as opposed to developing an explicit surrogate model for $g(x)$. Membership of the two classes depends on whether the inequality $g(x) > 0$ is satisfied. We first consider a case in which the failure and safe domains are linearly separable. Using a set of $n$ labelled training points $D = \{ (x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)}) \}$, where the class labels $y^{(i)} \in \{-1, 1\}$ indicate whether the inequality constraints are satisfied or violated, an SVM aims to approximate the LSF by finding the hyperplane with maximal margin that separates the two classes:

$$\mathbf{w}^{\top} \mathbf{x} - b = 0,$$

where $\mathbf{w} \in \mathbb{R}^{n_x}$ is the normal to the hyperplane and the bias, $b$, is a scalar. This is illustrated in Figure 1 for a $n_x = 2$ physical space.

A set of constraints are imposed $g^{(i)}(\mathbf{w}^{\top} \mathbf{x}^{(i)} - b) \geq 1$ for $i = 1, \ldots, n$ that have the effect of defining a margin of width $\frac{2}{\|\mathbf{w}\|}$ within which no training data falls. The separating hyperplane containing the largest margin is given by the quadratic optimization program:
where \( n \) \( \alpha \) port vectors. The associated Lagrange multipliers, \( \alpha_i \), can be found by using quadratic programming from the joint distribution of the LSV i.e., those for which \( y \) \( \alpha \) sample, categorised into two classes. Samples that lie on the boundary find a separating hyperplane that maximises the width of the margin between training data that is linearly separable. However, this might not be the case in many practical applications. In such cases a transformation, \( \Phi : X \rightarrow Z \), is used to map the training data from physical space, \( X \), to a higher dimensional feature space, \( Z \in \mathbb{R}^{n_z} \), where the data is linearly separable and a separating hyperplane can be found:

\[
\begin{align*}
\max & \quad \sum_{i=1}^{n_{sv}} \alpha_i y^{(i)} \\Phi(\bar{x}^{(i)}) \\
\text{subject to} & \quad \sum_{i=1}^{n_{sv}} \alpha_i = 1, \quad \alpha_i \geq 0, \quad i = 1, \ldots, n
\end{align*}
\]

Fig. 1.: A trained Support Vector Machine (SVM) finds a separating hyperplane that maximises the width of the margin between training data that is categorised into two classes. Samples that lie on the margin are referred to as the Support Vectors.

By reformatting this problem using the Lagrangian dual we find that the only non-zero Lagrange multipliers correspond to the subset of points in the training data that lie on the boundary of the LSV i.e., those for which \( |w^\top x - b| = 1 \). These parameter points are referred to as the support vectors. The associated Lagrange multipliers, \( \alpha_i \), can be found by using quadratic programming to solve the Lagrangian dual, a convex optimization problem. The classification of an unlabelled sample, \( x \), is determined by the sign of its distance from the separating hyperplane, \( s(x) \):

\[
s(x) = w^\top x - b, \quad w = \sum_{i=1}^{n_{sv}} \alpha_i \bar{y}^{(i)} \bar{x}^{(i)} \quad (5)
\]

where \( n_{sv} \) refers to the number of support vectors and \( \bar{x}^{(i)} \) the \( i \)th support vector with class label \( \bar{y}^{(i)} \). The separating hyperplane, given by \( s(x) = 0 \), is the approximation to the LSF; with the sign of \( s(x) \) used to determine class membership. The LSV is defined as the subspace \( \{ x : -1 \leq s(x) \leq 1 \} \). The decision function may be evaluated inexpensively, allowing the probability of failure to be estimated using \( n_{mc} \) Monte Carlo samples drawn from the joint distribution \( f_x(x) \):

\[
P = \frac{1}{n_{mc}} \sum_{i=1}^{n_{mc}} \mathcal{I}(s(\bar{x}^{(i)}) > 0), \quad (6)
\]

where \( \bar{x}^{(i)} \) is a Monte Carlo parameter point drawn from \( f_x(x) \).

The formulation described above can be used to find the maximally separating hyperplane in physical space, provided that the training points are linearly separable. However, this might not be the case in many practical applications. In such cases a transformation, \( \Phi : X \rightarrow Z \), is used to map the training data from physical space, \( X \), to a higher dimensional feature space, \( Z \in \mathbb{R}^{n_z} \), where the data is linearly separable and a separating hyperplane can be found:

\[
w^\top \Phi(x) - b = 0, \quad w = \sum_{i=1}^{n_{sv}} \alpha_i \bar{y}^{(i)} \Phi(\bar{x}^{(i)}) \quad (7)
\]

where \( w \in \mathbb{R}^{n_z} \). The form of the Lagrangian dual is such that in theory it is not necessary to specify \( \Phi \) explicitly in order to find the Lagrange multipliers; only the kernel of the transformation, \( \mathcal{K}(p,q) = \Phi(p)^\top \Phi(q) \), \( p, q \in X \) need to be defined. This is referred to as the ‘kernel trick’ and allows transformations to infinite dimensional feature spaces through the use of Gaussian kernels. However, in this work we restrict our attention to polynomial feature maps, which are of finite dimension, to not only work on a finite dimensional feature space but also to use polynomial optimization strategies. This mapping is parametrized by the polynomial degree of the transformation and the number of monomials that are kept. In particular we use polynomial mappings with kernels:

\[
\mathcal{K}(p,q) = (1 + p^\top q)^d, \quad (8)
\]

where the order, \( d \), of the kernel is user defined. For instance, a polynomial kernel of degree \( d = 2 \) and \( n_x = 2 \) corresponds to the feature map:

\[
z = \Phi(x = \{x_1, x_2\}) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1 x_2 \end{bmatrix}. \quad (9)
\]

The inverse feature map, \( \Phi^{-1} \) is found using the monomial terms of degree 1, in this case:

\[
x = \Phi^{-1}(z) = \frac{1}{\sqrt{2}} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}. \quad (10)
\]

Figure 2 demonstrates how a polynomial transformation may be used to lift training data...
from a one-dimensional physical space to a two-dimensional feature space where it becomes linearly separable. In this case a separating plane in physical space does not exist. However, the polynomial transformation $\Phi(x) = [x^2 \ x]^T$ may be used to lift the data to a two-dimensional space where the data is linearly separable and a maximally separating hyperplane may be found. Note that the entire physical space is mapped into a polynomial manifold in feature space. Every point at the intersection between this manifold and the separating hyperplane $s(z) = 0$ will map to the LSF, $g(x) = 0$. In Figure 2 there are two such points, with the vertical construction lines indicating their positions in physical space. However, in higher dimensional feature spaces there will be infinitely many such points, with the estimate of the LSF corresponding to the locus of points on the manifold which intersect the separating hyperplane. In the next section we describe an adaptive learning algorithm which at every iteration selects the most informative point from among these to label and add to the training data.

**3. Algorithm for Adaptive learning with Support Vector Machines**

In this work we propose an algorithm for identifying a parameter point driven by an optimality criterion. The response of the expensive code is then evaluated at such a point, and the corresponding outcome is used to expand the training dataset which will be used in the following iteration. There are three desirable properties for a candidate training point to possess:

1. Lying on the estimated LSF
2. Attaining relatively high likelihood
3. Attaining comparatively large uncertainty function value

The range of failure probabilities described above can be computed at any iteration. The algorithm will proceed until the spread of this interval is below a given threshold. In the following sections we discuss how the proposed algorithm selects candidates with these desirable features.

**3.1. Selecting candidates on the LSF**

In general, the most informative candidates lie on the estimated LSF. If an SVM meta-model is employed, then these points will be those that satisfy $\{x : s(x) = 0\}$. For numerical reasons this equality is typically relaxed to the inequality constraint $\{x : |s(x)| \leq \epsilon\}$, where $\epsilon$ is a tolerance on the distance of the candidate from the separating hyperplane in feature space. Identifying points on the LSF by exploring its physical space representation might be difficult. For this reason we introduce a formulation for exploring the LSF in feature space. Recall from equation (7) that the SVM fits a separating hyperplane between the labelled data in feature space with normal $w$. Through Gram-Schmidt orthogonalisation it is possible to develop an orthonormal basis that spans this separating hyperplane:

$$\mathbf{z} = \sum_{i=1}^{n_z-1} \gamma_i \mathbf{e}^{(i)} + \mathbf{e}^{(0)} \quad (11)$$

where $\mathbf{e}^{(i)} \in \mathbb{R}^{n_z}$, $i = 1, \ldots, n_z - 1$ represent an orthonormal basis in $Z$, $\gamma \in \Gamma \subseteq \mathbb{R}^{n_z-1}$ the associated weights, and $\mathbf{e}^{(0)} \in \mathbb{R}^{n_z}$ a vector connecting the origin and the hyperplane. By definition, any choice of the coefficients $\gamma$ in this basis will produce a point that lies on the separating hyperplane in feature space. However, the region of interest is the intersection between this hyperplane and the mapping of physical space onto feature space, i.e. $\mathbf{z} = \Phi(\Phi^{-1}(\mathbf{z}))$ where $\mathbf{z}$ is in (11).

**3.2. Finding sets containing points with high likelihood**

The likelihood of a point in physical space is quantified through the likelihood function $L(x)$, which is known prior to the reliability analysis. The joint density is an example of such a likelihood function. The high likelihood region near the LSV can be prescribed in terms of super-level sets of the joint density. For instance, in the case of Gaussian joint densities these level sets are ellipsoids, for which a closed-form expression is readily available. Alternatively, if a closed form expression for this contour is unavailable then we propose constraining the search to lie within the...
region satisfying the inequality \( \hat{L}(x) \geq \underline{L} \), where \( \underline{L} \) represents a minimum admissible likelihood value. Note that this region might be multiply connected and non-representable in closed form. This tolerance is calculated as a function of the maximum value that the likelihood function takes on the estimated LSF, which we denote \( \hat{L} \). This quantity is found through the solution of the optimization problem:

\[
\max_{\gamma, \zeta} \hat{L}(\Phi^{-1}(z)) \quad \text{such that} \quad ||z - \Phi(\zeta)||^2 \leq \epsilon, \ z = \sum_{i=1}^{n_x-1} \gamma_i e^{(i)} + e^{(0)},
\]

where \( \gamma \in \mathbb{R}^{n_x-1} \) and \( \zeta \in \mathbb{R}^{n_x} \). The dummy vector \( \zeta \in X \subseteq \mathbb{R}^{n_x} \) is used in a manner analogous to Lagrange multipliers. Hence, the solution to (12) yields a parameter point, \( \zeta^* \), that is in the physical space, close to the LSF, and for which the likelihood is maximal. Note that by assuming \( \epsilon > 0 \) we relax what should nominally be an equality constraint. With \( \hat{L} \) determined through optimization, a simple formulation to guarantee exploration of regions with relatively high likelihood is to impose the inequality constraint \( \hat{L}(x) \geq \underline{L} = \kappa \hat{L} \), where \( \kappa < 1 \) is a positive constant specified by the user.

### 3.3. Promoting exploration and quantifying predictive uncertainty

Having introduced a formulation to explore the separating hyperplane in feature space, with a set of constraints that ensure exploration of regions in \( X \) where the likelihood function exceeds a threshold, we introduce a function for quantifying the separation of a candidate parameter point from the training set in \( X \). We term this function the uncertainty function, \( U(x; D_x) \), where \( x \in \mathbb{R}^{n_x} \) represents the candidate parameter point in \( X \) and \( D_x = \{ x^{(1)}, x^{(2)}, ..., x^{(n)} \} \in \mathbb{R}^{n_x \times n} \) the subsequence of the training data corresponding to the locations of the training points in \( X \). In addition to quantifying the informativeness of the candidate point, the uncertainty function may also be used to quantify the approximation error in the failure probability estimate resulting from only using a limited number of training points. In so doing, the uncertainty function is analogous to the variance in a GPM and is intended to marry this feature of GPMs with the efficiency with which SVMs scale with \( n_x \) and \( n \). Any non-negative function which at any fixed value of \( x \) takes on values proportional to the distance from that \( x \) to the closest element of \( D_x \) can be used as an uncertainty function. This property can be written as:

\[
U(x; D_x) = \min_i \mathcal{D}(x; x_i^{(0)}),
\]

where \( \mathcal{D} \) is any norm. In this article we use the Mahalanobis distance given by:

\[
\mathcal{D}(p; q) = (p - q)^\top W(p - q),
\]

where \( W \in \mathbb{R}^{n_x \times n_x} \) is a positive definite matrix of weights. In the case of \( W = I \) this distance is equivalent to the squared Euclidean distance. Candidates in \( X \) that maximise the uncertainty function are considered to be the most informative and are identified through an optimization process:

\[
\max_{\gamma, \zeta} U(\Phi^{-1}(z); D_x) \quad \text{such that} \quad ||z - \Phi(\zeta)||^2 \leq \epsilon, \ L(\Phi^{-1}(z)) \geq \underline{L},
\]

\[
z = \sum_{i=1}^{n_x-1} \gamma_i e^{(i)} + e^{(0)}.
\]

Hence, we seek a parameter point that maximizes the uncertainty function while attaining a sufficiently large likelihood. In general, the optimization problem described here is likely to be non-convex and for this reason a Sequential Quadratic Programming (SQP) algorithm is run from several initial conditions in order to select the best point.

In practice it may not be feasible to run an adaptive algorithm to convergence due to the computational expense in evaluating \( g(x) \) at each iteration. For this reason, it is convenient to quantify the uncertainty in the empirical estimate of the failure probability at each iteration, which could be expressed as the bounded interval \([P_1, P_n]\). The formulation of a SVM does not provide a means to explicitly calculate this interval; we propose a method that uses the available knowledge of the requirements \( g \), together with the uncertainty function. This data corresponds to the evaluation of the requirements for the parameter points in \( D_x \), which we denote: \( G = \{ g(x^{(i)}) \}, i = 1, \ldots, n \).

Firstly, we use an empirical function to approximate \( g \) as a function of \( s \), which we denote \( \hat{g}(s) \). Recall that \( s \) is itself a function of the distance from the separating hyperplane in feature space. \( \hat{g}(s) \) is an increasing function that passes through the origin. We represent \( \hat{g} \) as the weighted sum of a basis of functions that share these properties, for instance in this work we choose:

\[
\hat{g}(s) = \begin{cases} 
\beta^\top f(s), & s \geq 0 \\
\delta^\top f(s), & s < 0
\end{cases}
\]

where \( f(s) = [s, s^3, \text{erf}(s)]^\top \) and \( \beta, \delta \in \mathbb{R}^3 \). This contains non-negative weighing coefficients. This
format guarantees that \( \hat{g} \) and \( s \) have the same sign. Alternative forms can be used instead. \( \beta \) and \( \delta \) can be found through a least squares fit to the training data, using \( G \) and the evaluation of \( s(x) \) for the parameter points in \( D_x \). We combine the empirical approximation for \( \hat{g}(s) \) with the uncertainty function to estimate the interval \([P_l, P_u]\) through:

\[
P_u = \frac{1}{n_{mc}} \sum_{i=1}^{n_{mc}} I(\hat{g}_u(s(\hat{x}^{(i)})) > 0),
\]

\[
P_l = \frac{1}{n_{mc}} \sum_{i=1}^{n_{mc}} I(\hat{g}_l(s(\hat{x}^{(i)})) > 0),
\]

where \( I \) refers to the indicator function defined above, \( \hat{g}_u(s(\hat{x}^{(i)})) \) and \( \hat{g}_l(s(\hat{x}^{(i)})) \), which represent a pessimistic and optimistic estimate of \( \hat{g} \) for the \( i \)th Monte Carlo sample, respectively, are determined as:

\[
\hat{g}_u(s(\hat{x}^{(i)})) = \hat{g}(s(\hat{x}^{(i)})) + \rho_u U(\hat{x}^{(i)}; D_x),
\]

\[
\hat{g}_l(s(\hat{x}^{(i)})) = \hat{g}(s(\hat{x}^{(i)})) - \rho_l U(\hat{x}^{(i)}; D_x),
\]

where the coefficients \( \rho_u \) and \( \rho_l \) are chosen by the user to make \( U \) and \( \hat{g} \) comparable. We suggest defining these coefficients as a percentage of the values that \( \hat{g} \) takes at the boundaries of the LSV (we choose 90% here). The set \( \{ s : g(s) > 0 \} \) corresponds to the failure domain, therefore a positive shift will result in a more pessimistic estimate of \( P \) as some Monte Carlo samples for which \( \hat{g}(s) \leq 0 \) will be shifted to the upper half plane. Conversely, subtracting the second term corresponds to a more optimistic of \( P \) (recall that \( U \) is a non-negative function). We illustrate this in Figure 3 for two Monte Carlo samples, where \( \hat{g}(s) \) is estimated from a training set containing 10 parameter points. The formulation described in (17) and (18) has several desirable qualities:

1. The interval is based on \( U \) and is a function of all of \( D_x \), as opposed to an interval based on the LSV.
2. \( g(s) \) is smooth of discontinuities since no training data falls in the LSV.
3. The algorithm is guaranteed to decrease the width of the interval \([P_l, P_u]\) as \( \hat{g} \) and the LSV tend to 0 as more points are added.

Having discussed in detail how the formulation of the adaptive sampling algorithm is intended to promote the selection of an informative parameter point to add to the training data at each iteration, we present the corresponding algorithm next:

**Algorithm 1 (Adaptive learning with SVMs for reliability analysis):**

Inputs: Training dataset \( D = \{ (x^{(i)}, y^{(i)}) \} \), \( G = \)

1. Train a SVM meta-model using the training dataset.
2. Generate the orthonormal basis \( e \) & constant vector \( e^{(0)} \) in \( Z \).
3. Determine \( \mathcal{L} \) through (12), calculate \( \mathcal{L} \).
4. Estimate \( P, P_u, \) & \( P_l \) through (6), (17) & (18).
5. If \( |P_u - P_l| < \epsilon \), stop. Otherwise, continue.
6. Determine next parameter point \( \zeta^* \) through (15).
7. Evaluate \( g(\zeta^*) \).
8. Update \( D \) & \( G \). Go to step 1.

Recall that if closed form expressions for contours of equal likelihood exist, for example ellipsoids in the case of Gaussian joint densities, then these can be used as a constraint on the likelihood, making step 3 unnecessary. The convergence of \( P_u \), which represents a worst case estimation of the failure probability given the available data, can be used as a stopping criterion for the algorithm. We suggest a stopping criterion based on the interval value of \( P_u \) and \( P_l \), i.e., \( |P_u - P_l| < \epsilon \).

In the next section we apply the above algorithm to two benchmark examples and compare its performance against an adaptive method employing Gaussian Process models.

**4. Validation with Synthetic Test Cases**

In this section we apply the algorithm for adaptive learning with SVMs presented above, which we refer to here as AL-SVM for convenience, to two synthetic test cases. In both instances we compare the performance of AL-SVM to AK-MCS, an adaptive learning algorithm that utilises GPMs.
More details on AK-MCS may be found in Echard et al. (2011).

4.1. Four branch function

The first test case involves a $n_x = 2$ physical space constrained by two polynomial and two linear requirements referred to as the four branch function:

$$
g(x) = \max \left\{ \begin{array}{c}
3 + 0.1(x_1 - x_2)^2 - \frac{1}{\sqrt{2}}(x_1 + x_2) \\
3 + 0.1(x_1 - x_2)^2 + \frac{1}{\sqrt{2}}(x_1 + x_2) \\
x_1 - x_2 + \frac{6}{\sqrt{2}} \\
x_2 - x_1 + \frac{6}{\sqrt{2}}
\end{array} \right\}
$$

in which $\mu$ refers to the means and $\Sigma$ the covariances of the joint density. The four branch function is a useful test case as the geometry of the requirements in physical space can be easily visualised and are cheap to evaluate, allowing us to study the convergence of the two adaptive sampling methods to the failure probability estimated by Monte Carlo sampling, referred to as $P_{mc}$.

Monte Carlo samples are drawn from a multivariate Gaussian joint density $N(\mu, \Sigma)$, where:

$$
\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \text{and} \quad \Sigma = \begin{bmatrix} 1 & 0.3 \\ 0.3 & 1 \end{bmatrix},
$$

The number of iterations for the percentage deviation between $P$ and $P_{mc}$ to decrease to below a given tolerance are presented in Table 1. AL-SVM was implemented three times, with varying choices of $W$. It was found that the choice $W = I$, corresponding to the squared Euclidean distance, yielded the fastest convergence of the three. In any case, it appears that the convergence of AL-SVM is at least comparable to, if not faster than, AK-MCS with respect to the number of requirement evaluations needed. This speed of convergence comes at a cost: the mean CPU time per iteration for AL-SVM was 249s, compared to 25.2s for AK-MCS. This cost arises from the solution of the optimization described in (15). As has been discussed, this optimization is likely to be non-convex and for that reason a local solver is run from 12 separate starting points on each iteration. The computational cost of AL-SVM in its present formulation may be prohibitive for certain applications, where the timescale of each evaluation of the requirements may be on the order of seconds as opposed to minutes or hours.

Figure 4 illustrates the true LSF of the four branch function; the estimated LSF and $D_x$ at iteration 80; the location of the Monte Carlo samples; and the initial parameter points in the training dataset that was fed to both adaptive learning algorithms. The convergence of $P$ for both AL-SVM and AK-MCS are plotted in Figure 5, which also indicates the interval $[P_l, P_u]$ at each iteration.

### Table 1.

<table>
<thead>
<tr>
<th>Method</th>
<th>Iterations until $\epsilon \leq \epsilon_{tol}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL-SVM ($W = I$)</td>
<td>10% 5% 2%</td>
</tr>
<tr>
<td>AL-SVM ($W = \Sigma$)</td>
<td>38 60 73</td>
</tr>
<tr>
<td>AL-SVM ($W = \Sigma^{-1}$)</td>
<td>62 65 88</td>
</tr>
<tr>
<td>AK-MCS</td>
<td>57 80 97</td>
</tr>
</tbody>
</table>
4.2. Physical space with discontinuities

In the second test case we consider a simple feasible region corresponding to a square in a \( n_x = 2 \) physical space with two constraints:

\[
g(x_1, x_2) = \begin{cases} 
10 & \text{if } x_1 > 0.8 \lor x_2 > 0.8 \\
-2 & \text{otherwise}
\end{cases}
\]  

(21)

where \( x_1 \) and \( x_2 \) again represent the uncertain inputs. Monte Carlo samples for these inputs are generated from two independent uniform distributions \( U(0, 1) \). This LSF is plotted in the top panel of Figure 6, together with the estimated LSF from AL-SVM and the locations of the parameter points added to the training set by the algorithm. The locations of the 5 parameter points in the initial training set are ringed in black. While the geometry of the feasible region is straightforward, it can be considered a challenging case for adaptive methods that employ meta-models to learn \( g(x) \) directly as \( g(x) \) changes discontinuously across the LSF. Such discontinuities can occur frequently in real-world problems, for example in buckling behaviour.

The convergence of AL-SVM is plotted in the bottom panel of Figure 6. For this test case the weight matrix \( W = I \) was chosen. The AL-SVM approach converges to a value of \( P \) with a deviation from \( P_{mc} \) of less than 2% after 17 iterations. On the other hand, a naive implementation of AK-MCS fails to converge as the variances shrink near the discontinuities. This test case demonstrates an inherent advantage of classifier based approaches to reliability analysis: that they can be used to approximate constraints that vary discontinuously without modifying the algorithm.

5. Conclusions

We have presented a novel algorithm for adaptive learning of an LSF, using SVMs to construct a semi-algebraic approximation to the LSF. The method identifies informative parameter points through an optimization process in which the uncertainty function, a polynomial function that depends on the proximity of a candidate parameter point to the training data, is maximised. By exploiting the properties of polynomial kernel functions, the algorithm executes a constrained search in feature space. The predictive uncertainty of the SVM meta-model is expressed using an analytical expression for the constraints as a function of distance from the separating hyperplane and the uncertainty function. We applied the algorithm to two benchmark cases in \( n_x = 2 \) physical spaces and demonstrated performance that was consistent with, if not superior to, the performance of a method utilising GPMs.

While these results are encouraging there are several aspects in which the work presented here may be extended. Firstly, an application of the method to a higher-dimensional physical spaces is crucial and may be instructive to the future development of the method. Secondly, we do not fully exploit the polynomial formulation of the algorithm here. Implementing the method using polynomial optimization methods will improve the performance of the algorithm. Finally, the performance of the algorithm may benefit from a subroutine for tuning the polynomial degree of the SVM with iteration.

References

