Fast flutter evaluation of very flexible wing using interpolation on an optimal training dataset

Norberto Goizueta*, Andrew Wynn† and Rafael Palacios‡
Department of Aeronautics, Imperial College London, SW7 2AZ

Machine learning strategies can be efficiently used to accelerate the exploration of the design space or flight envelope of highly flexible aeroelastic systems. In this paper, we explore the use of interpolation between parametric state-space realizations to, with few true systems sampled in the parameter space, produce with adequate accuracy a state-space model anywhere in the parameter space. The location of the sampling points is shown to be decisive thus the selection of these points takes the focus in this work. Several approaches are explored, putting emphasis on adaptive schemes that locate the optimal points in the parameter space that are needed to capture the changing system dynamics. Since the evaluation of the true system is costly, optimization techniques based on statistical surrogate models are sought, which need to be trained but are effective in locating the best locations to use as sampling data. A novel method inspired by Bayesian optimization is used to make the most out of a limited number of known state-spaces by taking different combinations as training and testing data of the statistical surrogate, leading to not only an accurate interpolation framework but also to a reduction of 50% of the number of costly full system evaluations compared to a standard Bayesian optimization set-up. These methods are demonstrated on the Pazy wing, a very flexible wing with a complex stability envelope, whereby we produce a very accurate representation of the flutter envelope at a reduced computational cost.

I. Introduction

Loads and stability analyses across the flight envelope of any aircraft is a crucial step in its design and certification process for nominal and gusty off-design conditions. For conventional aircraft certified under FAR23 or FAR25, this process requires of hundreds of thousands of evaluations of the system and is a computationally expensive process [1]. In addition, engineering judgment based on past experience on similar aircraft models is typically employed to reduce the number of evaluations [2] and, clearly, is no longer an option for new concepts, such as high-altitude long-endurance aircraft, where nonlinear and/or high-fidelity tools are necessary.

For very flexible aircraft, where the design must be undertaken from an aeroelastic perspective, models may become significantly more complex, given the need of a coupled analysis and the use of geometrically nonlinear tools to capture effects driven by large structural deformations. The use of high fidelity models such as coupled CFD/3D FEM can soon become prohibitive and the problem of determining the loads envelope may become simply unfeasible*. Therefore, as very flexible aircraft tend to share the common design aspect of high aspect ratio wings where the flow is mostly attached, potential flow methods for the aerodynamic solution and one-dimensional nonlinear beam structural models offer an suitable option to simulate these vehicles. Although these methods are considerably more affordable to run on typical hardware, the load envelope evaluation may still be prohibitive, as the nonlinear solutions required to capture the large deformation effects make necessary the use of time domain simulations to determine gust load envelopes [3].

Downselection of load cases to speed up the loads analysis problem is a on-going endeavor, with several different approaches possible. The use of SVD methods to remove redundant load cases from a large dataset of possible static loads combinations [4] is an option, although performing such operations on very large matrices (which contain the load case information) may be taxing. Statistical approaches (like Kriging) to build surrogate models and efficiently sample the design space are common practice, for instance in the prediction of the worst-case loads for “1-cosine” gusts [2] or to estimate model uncertainty in system outputs in the presence of random uncertainties [5] can be considered most similar

*PhD Student, CAGB 308, South Kensington Campus (norberto.goizueta13@imperial.ac.uk)

†Reader in Control and Optimization, CAGB 340, South Kensington Campus (a.wynn@imperial.ac.uk)

‡Professor in Computational Aeroelasticity, CAGB 338, South Kensington Campus. AIAA Associate Fellow (r.palacios@imperial.ac.uk)

*Nonetheless, it is possible that the interpolation methods presented in this paper, which ultimately are data-driven physics-based methods (what today’s buzzwords refer to as machine learning), enable the efficient use of high-fidelity tools for large parameter space explorations.

to what we will present herein. Design space exploration for multi-disciplinary optimization (MDO) of aircraft that includes certification constraints aims to incorporate dynamic loads into the MDO, as opposed to pseudostatic loading which could underestimate component loads [6]. To accomplish such exploration without prohibitive costs calling physical based models, surrogates are used [7] that are sampled using design of experiments methods, such as Latin Hypercube sampling (LHS). The number of sampling cases, which is a key-driver of the MDO's cost, can be reduced using adaptive sampling to refine the surrogate models based on residual error, information gain or weighted-information gain [8]. Other applications of adaptive sampling can be found in surrogate-based optimization [9], which uses optimal LHS to obtain adequate distributions of training points in the design space; or information-gain-based approaches used in global optimization [10] and optimal experiment design. For the latter, we find cases most similar to our application such as seeking an optimal experimental set-up by placing sensors at places of maximum expected information gain [11] or by finding using statistical models the sets of experiments that provide most information about a target set of parameters [12].

In this work, we seek to explore the design envelope in a more efficient manner by using linear analysis tools to determine stability and load limits without resorting to time domain simulations or statistical surrogates of the physical processes. This is based on the linearization of the aeroelastic systems, parametrized in terms of the flight condition, about the deformed (nonlinear) equilibrium state at a few locations across the design space. Then, by means of interpolation, the effects of changing the equilibrium shape from one point to another can be captured without needing to compute a new linearized system. Not only would this enable an efficient sampling of the flight envelope (the interpolation is performed between reduced order models), but also having a linearized system at any point in the design space allows for the use of linear analysis techniques, such as frequency analysis in the response to gusts, that would replace the need for costly and numerous time domain simulations and account for the dynamic loads and its effects.

This approach will be demonstrated on the Pazy wing [13], a very flexible wing designed for wind tunnel flutter experiments, whose stability envelope and flutter characteristics have been previously analyzed in [14]. The nonlinearities present in the Pazy wing model require the linearization and eigenvalue evaluation at each point of interest in the parameter design space in order to determine the flutter solution, which is a costly process. Thus, we seek with this approach to significantly reduce the computational time in which the flutter envelope is obtained, while also paving the way for loads analysis in an analogous manner. To such extent, we build upon the parametric reduced order modeling and interpolation framework for aeroelastic systems in [15] and include a strategy to efficiently sample the design parameter space to discover its stability properties. In the case presented, the flutter characteristics of the system are unveiled using a few training points and interpolating the rest of the parameter space. Compared against the analysis in [14] which took days to perform, interpolation can be performed in minutes, provided an adequate set of training of points is known, which makes it highly attractive for analysis during the design stage (where a rapid turnaround is important) and even to be used as part of optimization routines.

The crux of the matter is the selection of the interpolation training points. Initial exploration of the flutter envelope using the interpolation framework with regular grids typically employed in model interpolation problems [16] has uncovered the tremendous sensitivity to the choice of the sampling points. Thus we focus on two methods to perform an optimal sampling of the parameter space in order to obtain a training set that minimizes a user-defined error metric, and for this we turn to the statistical surrogates to model the interpolation error, since a direct evaluation is costly. For the Pazy flutter example in hand, an error metric based on the difference between interpolated and true aeroelastic eigenvalues is employed, and this drives a Bayesian sampling method that, with only a few initial testing points in the domain, generates expected errors across the entire space, such that those regions that present a larger difference are more finely sampled.

The framework is developed on SHARPy (Simulation of High Aspect Ratio aeroplanes and wind turbines in Python[†]) [17], our in-house open-source, nonlinear aeroelastic simulation toolbox. The design space is sampled at optimal training points, such that the interpolation error is minimized. At points where the true system is required (training and testing points), the nonlinear equilibrium condition is obtained [18], the system linearized [19] and reduced using Krylov subspace methods [15], resulting in a library of reduced order parametric models that will serve as basis for the interpolation. The Pazy wing example can be used as demonstrator to showcase the capabilities of the framework prior to using it on very flexible aircraft systems to evaluate the loads response across the flight envelope.

[†]<http://imperial.ac.uk/aeroelastics/sharpy>

II. Nonlinear Aeroelastic Solution, Linearization, Reduction and Compatibility

The process for obtaining reduced order, linear parametric aeroelastic models to be used for the interpolation scheme can be divided in four independent steps: i) finding the nonlinear equilibrium condition of the full-order aeroelastic model, ii) linearizing the dynamical system about the deformed equilibrium condition, iii) reducing the highly dimensional state-space using Krylov subspace methods and iv) projecting all systems onto a compatible coordinate system prior to interpolation. This process has been thoroughly detailed in [14], thus this section will serve as a brief outline and the reader is referred to the paper for full details.

The structural model within SHARPy's aeroelastic solver is based on a geometrically-exact composite beam formulation, with linear constitutive relations and nonlinear kinematic relations. The formulation is parametrized in displacements and rotations and applied by discretizing the beams in quadratic (3-node) finite elements [20, 21]. The discretized structural equations [22, 23], denoted with $(\bullet^S)$, that result from applying Hamilton's principle

$$\mathcal{M}(\eta) \begin{Bmatrix} \ddot{\eta} \\ \ddot{\beta} \end{Bmatrix} + \begin{Bmatrix} N_{\text{gyr}}^S(\eta, \dot{\eta}, \beta) \\ N_{\text{gyr}}^R(\eta, \dot{\eta}, \beta) \end{Bmatrix} + \begin{Bmatrix} N_{\text{stif}}^S(\eta) \\ \mathbf{0} \end{Bmatrix} = \begin{Bmatrix} N_{\text{ext}}^S(\eta, \dot{\eta}, \beta, \chi) \\ N_{\text{ext}}^R(\eta, \dot{\eta}, \beta, \chi) \end{Bmatrix}, \quad (1)$$

and also include — if necessary — the rigid body equations of motion $(\bullet^R)$, which are solved simultaneously about a reference node (not the center of gravity). The formulation is parametrized by the flexible (structural) degrees of freedom, $\eta_i \in \mathbb{R}^6$ for $i \in \{1, \dots, n_{\text{node}} - 1\}$ that includes the linear displacements and Cartesian rotation vector of each node; the rigid body linear and angular velocities represented by $\beta \in \mathbb{R}^6$ and the orientation of the structure, parametrized using a quaternion $\chi \in \mathbb{R}^4$ (the latter two only if rigid body motion is modeled). The mass matrix, $\mathcal{M}$, is a function of the instantaneous geometry. Finally, the vector of discrete gyroscopic, stiffness and external (aerodynamic, thrust, etc.) forces at the nodes is denoted by N , short for the forces and moments at the flexible degrees of freedom, N^S , and at the rigid body reference N^R .

The aerodynamics in SHARPy are solved using the Unsteady Vortex Lattice Method [24, 25], based on potential flow assumptions and with viscous effects confined to thin boundary layers that are neglected. The mesh consists of a lattice of vortex rings projected over the lifting surfaces of the structure, with the spanwise coordinates of the panels coincident with the structural nodes. The wake is modeled by a discretized sheet of vortex panels that can adopt any arbitrary shape (including wake roll-up effects). Control surface deflections are simulated by allowing the relevant panels to deflect, thus only having an aerodynamic impact (there is no structural element associated to them). The vortex rings induce a velocity on the field inversely proportional to their distance squared and whose strength (circulation) is computed by enforcing a non-penetration boundary condition at the vortex collocation point. This simultaneously introduces both a time dependency, as the distance between panels is dependent on the instantaneous geometry and kinematics of the lifting surfaces, and a nonlinearity since the aerodynamic influence coefficients require updating at every time step [19]. The circulation of the bound vortices is shed downstream onto the wake, which therefore carries a history of previous bound circulation and allows for unsteady aerodynamic effects. With the circulation of the vortices solved for, the steady and unsteady aerodynamic forces are computed: the former are calculated using Joukowski's theorem while the latter using Bernoulli's unsteady formulation, with the time derivative of the circulation calculated using finite differences [23, 26, 27].

A linearized aeroelastic model can then be built around an equilibrium reference condition under the assumption of small perturbations to the degrees of freedom. On the aerodynamic side, the linearization of the UVLM is performed analytically as in [19] assuming constant aerodynamic influence coefficients and a frozen, yet arbitrary, wake shape. The resulting discrete, linear time invariant (DLTI) system in state-space form

$$\begin{aligned} \mathbf{x}_a^{n+1} &= \mathbf{A}_a \mathbf{x}_a^n + \mathbf{B}_a \mathbf{u}_a^{n+1}, \\ \mathbf{y}_a^n &= \mathbf{C}_a \mathbf{x}_a^n + \mathbf{D}_a \mathbf{u}_a^{n+1}, \end{aligned} \quad (2)$$

atypically defines the input at a future time step $n + 1$ to account for the elliptical nature of the underlying Laplace equations of potential flow, by which a disturbance is felt instantaneously across the entire flow field. The system's state vector, $\mathbf{x}_a^n$ corresponds to perturbations to the bound and wake circulations, circulation time derivative and circulation at the previous time step. The inputs $\mathbf{u}_a^{n+1}$ are the lattice grid coordinates, velocities and external fluid disturbances (i.e. gusts). Finally, the output $\mathbf{y}_a^n$ is the vector of normalized aerodynamic forces at the lattice vertices. The size of these vectors is dependent on the number of UVLM bound panels, the number of wake panels and the number of panel vertices. Additional inputs can be introduced to the input vector, such as control surfaces deflections and velocities.

The number of panels is typically large since refined discretizations are often required in order to achieve convergence for flutter analyses [14, 19, 28]. Thus, the resulting linearized UVLM systems may be of the order of $O(10^4 - 10^5)$

states and $O(10^3)$ inputs. This prevents from the use of traditional linear analysis (eigenvalues or frequency response) on typical desktop hardware, therefore the need to turn to model reduction techniques. Krylov subspace methods are the model reduction procedure of choice, given their efficiency in dealing with very large systems (the method uses matrix-vector multiplications) and also since it is based on transfer function matching about a user-defined frequency. For aerodynamic systems such as the linear UVLM, where the low frequency spectrum is of interest, the match is performed about the steady-state gain, which is preserved by the ROM method and offers best results in that end of the spectrum. The reduced UVLM can now be coupled with the linearized structural system projected onto the modal coordinate space to yield a reduced order coupled aeroelastic discrete-time system. Since these are going to conform the training models for the interpolation scheme, and to avoid interpolating between discrete-time systems with different time-steps, the systems are transformed to continuous time systems using a bi-linear transformation [29]. Therefore, once transformed into continuous time, only features below one half of the Nyquist frequency, $\omega_N = \pi/\Delta t$, of the discrete time system are considered.

Once the library is complete, the final step is to project the systems onto a compatible coordinate system, which is possible because the change in dynamics of the system occurs smoothly (for instance there are no shockwaves). The reason for this is that the individual system states are reduced states (Krylov and modal states) that if interpolated directly could result in interpolation between states that represent different dynamics. The projection onto the compatible space on which the interpolation is performed is described in detail in [15] and is based on collecting the reduced order bases of all parametric systems and performing an SVD to retain a desired number of bases, which will be the most dominant across the parameter space, and use these to define a transformation onto the new coordinate space. Having done this, all training models share a common state vector and the interpolation may be performed directly on each element of the system's matrices. The interpolation scheme is then a choice depending on the dimension of the parameter space and on the selection of the training data (structured vs unstructured grids, for instance). This will be discussed in the following section.

III. Sampling Strategies

The interpolation framework is not complete without its parameter space sampling strategy, and indeed, without adequate sampling the results from the interpolation can be either physically meaningless or computationally untractable for a large parameter space. The significant sensitivity to the choice of training sets has driven the search for algorithms to choose the best suited models in the parameter space that minimize the interpolation error. However, we commence our discussion with a simple regular grid sampling of the parameter space to explore what effect the selection of the sampling points has, followed by approaches of increasing complexity to sample the parameter space.

A. Definitions

Prior to discussing the different sampling strategies, we introduce the notation and definition for the different system properties, sets of points used as interpolation training and testing points and error metrics. First, we define the p -dimensional parameter space $X \subset \mathbb{R}^p$,

$$X = [l_1, u_1] \times [l_2, u_2] \times \cdots \times [l_p, u_p] \quad (3)$$

where p is the number of parameters and the i -th parameter is assumed to be bounded between $[l_i, u_i]$. Then, we define the set $X_0 \subset X$ as a set of points in the parameter space at which a true system is known and is locally accurate in the vicinity of $x \in X_0$. For each of these true systems, acquired using SHARPy following the procedure in Sec. II, the transfer function is known and denoted as $G[x](s)$, which represents the true transfer function at the parameter point $x \in X_0$. For simplicity of the notation the Laplace variable s is omitted, resulting in $G[x]$.

In addition, we define the transfer function of an interpolated system at a point $x \in X$ as $G[x; Z]$, which has been computed using the set of training points $Z \subseteq X_0$ in the interpolation framework. Then, by definition of the interpolation method, we can state that

$$G[z; Z] = G[z], z \in Z \quad (4a)$$

$$G[x; Z] \neq G[x], x \in X \setminus Z. \quad (4b)$$

In other words, (4a) is stating that the interpolated system is equal to the true system at a point where the system is known and (4b) states that there may be an error in the approximation of a transfer function at a point not part of the training set of the interpolation framework. This is what we seek to quantify next: the error in the approximation to a true system using the proposed interpolation framework.

B. Error Quantification

We quantify the error in the approximation by means of a cost function, $\varepsilon = h(\cdot, \cdot)$, which is scalar-valued and possibly nonlinear, and may take the transfer functions of true and interpolated systems as

$$\varepsilon := |h(G[x], G[x; Z])| \in \mathbb{R}. \quad (5)$$

The choice of cost function is influenced by the scope of the problem. In the case in hand where we seek to predict the flutter boundary of an aeroelastic system we devise a cost function based on the eigenvalues of the system; alternatively, for a case seeking loads, for instance, one could use metrics of transfer functions of gust velocity input to wing root loads outputs or a combination of them both. A beneficial characteristic of the cost function is that it is chosen such that it has physical meaning and therefore a threshold can be established below which a training set may be deemed accurate.

Note we will employ for other system properties the same notation used for the transfer function G . Thus, for example, the set of eigenvalues of the true system at a point $x \in X$ will be $\lambda[x]$ and the interpolated counterpart given it has been obtained using the training set $Z \subseteq X_0$ will be denoted $\lambda[x; Z]$.

The currently explored cost function has been conceived for an interpolation scheme used to find the stability boundary of a system, i.e. where the eigenvalues have zero real part. Recall that we perform the interpolation on the state-space rather than the eigenvalues themselves in order to retain a realization that can be used to explore, for instance, the frequency response, etc. For this case, the interpolation cost function takes the N eigenvalues of the interpolated and true systems, $\lambda[x; Z]$ and $\lambda[x]$ respectively, and finds the Euclidean distance between corresponding eigenvalues, weighted based on the distance to the imaginary axes as

$$\varepsilon^{\text{flutter}}[x; Z] = \sum_{j=1}^N w(\zeta[x]_j) \|\lambda[x, Z]_j - \lambda[x]_j\|_2, \text{ for } x \in X_0 \quad (6)$$

where the weight $w(\cdot)$ is given by a Gaussian shaped function parametrised by the damping ratio of the current mode, $\zeta[x]_j$,

$$w(\zeta[x]_j) = \exp\left(\frac{-\zeta[x]_j^2}{2\zeta_\sigma^2}\right). \quad (7)$$

This weighting function highlights errors closest to the stability boundary, as modes that are highly damped (or have already crossed the imaginary axes in the case of multiple unstable modes, hence the symmetric shape) do not drive the cost function. The parameter ζ_σ is user-defined to mark the damping ratio range between which modes are most penalized. For a clearer representation, Fig. 1, illustrates the cost function and weighted-averaging method.

Note that the cost function can only be evaluated at those points in the parameter space for which an instance of the true system exists, i.e. $x \in X_0$.

C. Sampling Grids

Having quantified the error between interpolated and true system, all that is left is to choose the training set $Z \subset X_0$ that will be the basis of the interpolation framework. We present several methodologies to find Z in order of increasing complexity.

1. Non-adaptive grids

The simplest choice for Z is likely a regular grid, for which points are chosen at regular intervals across the parameter space. Thence, a Delauney triangulation is performed over the chosen points, the domain tessellated, and the interpolation is performed using either a Clough Tocher interpolation scheme for 2-dimensional parameter spaces (based on piecewise cubic, continuously differentiable Bezier polynomials on each triangle [30]) or a linear N -dimensional interpolation method, both of which are implemented in the standard Python library `scipy`.

Regular grids are cursed by dimensionality, and soon become untractable if a refined sampling is chosen in a large dimensional parameter space. This is the classic design of experiments problem and several options exist which offer adequate visibility into the parameter space without incurring in the cost of structured grids.

A notable approach to sample the parameter space is then a Latin Hypercube Sampling (LHS), which presents a distributed option for the choice of Z in particular when the dimensions of the parameter space are large. This sampling results in an unstructured grid which is then tessellated and used by the interpolation scheme.

(a) Example of euclidean distance based metric between eigenvalues shown on Argand diagram close-up of the Pazy wing.

(b) ζ_σ -normalized damping-ratio penalty function to weigh eigenvalue distance.

Fig. 1 Graphical representation of the cost function based on distance between true and interpolated eigenvalues.

2. Optimal sampling using Bayesian Optimization

The above sampling methods are overly simplistic and, depending on the problem, may not offer a training set Z that performs well across the entire parameter space. In particular, if there are largely changing dynamics across it and a more refined sampling is required in certain areas of the parameter space.

Therefore, we also seek an adaptive method to sample the parameter space. However, given that the underlying cost function $\varepsilon[x; Z]$ $x \in X$ cannot be computed for all x since it would require $X_0 = X$ and defeat the purpose of this exercise, we turn to Bayesian optimization (BO) to estimate the expected error which can be known for all $x \in X$. The objective is that the Bayesian optimizer maximizes the cost function and the point in the parameter space at which this occurs can then be added to the set of training points.

Bayesian optimization [31] is well-suited to find the optimal (maximum cost) location to add points to the training set, since it does not assume anything about the underlying cost function (which is expensive to evaluate) and does not compute derivatives. The successful optimization of takeoff maneuvers in SHARPy using BO [32] showcases the advantages of this method, hence we seek to use it for this problem.

The BO scheme depends on two constructions: a surrogate statistical model (Gaussian process) of the cost function which provides a mean and variance result across the entire domain, and an acquisition function which decides where to sample the domain next in order to update the statistical model [33]. The former gives the posterior probability distribution as known data is added and the latter uses the expected improvement, which samples the next location using a tradeoff between the current posterior mean and the uncertainty of the model.

The current BO uses the implementation in the open-source research code GPyOpt [34]. The BO is given an initial training set, Z_0 , which includes the edges of the parameter space such that the entire domain X falls within the convex hull of Z_0 and therefore no extrapolation schemes are needed. The BO posterior probability distribution is given by

$$\mu_\varepsilon[x; Z_0] = \mathbb{E}[\varepsilon[x; Z_0] \mid \varepsilon[X_s; Z_0]], \quad x \in X \quad (8)$$

anywhere in the domain. In other words, $\mu_\varepsilon[x; Z_0]$ is the expected error of an interpolation at $x \in X$ using Z_0 as training data set and given that the true error ε is known at X_s , where X_s corresponds to a set of new points at which the scheme has computed the true system. At these points, X_s , it can compute the error $\varepsilon[x_s; Z_0]$ and use this result to inform the Gaussian process (8). This is an iterative process, and points x_s are computed according to the expected improvement function. The output of the optimization is the maximum cost and the location at which it occurs, x^+ , and this becomes the next best point to add to the training set for the next iteration $Z_1 = \{x^+ \cup Z_0\}$.

The number of evaluations during the optimization, which determine the size of X_s , depends on the problem. It can be seen as testing data and the larger the testing set, the more accurate the prediction of the cost function is likely to be,

despite an increased computational cost.

The parameter space location where the maximum of (8) occurs, x^+ is then the next point to add to the training set of the interpolation framework, $Z_1 = \{x^+ \cup Z_0\}$. This subsequent iteration, which aims to maximize $\mu_\varepsilon[x; Z_1]$ has the benefit that it can update its posterior mean with the previously computed points X_s . Therefore, as iterations advance, X_s , becomes larger and estimates of the posterior mean become more accurate. This iteration can continue until either a maximum number of iterations (computational budget) is achieved, or the expected maximum error is below a user-defined threshold.

The limiting factor of this scheme is the large number of evaluations of the true system (which are the ones that this framework aims to minimize) that are used as testing data only in order to generate μ_ε , not making it onto the training set and thus not being used to improve the interpolation accuracy. As with any regression that depends on testing and training data, it is clear that the sweet-spot lies in a careful balance between the two and the authors have seen that this BO driven sampling ends up with a ratio (at best) of 6:1 testing to training points, enough to deem it inefficient albeit functional. This drawback has encouraged the development of the next sampling strategy, which aims to reuse a set of computed points as training and testing set in order to obtain a more informed Gaussian regression with the same number of full system evaluations.

3. Optimal sampling using a Bayesian approach with multipurpose sets

The previous approach of using a standard BO implementation has highlighted an important limiting factor. However, we can take inspiration from the statistical approach in finding the error function to increase its efficiency and reduce the testing to training data ratio. For such purpose, we use a pool of known true systems (recall these are the ones expensive to evaluate) from which we choose different combinations to, without computing any more, find with reasonable accuracy the location at which the maximum error occurs.

Before delving into how we use find the expected error in the parameter space, it is convenient to define two sets: the training-only set Z_i , which contains points that will only be used as interpolation training points; and the multipurpose set, M_i , which contains points that may be used as testing or training points in different combinations. The $(\bullet)_i$ subscript marks the current iteration. The following applies to these two sets: $Z_i \cup M_i = X_i$ and $Z_i \cap M_i = \emptyset$, i.e. all points for which the true system is known at every iteration, X_i , belong to either M_i or Z_i .

The training-only set, Z_i , is simple. It is fixed at each iteration and is initialized with the parameter space edge points such that all evaluation points fall within the convex hull of the training data. The multipurpose set, M_i , is slightly more complex. From M_i we desire to obtain two sets, the *tRaining* set R_j and the *teSting* set S_j , such that $S_j \cup R_j = M_i$. Numerous splitting options exist and in total J different sets can be obtained by selecting combinations of all the members in M_i . To avoid a large number of possible combinations J of sets in M_i , the sets are constrained in size such $|S_j| - |R_j \cup Z_i| < 2$, in other words, the testing and training data sets will be of the same size, or with one or two members more than the other. This ensures that the amount of training and testing data is balanced, and it will be explained why this is required. To further clarify the concept behind the different sets, a few subsets of the j possible combinations are shown in Fig. 2, which shows an example parameter space with training and testing points. Note how the points belonging to Z are fixed in both cases, whereas those in M_i can change their function, like the one in the vicinity of $(2.5^\circ, 60 \text{ m/s})$ which goes from being a testing point in S_j to a training point in R_{j+1} .

Therefore, at each iteration we find available J different S_j and R_j sets and a fixed Z_i set. For every combination $j \in \{1, \dots, J\}$, it is possible to find the expected error function at every point in the parameter space given the known value of the error function at the testing data points, S_j , for the interpolation framework that uses training points, $R_j \cup Z_i$, as

$$\mu_{\varepsilon,j}[x; R_j \cup Z_i] = \mathbb{E}[\varepsilon[x; R_j \cup Z_i] \mid \varepsilon[S_j; R_j \cup Z_i]], j \in \{1, \dots, J\}, x \in X. \quad (9)$$

This posterior mean of the expected error is obtained using a Gaussian process regression, as done for the BO, and thus in the same form as (9), the expected variance $\sigma_{\varepsilon,j}^2(x; R_j \cup Z_i)$ for $x \in X$ is also known. With the posterior mean and the variance at each one of the j -th sets, we construct two metrics that aggregate the information:

$$\mu_i(x) = \frac{1}{J} \sum_{j=1}^J \mu_{\varepsilon,j} \sigma_{\varepsilon,j}^2, \text{ and} \quad (10a)$$

$$\sigma_i^2(x) = \frac{1}{J} \sum_{j=1}^J \sigma_{\varepsilon,j}^2. \quad (10b)$$

Fig. 2 Examples of different combinations of training and testing datasets using the training-only and multi-purpose sets on the angle of attack-velocity parameter space. The sets correspond to the training-only Z set, and two arbitrary partitions (j and $j + 1$ of the multipurpose set into S_j and R_j .

These two equations tell important information about the current training set, Z_i . The maximum of (10a) indicates where in the parameter space the error in the interpolation framework is greatest, and the bias provided by the variance makes this function also be affected by model uncertainty while also reducing the likelihood of the maximum error being coincident with one of the known points. Since this is in essence where the worst case condition is located, it is a good point, $x_{\mu,i}^+$, to add to the training-only dataset for the subsequent iteration, $Z_{i+1} = \{x_{\mu,i}^+ \cup Z_i\}$. From (10b), it can be inferred where the maximum uncertainty in the Gaussian model is present, therefore, the location of its maximum point, $x_{\sigma,i}^+$, is added to the multipurpose set for the subsequent iteration, $M_{i+1} = \{x_{\sigma,i}^+ \cup M_i\}$. Consequently, for every iteration two new points are computed, one that is added to the training-only set at the location of the worst error and the other to the multipurpose set at the location of the maximum uncertainty. The former ensures areas very sensitive are finely sampled and the latter that undersampled areas do not remain unexplored (equivalent to the expected improvement function in the standard BO).

Finally, let us discuss a few of the decisions for the proposed approach. First, the advantage of having training-only set, Z_i , as opposed to basing the training data solely on the combinations of R_j . The rationale behind this is that if we were to rely solely on M_i without a fixed training set, we could encounter the situation where the error at one point in one of the sets j is orders of magnitude larger than in other sets. If that point were added then to M_{i+1} in the next iteration, the interpolations that do not include that point in their respective training sets R_i would see again that same error orders of magnitude larger than elsewhere in the parameter space. Thus, the resulting “worst case” condition for that iteration could be the same as in the previous one, rendering the iteration futile. Therefore, by having a training-only set, we ensure that points in the parameter space that cause large deviations in the interpolated versus true data are always considered, allowing subsequent iterations to continue sampling the parameter space at other locations for additional “worst case” points.

Another clarification would come from the need of having another point computed from the outcome of (10b) added to M_{i+1} . This point is necessary to make the optimal sampling scheme independent of the initial conditions, in this case the points X_0 , Z_0 and $M_0 = X_0 \setminus Z_0$. Effectively, by adding a new point to be used as training or testing in areas of the parameter space where there is large variance (uncertainty), permits the optimization to increase its knowledge on areas of potentially high error which otherwise could be ignored.

These sampling strategies will now be employed to test the interpolation framework to predict the flutter boundaries of the Pazy wing.

IV. Case Study: Pazy Wing

The Pazy wing will be the subject of this study as its design space has been previously analyzed and it was a computationally expensive task that could benefit from this scheme. The wing is a very flexible wing, designed and developed at the Technion as part of the 3rd Aeroelastic Prediction workshop, to be used as benchmark for numerical and experimental demonstrations [13] (Fig. 3). It is 550 mm in span and 100 mm in chord; it consists of an aluminum spar within a Nylon 3D printed chassis with a NACA0018 airfoil profile ribs to which an Oralight film skin gives its aerodynamic shape. A one-dimensional beam model of the wing has been derived by Riso et al. in [35] where a 3D finite element model has been reduced to sectional coefficients such that it can be used in tools such as SHARPy. The Pazy wing is capable of undergoing large deformations (approximately 50% of span) and in previous work by the authors, the wing has shown a complex stability envelope driven by its deformation [14]. The envelope is shown in Fig. 4, where it can be seen that the wing exhibits an initial instability characterized by a 1st flutter mode that becomes stable again as the deformation increases, leading to a region of stability before a 2nd flutter mode onset at higher speeds. This analysis was useful in designing a wind tunnel experiment that was safe and non-destructive, and for that the stability envelope was an important informative tool.

The drawback is that producing such envelope is a computationally expensive process. First, obtaining a linearized system of a highly discretized model (needed for flutter convergence) and subsequent model reduction to evaluate the eigenvalues is in itself costly, as previously described in Sec. II by steps (i)-(iii). Moreover, this system is influenced by many nonlinearities, with structural mode switches and large deformations which affect both the structural and aerodynamic subsystems, and requires of linearization and reduction at every angle of attack and free stream velocity pair. Thus, on a desktop system, the analysis shown in Fig. 4 in which the parameter space was sampled in 1 m/s increments (totaling 560 system evaluations) took approximately 60 hours to run.

Therefore, we turn to interpolation methods to reduce that time considerably, such that this information can be obtained in a quicker fashion or even for it to be used as part of a design space optimization routine. The objective is to produce a minimal library of training data that can be used as part of an interpolation scheme to sample the design parameter space efficiently and accurately. To build this minimal set we employ the methods described in the previous section, starting from simple, non-adaptive grid cases to the adaptive-grid cases which aim to produce an optimal training library.

One of the many benefits of interpolating the underlying dynamical systems using this framework, as opposed to interpolating the flutter speed directly, for example, is that the resulting state-space realizations can be used to compute loads or even for controller designs (such as flutter suppression systems).

Fig. 3 Pazy wing experimental model [13].

Fig. 4 Pazy wing stability envelope [14].

The parameter space is 2-dimensional, with angle of attack, α and free stream velocity, U_∞ as the independent variables and, attending to the definition in (3), can be described as

$$X = [\alpha = 2^\circ, \alpha = 5^\circ] \times [U_\infty = 10 \text{ m/s}, U_\infty = 70 \text{ m/s}] \in \mathbb{R}^2. \quad (11)$$

This corresponds to a subset of the design space shown in Fig. 4 that intends to capture the first flutter (T1 + OOP2) instability. This instability is a “humped” mode, whereby the eigenvalue stabilizes shortly after crossing the imaginary axes and is bound to be very sensitive to modeling uncertainty. We will explore this space to judge the feasibility of the interpolation methods.

A. Non-adaptive grids

A regular grid provides a simple example case from which to get a first glance about how a nonlinear system with several mode crossings can be captured by an interpolation based on a few sample points. We heuristically explore a few possibilities in the training data set for the interpolation, Z , and in this case for validation purposes X_0 is a large set of known true points, sampled every 1 m/s and 0.5 degrees angle of attack, totaling 215 known points. This enables a comparison in the flutter envelope between true and interpolated systems.

The first two cases use a structured, evenly distributed grid, being the only difference between the two the location of the training points at 40 m/s versus 50 m/s. These are shown, along with the true flutter envelope, in Fig. 5 as well as the location where the interpolated state-spaces have zero damping. This figure is a reparametrization of Fig. 4, where the abscissa now corresponds to angle of attack as opposed to wing tip deflection, and those are now shown as contours. The interpolated results are produced interpolating the reduced state-space matrices element-wise and can be computed anywhere in the parameter space. In this case, the interpolation points are equal to X_0 (described in the previous paragraph) in order to have a direct comparison. This computational cost of the interpolation is negligible, as more than 100 points can be interpolated in under a minute. As can be seen from the figures, the choice of sampling points has a very significant impact on the resulting flutter envelope. The case sampled at 40 m/s (Case 1 - Fig. 5a) offers a poorer interpolation result, in particular in the lower angle of attack region where the first instability is missed. Shifting the intermediate sample points to 50 m/s (Case 2 - Fig. 5b) changes completely the performance of the interpolation, with a very good representation of the flutter boundary at all angles of attack. Qualitatively, judging from the flutter envelope, the sensitivity to the sampling points is extremely high, and likely these cannot be chosen heuristically as the result could not correspond with the underlying true system.

The contours of span-normalized wing tip deflection shown in Fig. 5 (and subsequent figures), correspond always to nonlinear aeroelastic equilibrium of the true system, as the interpolation is always performed between linearized reduced systems about this equilibrium condition.

Fig. 5 Flutter boundary for two interpolation cases using regular grid sampling at different free stream velocities with contours of span-normalized wing tip displacement.

Another option for choosing a sampling set that is independent from user input is a Latin hypercube sampling, often used in design of experiments. However, it goes into the pitfall that sampling certain regions has a significant impact on the parameter space, thus the randomness has not offered acceptable results in predicting the flutter envelope of the wing and was discarded after an initial exploration. Therefore, the need for other methods that have information on of

the underlying physics and are capable of choosing the optimal location for the sampling points.

B. Adaptive and optimal grid sampling strategies

1. Optimal sampling using Bayesian Optimization

Following the results with non-adaptive grids, it was apparent that there could exist an optimal sampling of the parameter space. As discussed in Sec. III, a reasonable strategy would be to use a Bayesian optimizer to maximize the interpolation error function and add the point where the greatest error occurs to the training set, and repeat this iteration until a certain condition (either computational budget or error threshold) were satisfied.

The results of this implementation can be seen in Fig. 6. We performed an iteration with 5 points as budget, in addition to the initial conditions of the training set, Z_0 , which includes the 4 corner points of the parameter space. For each iteration, the optimizer was granted three evaluations of the cost function given by a Latin hypercube sampling to construct the Gaussian model which become the initial members of X_s in (8). Then, three additional points are computed (and added to X_s) which are now determined by the acquisition function (expected improvement). For subsequent iterations, i.e. after each Bayesian optimization that produces a new training point, the X_s set includes all previously computed X_s , updated with the latest training set Z_i . This process is computationally cheap (no further evaluations of the full system are required) and results in a significant larger dataset to construct the Gaussian statistical model.

Figure 6a shows that, with the additional 5 points sequentially computed after each optimization iteration, the interpolation framework presents an excellent agreement with the underlying, true, stability envelope. Fig. 6b shows how the optimizer has sampled the parameter space across all iterations to obtain the actual values of the error function at $\varepsilon[X_s; Z_5]$. In terms of the locations sampled, the optimizer has done a more refined sampling in velocity as opposed to that in angle of attack, focusing on the edges of the domain indistinctly. Finally, the drawback: for every training point that has been added, 6 evaluations of the true system have been performed. This equates to 30 new points that have been computed that have not made it into the optimal training dataset. In formal terms, following the optimal sampling strategy $|Z_5| = 9$ and $|X_5| = 30 + |Z_5| = 39$. It is this inefficiency that leads to the next sampling strategy.

(a) Stability boundary with contours of span-normalized wing tip displacement.

(b) Cost function evaluations during optimization

Fig. 6 Grid sampling using Bayesian optimization for the selection of next best sampling point.

2. Optimal sampling using a Bayesian approach with multipurpose sets

The cost of locating a new training data point via a standard direct BO process has presented itself as inefficient. Therefore, we apply the method explained in Sec. III.C.3 on the Pazy wing. To compare with the previous case, we seek to add the same number of points, 5, to the training only data set. In this case, we also need to initialize the multipurpose set, which has taken four points in the parameter space sampled using Latin hypercubes (although thanks to the corollary in (10b) the interpolation is independent of the actual choice). However, more important than the location of initial

points in M_0 is the number of points in the set, as this effectively is going to determine how many possible combinations between sets there are. Since Z_0 is initialized with the parameter edges, i.e. 4 points in 2D spaces, 5 initial points in M_0 is a very reasonable choice, since we will obtain combinations of 4 training / 5 testing, 6 training / 3 testing and 5 training / 4 testing, which will be all balanced between themselves allowing the aggregation of the different $\mu_{\varepsilon,j}$ without having any combination that is overly unbalanced and also having a decent number of combinations. For these particular initial conditions, Table 1 summarizes the number of sets for the first iteration.

Table 1 Number of sets and training/testing set split for the initial iteration.

Number of training points $ Z_0 \cup R_j $	Number of testing points $ S_j $	Number of sets
4	5	1
5	4	5
6	3	10
Total combinations, J		16

This scheme then, compared to the previous BO, just in terms of evaluations of the full system, already proves itself more efficient. Here, 10 initial evaluations are required, followed by 2 per iteration (one at each maximum in (10)) times 5 iterations (computational budget), totaling 20 evaluations, compared to the 39 in the prior example, practically a 50% reduction.

We can now illustrate the multipurpose set sampling more clearly. Once the sets are initialized we know Z_0 and j -combinations of S_j and R_j , with which we construct the statistical representation of $\mu_{\varepsilon,j}$ of each. Figure 7 shows the posterior mean and variance for two different j 's in the first iteration $i = 0$. Set 1 (Fig. 7a-7b) has 6 training points and 3 testing points which are marked with circles. From the posterior mean of the Gaussian estimation constructed using these testing points, the expected error is highest in the upper angle of attack and velocity region, where we know from the previous experiments that flutter and mode crossings occur. On the other hand, in the lower velocity end of the spectrum where the wing's dynamics change smoothly and the error in the interpolation scheme is low. Set 2, obtained by recombining the previous points (no new simulations), reinforces this hypothesis (Fig. 7c-7d). First note that the training and testing sets are different from Set 1: the point at $(4.5^\circ, 50 \text{ m/s})$ is now a training point and the one at $(3.0^\circ, 40 \text{ m/s})$ has become a testing point. Despite having a training point closer to the flutter region, the expected error is still high in the upper portion of the parameter space. Additionally, in this second set the lack of training points in the lower half has caused the expected error to grow, however, it is 50% of the maximum error in the upper half. From these two sets we can already infer that a much more refined sampling is going to be needed in the upper half where flutter occurs than in the lower one where the wing is stable.

The variance in both sets (Fig. 7b and Fig. 7d) sheds light on the regions where the Gaussian model has highest uncertainty. The initial points in M_0 were clearly biased towards the upper half of the parameter space, thus the variance is largest in the region below 30 m/s, and this is where we expect a new point to be added to the multipurpose set at the next iteration, M_1 .

The aggregated result, which determines where to sample next the parameter space, at the end of the first iteration, follows what is shown by the two sample sets. The optimum location after the first iteration is $x_{\mu,0}^+ = (5^\circ, 61 \text{ m/s})$ and $x_{\sigma,0}^+ = (2^\circ, 20 \text{ m/s})$, which will be added to the training-only set Z_1 and multipurpose set M_1 , respectively.

The remaining iterations take place in a similar fashion, with two new points computed at the end and added to the training-only and to the multipurpose set. The results of the average error across the different iterations

$$\mu_{\varepsilon} = \frac{1}{J} \sum_{j=1}^J \mu_{\varepsilon,j} \quad (12)$$

can be stacked to visually show the effect of adding more points as the iterations progress. This is illustrated in Fig. 8, where the expected average error (12) is plotted for each i -th iteration in Fig. 8a and compared to the actual underlying error, ε , in Fig. 8b. These figures shed significant light onto the performance of the adaptive Bayesian sampling. First, it can be seen how the magnitude of the error decreases as more training points are added in subsequent iterations. Additionally, the statistical modeling of the cost function by means of the Gaussian process is better suited to correctly estimate the behavior of large regions in the parameter space, as small local features may be filtered out. This is clear

(a) Posterior Mean of Set 1. Testing set colored according to value of actual error at those points.

(b) Variance of Set 1.

(c) Posterior Mean of Set 2. Testing set colored according to value of actual error at those points.

(d) Variance of Set 2.

Fig. 7 Mean and variance of the expected error for two different sets.

from Fig. 10, iteration $i = 4$, where the Gaussian model picks up the error across the parameter space at $U_\infty \approx 50$ m/s as a smooth undulation that is also visible in the true error function. However, small local areas of larger error like the one seen on the actual system at $(\alpha \approx 2^\circ, U_\infty \approx 50$ m/s) are mostly damped out in the statistical model. Notwithstanding, this is common and expected, and is akin to an aliasing issue in when working with discrete-time systems for instance, where features of frequency higher to the sampling rate may be totally missed.

The final test then corresponds to the actual flutter envelope comparison against the interpolation for cases with 5 and 10 iterations of the adaptive Bayesian sampling technique, which are shown in Fig. 9a and Fig. 9b, respectively. It can be seen that the adaptive sampling technique starts refining the sampling in the upper velocity region, where the flutter mode stabilizes, and the boundary is well captured at 5 iterations. On the other hand, the first case errs on the lower velocity flutter boundary, where the interpolated result is relatively constant with velocity (as opposed the the actual system where the flutter speed decreases with angle of attack). Letting the scheme continue to add points, it has added points strategically near $U_\infty \approx 40$ m/s and the flutter boundary is now well captured.

To show that not only the flutter speed is well captured but that also the dynamics of the system can be predicted across the parameter space, Fig. 10 shows velocity-damping-frequency plots at a constant 4.5° angle of attack for the two cases obtained after 5 and 10 iterations. Both sampling cases capture the major changing dynamics of the system, with

Fig. 8 Change in the expected and actual error as the adaptive sampling progresses.

Fig. 9 Flutter envelope of the Pazy wing using adaptive sampling after 5 and 10 iterations with contours of span-normalized wing tip displacement.

the interpolated systems observing the mode crossing at approximately 50 m/s leading to the “hump” mode instability (note positive damping ratio denotes an unstable system). Looking more closely at the differences between the two, the interpolation after 10 iterations (Fig. 10b) captures much better the peak in damping ratio, which in turn results in a more accurate flutter boundary definition. In the less refined case (Fig. 10a), the peak damping ratio is underestimated, leading to a shallower gradient and delayed prediction of the instability onset. Notwithstanding, both cases offer a realistic representation of the underlying system dynamics, which justifies the advantages of the interpolation being performed on the actual state-space realization as opposed to interpolating on the flutter speed only.

Comparing the performance of this sampling scheme against the standard Bayesian optimization shown in the previous subsection, a direct results comparison can be drawn between Fig. 6a and Fig. 9a, as they both have the same number of points. It is clear that the former, the BO case, is more accurate in the prediction of the flutter boundary, thus it could be said that the sampling points chosen by the BO are of greater importance in capturing the underlying dynamics. However, for this number of iterations, the BO scheme has had to sample the true system twice as many times. In comparison, in the Bayesian sampling with multipurpose sets, going from the situation in Fig. 9a to that in Fig. 9b

Fig. 10 Velocity-damping-frequency at 4.5 degrees angle of attack using adaptive sampling after 5 and 10 iterations.

“costs” 10 evaluations to the full system from which substantial accuracy is gained and in terms of total evaluations, it stands at 30, as opposed to the 40 needed to choose the standard BO scheme points in Fig. 6a.

Although the BO sampling has been more precise in the selection of the sampling points, it has done so at larger computational cost than the multipurpose sampling, which for a reduced number of evaluations has produced the same level of accuracy. Since the number of training points does not hinder the performance of the interpolation (aside from the need to compute them at some point), the newly proposed scheme can be considered more efficient in the task of seeking an optimal sampling set for the interpolation framework.

Hitherto, all simulations have been carried out with relatively coarse mesh of the Pazy wing when using SHARPy in order to speed up the testing process of this new methodology. The algorithm just presented is ready to be tested on the fully refined Pazy wing model in the next section.

3. Fully refined Pazy wing

The final test case then is on the Pazy wing with a refined mesh for flutter analysis. It is the same mesh employed in [14] to produce the wing’s stability envelope, a computationally intensive task. Therefore, with the proposed framework we aim to reduce the computational burden while obtaining accurate results. Whereas in the previous section only a subset of the parameter space in Fig. 4 was used, the exploration region is now expanded to include the whole design space

$$X = [\alpha = 0^\circ, \alpha = 5^\circ] \times [U_\infty = 10 \text{ m/s}, U_\infty = 90 \text{ m/s}] \in \mathbb{R}^2. \quad (13)$$

This problem immediately presents itself more challenging given the larger variations in the system underlying dynamics across the parameter space. For instance, attending to Fig. 4, it is known that in the lower angle of attack regions at high velocity multiple modes will be unstable, as opposed to at higher angles of attack where the two regions of instability are more separated.

This paper is focused on reducing the computational time in exploring a nonlinear aeroelastic system design space, thus, now that we focus on the full size system, it is reasonable to discuss the computational burden of an interpolation approach to computing the flutter envelope versus the approach employed in [14] which involved multiple evaluations across the whole space (each requiring the execution of steps (i)-(iii) in Sec. II). To recap on the sizes of the system, the converged Pazy wing model built in SHARPy has 960 bound vortex panels and 19200 wake vortex panels. The aerodynamic linearized state-space is then of size 22080 states, which following the Krylov-based reduction it is reduced to a stable realization of 48 states. The structural system consists of a 16 element (3-noded) beam which, after linearization, is truncated to retain the first 8 vibration modes (16 states). The coupling of both systems results in a reduced, linear aeroelastic state-space of 64 states in size, which is a very adequate size to perform operations such as interpolation in this work or eigenvalue analysis in the previous. However, the process of computing the nonlinear

equilibrium, linearization and reduction takes approximately 5-7 minutes on a quadcore Intel i7 4.2GHz CPU desktop workstation. The work in [14] which performed nearly 500 of these evaluations took approximately 60 hours to run. Note that some cases take longer to run than others based on how quickly the nonlinear static solution converges.

Now, as stated in previous sections, the objective is to reduce the number of these evaluations. Although the interpolation framework tests presented up to now have demonstrated such capabilities, the full size system introduces complexities in memory management that were not present with the smaller models or in the sequential evaluation of the parameter space. The greatest limitation identified thus far is in the number of points that may be used as training points, $|Z \cup R|$, since the reduced systems first need to be projected onto a compatible space (step (iv) in Sec. II) and this requires collecting their reduced order bases which are very “tall” matrices (the Krylov matrix is 22080×48 and the modal reduction matrix is 720×16) which need to be loaded in memory prior to the interpolation. The current implementation is performed on a workstation (64GB RAM) for which we have encountered a limit of 24 training points before running into memory issues. To circumvent this issue, we partition the parameter space into 4 separate quadrants on which the adaptive sampling based on multipurpose sets is performed independently and then the results are combined.

Table 2 Times to run processes on Intel i7 4.2GHz, 64GB RAM Workstation.

Single evaluation of the true system	7 min
Training statistical surrogates (computing true systems not included)	6 min / 15 sets
Interpolation of the parameter space (with 10 known training points)	1 min / 100 interpolation points

The multipurpose based adaptive sampling is performed in a similar manner in each quadrant. The quadrants are each initialized with a training set, Z_0 , containing the quadrant edge points and a first multipurpose set, M_0 , with 6 points chosen at random. The aggregate result is presented in Figure 11, which shows the flutter envelope derived from the Bayesian adaptive sampling method, where the actual areas of instability are shaded and contours of deflection are shown to emphasize the large variations in the wing shape. Recall, this figure is a reparametrization of Fig. 4, where the abscissa is now the angle of attack. In general, the interpolation results predict well the flutter envelope. The first flutter instability, the onset of the 1st torsion and 2nd out-of-plane bending flutter mode, is well approximated across all angles of attack. The stabilization of this mode is better captured at the higher angles of attack, where the next unstable feature is most separated. As hypothesized, the region where the two unstable modes converge is where it is most difficult to capture the crossing instability boundary and it is where we see most errors. Farther from this area, the 1st torsion and 1st out-of-plane bending unstable mode is generally well approximated too.

Fig. 11 Flutter envelope of the Pazy wing using adaptive sampling, with regions of instability shaded, contours indicate span-normalized wing tip displacement and the separate quadrants are outlined.

The quadrants employed to generate to the combined result are faintly outlined in Fig. 11 and described in Tab. 3;

they have been chosen to be different in size in order to explore whether smaller quadrants offered significantly better approximations. The quadrant size shouldn't affect the quality of the interpolation directly; it may require more points if it were larger, but the adaptive sampling technique should end up identifying the key points to sample. Due to the memory limitations previously mentioned, the interpolations are performed independently within each quadrant and the training points on the boundary belong to a given one, they are not shared among quadrants, which limits the computational efficiency of the approach. Table 3 also shows the training points used to obtain each interpolation. Two quadrants reach the memory limit, quadrant 3 captures in general both boundaries that fall within it whereas quadrant 1 sees greater deviations in the complex region at low angle of attack. The other quadrants perform reasonably well with less than half of the points, which makes for an interesting case.

Table 3 Parameter space sectorization.

Quadrant	Angle of attack range [deg]	Free-stream velocity range [m/s]	Total training points
1	0 - 2	10 - 90	24
2	2 - 3.5	10 - 90	10
3	3.5 - 5	50 - 90	24
4	3.5 - 5	10 - 50	10

In order to shed some light on the performance of the optimizer in the top-left quadrant, we define a new, small quadrant $[0.5^\circ, 2^\circ] \times [60 \text{ m/s}, 90 \text{ m/s}]$ to observe in more detail how the sampling process takes place. The actual cost functions are shown in Fig. 12a-c after 5, 10 and 15 iterations, respectively. It can be observed how the error, in particular around the edges, reduces as the iterations progress yet the large error region around $(1.5^\circ, 75 \text{ m/s})$ does not see any improvement with the addition of new points. In fact, what likely strikes out most is how the optimizer veers toward the parameter space edges, where the sampling becomes very refined. This behavior, that is present in all runs, is doomed to affect most the performance of the scheme in this region, as it is where the deflection changes as a function of angle of attack most abruptly (and hence the system dynamics). This can be seen from the contours of deflection in Fig. 11, which are closest together in this region as, for example at 80 m/s, the deflection increases by 30% between 0.5 and 2 degrees angle of attack compared to a mere 5% between 3.5 and 5 degrees angle of attack. This is why the performance in other quadrants is not compromised as much by the points steering towards the edges as in this case. In addition, this tendency is troublesome to the point that these points are the only source of information to train the statistical surrogate, thus, attending to the evolution of the predicted error in Fig. 12d, we can see how in the latter iterations the expected error tends to zero, as the model has only been trained by points very close to the edges where the interpolation captures with great accuracy the true system. In the expected error prediction, Fig. 12d, it is clear how the error regions are well estimated in the initial iteration but then decay rapidly. A first remedy would be to modify the acquisition functions in (10), yet we have found no improvement to those already listed in using alternatives like a weighted average between mean error and variance. Therefore, other options to try would be to use different Gaussian processes surrogates, such as those that include Markov-Chain Monte Carlo (MCMC) improvements, which could be explored in further work.

Having discussed the performance and limitations, we now turn to the computing times required for this framework, which is an important objective. The time to sequentially add 5 additional points to a given quadrant using the proposed method is approximately 200 minutes. This includes the 4 parameter edge evaluations, $|Z_0|$, the initial multipurpose set, $|M_0| = 6$, and the two new points added after each iteration (the maxima of (10)), resulting in 20 calls to SHARPy that account for 140 minutes approximately. The remaining tasks include those listed in Tab. 2, however, it is important to note that the current software implementation is at present not very efficient in as much as the systems are loaded (and their large reduced order bases) at the same time and this is performed numerous times, i.e. for each individual set of the interpolation. This is as a result of the developing nature of this work, where the interpolation framework can be rapidly adjusted from non-adaptive grids, to Bayesian optimization or to the adaptive multipurpose set-based Bayesian sampling, albeit at this computational overhead.

These results open several lines for future work. It is possible that different choices of the cost function yield performance improvements, although this can't be given. For instance, we experimented with a narrower damping penalty but didn't improve the results, as it meant that cases away from the flutter boundary became invisible and relying on the sampling points to fall close to the flutter boundary. Therefore, such an aggressive approach did not contribute to create an accurate surrogate of the cost function across the parameter space. However, as highlighted by

Fig. 12 Evolution of the actual interpolation error function on a small test quadrant, with training points highlighted.

the test quadrant in the upper left region, it is important to further investigate the role of the statistical surrogate or the acquisition functions in ensuring appropriate sampling of the parameter space.

V. Conclusions

Building on from the foundation work previously presented in [15], where we outlined a method to parametrically linearize, reduce and make compatible for interpolation aeroelastic models of very flexible aircraft, we have tackled the problem of sampling focusing on the Pazy wing flutter envelope case based on the interesting phenomena observed in [14]. The sampling methods shown here have demonstrated how important this step is in performing a meaningful interpolation across parametric reduced-order models.

The use of non-adaptive methods has fallen short given the sensitivity to the location of the training points and the underlying physical phenomena, which may leave the choice of these points solely on heuristics and needing to compute numerous true systems simply to evaluate the adequacy of the training points. In light of this, we have implemented adaptive methods to build optimal training sets which aim to enhance the interpolator's performance for the given application, in this case stability boundary, which is driven by an adequate error metric.

Since evaluating error metrics between interpolated and true systems is a costly process (a full evaluation using SHARPy is needed), statistical methods are useful to generate surrogate models to the cost function, which can be updated with a few selected testing points to produce an expected error and variance across the entire parameter space. Bayesian optimization is an obvious choice to test, since with a few evaluations of the error function can generate an estimate for the error that can be used to search for the maximum's location in the parameter space without the need for gradients. This method to maximize the error function and add those points to the training set has proven itself effective, where the resulting training set after several iterations of the BO process is more refined in areas where we know that the dynamics of the underlying system change most abruptly. The resulting flutter boundary is very representative of the actual one, just by adding a few points in key regions. However, it is still slightly computationally inefficient, as the optimizer routine requires of several evaluations of the true system in order to estimate the maximum leading to a high testing to training data ratio, which effectively means that numerous data points for which the true system is known are never used to improve the performance of the interpolation. Notwithstanding, the results obtained do highlight the benefits of adaptive sampling, as the training set produced captures with great accuracy the complex stability envelope of the Pazy wing.

Therefore, our novel method towards adaptive sampling builds upon the foundations of Bayesian optimization, as it employs Gaussian processes to estimate the error and variance of the cost function across the parameter space. However, to overcome the BO's issue in requiring numerous evaluations to produce a statistical model, we implemented the multi-purpose set strategy whereby, using different combinations of all the known data points, we could produce multiple estimations of the expected error that can be averaged to produce a closer representation of the underlying cost function. This approach, like the BO, has produced effective optimal training sets yet with a very significant improvement in the number of evaluations of the full system and the cost has been cut by 50% of that of the standard BO implementation.

The work performed on the refined mesh model of the Pazy wing across the entire parameter space has shown the computational advantages of the framework, at the same time as identifying lines of additional work. As mentioned in the previous section, the use of different acquisition functions or statistical surrogate models may be necessary to thoroughly explore the parameter space. Whilst different error metrics may be better suited for other problems, the one presented for the flutter boundary problem herein is considered satisfactory. Finally, a necessary improvement would also include a memory-efficient implementation of this framework in order to not be limited to the number of systems that can be loaded at a given time or to have to resort to partitions of the parameter space.

The overall result of the adaptive schemes is, in any case, successful to quickly predict system properties across a large parameter space. The interpolation and sampling schemes are promising yet but could benefit from further investigations. The impact of the chosen interpolation scheme, which currently uses standard-python libraries could be tailored and bespoke to parametric systems and may reduce the sensitivity to the sampling point selection. The next step would naturally be to apply this sampling and interpolation approach to explore the large dimensional parameter spaces of very flexible aircraft in order to predict, for example, dynamic loads efficiently across the flight envelope and introduce it as part of a design optimization or analysis loop.

References

- [1] Lomax, T. L., *Structural Loads Analysis for Commercial Transport Aircraft*, 1st ed., American Institute of Aeronautics and Astronautics, 1996.
- [2] Khodaparast, H. H., Georgiou, G., Cooper, J. E., Riccobene, L., Ricci, S., Vio, G. A., and Denner, P., "Efficient Worst Case "1-Cosine" Gust Loads Prediction," *ASDJournal*, Vol. 2, No. 3, 2012, pp. 33–54. <https://doi.org/10.3293/asdj.2012.17>.
- [3] Deskos, G., del Carre, A., and Palacios, R., "Assessment of low-altitude atmospheric turbulence models for aircraft aeroelasticity," *Journal of Fluids and Structures*, Vol. 95, 2020, p. 102981. <https://doi.org/10.1016/j.jfluidstructs.2020.102981>, URL <http://arxiv.org/abs/1908.00372>.
- [4] Dharmasaroja, A., Armstrong, C. G., Murphy, A., Robinson, T. T., McGuinness, S. H., Iorga, N. L., and Barron, J. R., "Load case characterization for the aircraft structural design process," *AIAA Journal*, Vol. 55, No. 8, 2017, pp. 2783–2792. <https://doi.org/10.2514/1.J055544>.
- [5] Jiang, Z., Li, W., Apley, D. W., and Chen, W., "A Spatial-Random-Process Based Multidisciplinary System Uncertainty Propagation Approach With Model Uncertainty," *Journal of Mechanical Design, Transactions of the ASME*, Vol. 137, No. 10, 2015, pp. 1–13. <https://doi.org/10.1115/1.4031096>.
- [6] Xie, J., Sarojini, D., Cai, Y., Corman, J. A., and Mavris, D. N., "Certification-Driven Platform for Multidisciplinary Design Space Exploration in Airframe Preliminary Design," *Journal of Aircraft*, , No. In advance, 2021. <https://doi.org/10.2514/1.c036328>.

- [7] Mavris, D. N., Bandte, O., and Delaurentis, D. A., "Robust design simulation: A probabilistic approach to multidisciplinary design," *Journal of Aircraft*, Vol. 36, No. 1, 1999, pp. 298–307. <https://doi.org/10.2514/2.2437>.
- [8] Chaudhuri, A., Lam, R., and Willcox, K., "Multifidelity uncertainty propagation via adaptive surrogates in coupled multidisciplinary systems," *AIAA Journal*, Vol. 56, No. 1, 2018, pp. 235–249. <https://doi.org/10.2514/1.J055678>.
- [9] Queipo, N. V., Haftka, R. T., Shyy, W., Goel, T., Vaidyanathan, R., and Kevin Tucker, P., "Surrogate-based analysis and optimization," *Progress in Aerospace Sciences*, Vol. 41, No. 1, 2005, pp. 1–28. <https://doi.org/10.1016/j.paerosci.2005.02.001>.
- [10] Villemonteix, J., Vazquez, E., and Walter, E., "An informational approach to the global optimization of expensive-to-evaluate functions," *Journal of Global Optimization*, Vol. 44, No. 4, 2009, pp. 509–534. <https://doi.org/10.1007/s10898-008-9354-2>.
- [11] Villanueva, D., and Smarslok, B. P., "Using expected information gain to design aerothermal model calibration experiments," *17th AIAA Non-Deterministic Approaches Conference*, 2015. <https://doi.org/10.2514/6.2015-0660>.
- [12] Huan, X., and Marzouk, Y. M., "Simulation-based optimal Bayesian experimental design for nonlinear systems," *Journal of Computational Physics*, Vol. 232, No. 1, 2013, pp. 288–317. <https://doi.org/10.1016/j.jcp.2012.08.013>.
- [13] Avin, O., Raveh, D. E., Drachinsky, A., and Ben-Shmuel, Y., "An Experimental Benchmark of a Very Flexible Wing," *AIAA SciTech Forum*, 2021. <https://doi.org/https://doi.org/10.2514/6.2021-1709>.
- [14] Goizueta, N., Drachinsky, A., Wynn, A., Raveh, D. E., and Palacios, R., "Flutter prediction for a very flexible wing wind tunnel test," *AIAA SciTech Forum*, 2021. <https://doi.org/10.2514/6.2021-1711>.
- [15] Goizueta, N., Wynn, A., and Palacios, R., "Parametric Krylov-based order reduction of aircraft aeroelastic models," *AIAA SciTech Forum*, 2021. <https://doi.org/10.2514/6.2021-1798>.
- [16] Geuss, M., Butnaru, D., Peherstorfer, B., Bungartz, H. J., and Lohmann, B., "Parametric model order reduction by sparse-grid-based interpolation on matrix manifolds for multidimensional parameter spaces," *European Control Conference*, EUCA, 2014, pp. 2727–2732. <https://doi.org/10.1109/ECC.2014.6862414>.
- [17] del Carre, A., Muñoz-Simón, A., Goizueta, N., and Palacios, R., "SHARPy : A dynamic aeroelastic simulation toolbox for very flexible aircraft and wind turbines," *Journal of Open Source Software*, Vol. 4, No. 44, 2019, p. 1885. <https://doi.org/10.21105/joss.01885>.
- [18] Del Carre, A., and Palacios, R., "Efficient Time-Domain Simulations in Nonlinear Aeroelasticity," *AIAA Scitech Forum*, 2019, pp. 1–20. <https://doi.org/10.2514/6.2019-2038>.
- [19] Maraniello, S., and Palacios, R., "State-Space Realizations and Internal Balancing in Potential-Flow Aerodynamics with Arbitrary Kinematics," *AIAA Journal*, Vol. 57, No. 6, 2019, pp. 1–14. <https://doi.org/10.2514/1.J058153>.
- [20] Geradin, M., and Cardona, A., *Flexible multibody dynamics: a finite element approach*, John Wiley and Sons, Chichester, 2001.
- [21] Simpson, R. J. S., and Palacios, R., "Numerical aspects of nonlinear flexible aircraft flight dynamics modeling," *54th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, 2013. <https://doi.org/10.2514/6.2013-1634>.
- [22] Hesse, H., "Consistent Aeroelastic Linearisation and Reduced-Order Modelling in the Dynamics of Manoeuvring Flexible Aircraft," Ph.D. thesis, Imperial College London, 2013. URL <http://hdl.handle.net/10044/1/12258>.
- [23] del Carre, A., Teixeira, P., Palacios, R., and Cesnik, C. E., "Nonlinear response of a very flexible aircraft under lateral gust," *International Forum on Aeroelasticity and Structural Dynamics 2019, IFASD 2019*, 2019, pp. 1–27.
- [24] Katz, J., and Plotkin, A., "Unsteady Incompressible Potential Flow," *Low Speed Aerodynamics*, Cambridge University Press, 2001, Chap. 13, 2nd ed., pp. 369–433. <https://doi.org/10.1017/CBO9780511810329>.
- [25] Murua, J., Palacios, R., and Graham, J. M. R., "Applications of the unsteady vortex-lattice method in aircraft aeroelasticity and flight dynamics," *Progress in Aerospace Sciences*, Vol. 55, 2012, pp. 46–72. <https://doi.org/10.1016/j.paerosci.2012.06.001>.
- [26] Simpson, R. J. S., Palacios, R., and Murua, J., "Induced-Drag Calculations in the Unsteady Vortex Lattice Method," *AIAA Journal*, Vol. 51, No. 7, 2013, pp. 1775–1779. <https://doi.org/10.2514/1.j052136>.
- [27] Lambert, T., and Dimitriadis, G., "Induced Drag Calculations with the Unsteady Vortex Lattice Method for Cambered Wings," *AIAA Journal*, Vol. 55, No. 2, 2016, pp. 668–672. <https://doi.org/10.2514/1.j055135>.

- [28] Maraniello, S., and Palacios, R., “Parametric Reduced-Order Modeling of the Unsteady Vortex-Lattice Method,” *AIAA Journal*, Vol. 58, No. 5, 2020, pp. 2206–2220. <https://doi.org/10.2514/1.j058894>.
- [29] Antoulas, A. C., *Approximation of Large-Scale Dynamical Systems*, 1st ed., SIAM - Advances in Design and Control, 2005.
- [30] Farin, G., “Triangular Bernstein-Bézier patches,” *Computer Aided Geometric Design*, Vol. 3, No. 2, 1986, pp. 83–127. [https://doi.org/10.1016/0167-8396\(86\)90016-6](https://doi.org/10.1016/0167-8396(86)90016-6).
- [31] Mockus, J., *Bayesian Approach to Global Optimization*, 1st ed., Springer, Dordrecht, 1989.
- [32] del Carre, A., and Palacios, R., “Simulation and Optimization of Takeoff Maneuvers of Very Flexible Aircraft,” *Journal of Aircraft*, Vol. 57, No. 6, 2020, pp. 1097–1110. <https://doi.org/10.2514/1.C035901>.
- [33] Snoek, J., Larochelle, H., and Adams, R. P., “Practical Bayesian Optimization of Machine Learning Algorithms,” *Advances in Neural Information Processing Systems*, Vol. 25, No. December, 2012, p. 2951–2959.
- [34] The GPyOpt Authors, “GPyOpt: A Bayesian Optimization framework in Python,” <http://github.com/SheffieldML/GPyOpt>, 2016.
- [35] Riso, C., and Cesnik, C. E., “Correlations Between UM/NAST Nonlinear Aeroelastic Simulations and the Pre-Pazy Wing Experiment,” *AIAA Scitech 2021 Forum*, 2021. <https://doi.org/10.2514/6.2021-1712>.