End-to-End Differentiable Aircraft Loads at Large Scale via Concurrent Simulations

Alvaro Cea*, Lucian Iorga†, Rafael Palacios‡

Abstract

This paper demonstrates a new solution for large scale simulations of aircraft-sizing loads. A geometrically nonlinear aeroelastic description has been extended with concurrent capabilities to tackle efficiently the very demanding loading scenarios in the design process. The software can be deployed on modern hardware architectures and has been benchmarked on CPUs and GPUs. While keeping computational times in the order of seconds, the presented approach enables the construction of load envelopes corresponding to static and dynamic cases, seamless integration with full-aircraft models, and uncertainty quantification analysis via massively-parallel Monte-carlo simulations, among others. Furthermore, differentiation of both concurrent computations and aeroelastic loads is achieved via JAX library automatic differentiation engine and collective primitives. We explore a high-aspect-ratio-wing aircraft configuration, showing the ability of the solvers to capture nonlinear effects in a wide range of scenarios: discrete, manoeuvre and gust loads are assessed for multiple flow and parametric conditions running concurrently. Sensitivities of maximum wing-root stresses and tip displacements are computed and verified against finite differences.

1 Introduction

Aeroelastic analysis is expected to become increasingly important from the very early phases of the wing design process: while the focus was historically on the post-design stages to ensure in-flight integrity, it now becomes paramount to capture the cross-couplings between disciplines for optimal performance. This is specially important for high-aspect-ratio wings (HARWs) [1, 2] that improve aerodynamic efficiency but leads to an increase in flexibility, which brings new challenges both in terms of design and modelling capabilities. As highlighted in [3], formulations that include nonlinear effects should be developed that not only enhance current modelling techniques but that also allow rapid data turnaround for the industry. Real-time, hardware-in-the-loop flight simulators would also benefit of actively controlled, deformable airplane models. This leads to a more nonlinear

*Research Associate, CAGB 308, South Kensington Campus. (alvaro.cea-esteban15@imperial.ac.uk)

†Wing Airframe Integrator, Airbus Operations Ltd., Filton, BS99 7AR, United Kingdom

‡Professor in Computational Aeroelasticity, CAGB 310, South Kensington Campus. AIAA Associate Fellow (r.palacios@imperial.ac.uk)

landscape, where the overall aerodynamic performance needs to be calculated around a flight shape with large deformations [4]. A more holistic approach to the design also increases the complexity of the processes exponentially, and the trade-offs and cost-benefit analysis may not be possible until robust computational tools are in-place to simulate the different assumptions. Certification of new air vehicles is another important aspect that requires 100,000s of load cases simulations [5], as it considers manoeuvres and gust loads at different velocities and altitudes, and for a range of mass cases and configurations. This poses another challenge for new methods that aim to include new physics since they normally incur in prohibitively expensive computational times. Lastly, the mathematical representation of the airframe, embodied in the complex Finite-Element Models (FEMs) built by organizations, encompasses a level of knowledge that is to be preserved when including the new physics mentioned above [6].

Leveraging on the numerical library JAX [7], these considerations set the goals for the baseline work in [8]: 1) to be able to perform geometrically nonlinear aeroelastic analysis, 2) to work with existing generic FEMs in a non-intrusive manner, and 3) to achieve a computational efficiency that is equivalent to present linear methods (if not faster), 4) compute derivatives of the aeroelastic response via Algorithm Differentiation (AD). Those were extended in to include rigid-body dynamics, trimmed flight, and architectural benchmarks of GPU versus CPU [9]. In this work we explore the latest advances on accelerator's parallelisation, and how to integrate them into our solution process to enable intensive aeroelastic simulations under geometrically nonlinear assumptions. Specifically, we set out to characterise the dynamics of highly flexible aircraft in response to the large envelope of simulations required to capture in-flight loads encountered in the certification process –while introducing new physics that account for the large displacements and rotations ultra-high-aspect-ratio wings are expected to undergo. For this, a Single Program Multiple Data (SPMD) paradigm is employed with the main computation spanning as many devices as available in the cluster and performing collective operations to communicate between devices. By addressing in one program a substantial part of scenarios during flight (manoeuvres and gust responses at different velocities and altitudes, and for a range of mass cases and configurations), we are able to produce the critical loading characteristics of the aircraft in very short simulation times. Moreover, we can differentiate the boundaries of the critical cases using the already demonstrated capabilities of AD within JAX [8], but now extending them to produce derivatives across those design envelopes. This implies computing gradients across concurrent simulations and collective operations, which we show it is well managed by the chosen library, that is actively developed to solved similar problems in the realm of machine learning. This is expected to be highly applicable in providing designers with additional insights about sensitivities and in extending gradient-based optimization analysis [10] with load-sizing constraints. In fact, having shown our formulation works well with high fidelity structural models of industrial complexity [11], a further step is to increase the accuracy of the currently used Doublet-Lattice-Method (DLM) solution. By coupling with differentiable Computational-Fluid-Dynamics (CFD) solvers such as those of [12], [13], [14], a fully-differentiated high-fidelity aeroelastic solver could be achieved. This is a very feasible option within our workflow where the matrices of Generalised Aerodynamic Forces

(GAFs) from a DLM solution are replaced by CFD GAFs, which are a standard part of the process in flutter analysis [15, 16].
Another area we begin to explore herein is the development of tools for optimisation under uncertainty. While various methods have been envisioned to introduce Uncertainty Quantification (UQ) in simulations [17], Montecarlo methods remain a compelling option for generic problems. They can suffer from slow convergence, however they cater well for embarrassingly parallel algorithms as each path in the simulation is independent of the others. The accelerated simulations may open new ways for UQ of aeroelastic problems without resorting to surrogates [18]. Therefore the three major contributions of this work are summarized are as follows:

- Large-scale parallelisation of static and dynamic aeroelastic cases to produce the load envelopes used in aircraft-sizing design loops.
- Computation of gradients via AD across the critical cases previously calculated concurrently.
- Differentiation of any moment of a distribution (expectations of the powers of a random variable) produced as the output of an aeroelastic quantity of interest to an stochastic input.

It is also worth remarking the whole suite of capabilities outlined are entirely physics-based, thereby not relying on metamodels of any kind, but achieving similar computational performance during evaluation thanks to a physical condensation of the model and a very efficient implementation.
The paper is organised as follows: Sec. 2 gives and overview of the theoretical and computational developments that underpin this work with a focus on the new parallelisation capabilities, computation of derivatives across concurrent simulations, and a new pattern to build sensitivities of large load envelopes. In sec. 3, a representative configuration of an ultra-high-aspect-ratio aircraft is studied under various loading scenarios that have been parallelised; namely structural static loads, manoeuvre cases for varying flow conditions and dynamic loads with multiple gusts running concurrently. To our knowledge, this application of modern hardware architectures to aircraft nonlinear load analysis has not been explored so far and has a huge potential to be introduced inside current industrial processes. We conclude in Sec. 4 with a summary of the main advances and potential future work to consolidate the methods into a versatile, production-ready tool.

2 Theoretical and computational background

The main aspects of the aeroelastic framework we have developed are presented in this section. The approach is built on a non-intrusive reduction order process combined with a nonlinear description of the dominant dimension for slender structures. It achieves a nonlinear representation of aeroelastic models of arbitrary complexity in a very efficient manner and without losing the characteristics of the linear model. We target the calculation of flight loads in this work, but it can also be applied to the computation of aeroelastic stability phenomena such as flutter or divergence [11].The key features of the formulation have been presented in previous work:

- Geometrically nonlinear aeroelastic analysis using complex GFEMs: achieved via a three step process in which a condensed model is first produced, the dynamics of this reduced model are described by a system on nonlinear equations [19] written in material velocities and stresses, and a modal expansion of those variables is the final key step in seamlessly mapping the global FEM into the nonlinear description [20]. The overall process can be found in [21].
- Maximum performance: as a combination of the numerical library JAX with its JIT compiler and accelerator capabilities driving the calculations, a highly optimised and vectorized codebase with the main algorithms described in [9], and the newly added parallelisation capabilities.
- Differentiation and sensitivity analysis: using JAX algorithmic differentiation toolbox, the entire process, from inputs to aeroelastic outputs can be differentiated as shown in [22].

In addition to this, we show below how to leverage on modern hardware architectures and a parallelisation across devices to a) build load envelopes of static and dynamic aeroelastic simulations in seconds; b) differentiate across the concurrent simulations to obtain sensitivities of dynamic loads; c) quantify the uncertainties in the nonlinear response given a non-deterministic loading field and obtain derivatives of the expectations.

2.1 Nonlinear aeroelastic system

Given a general GFEM, a reduced model is obtained from a static or dynamic condensation into nodes along load paths that captures well the stiffness and inertia properties in the condensed matrices, $\boldsymbol{K}_a$ and $\boldsymbol{M}_a$. The eigenvalue solution of the FEM yields the modal shapes, $\boldsymbol{\Phi}_0$, and frequencies $\boldsymbol{\omega}$. A projection of the state variables, velocities $\boldsymbol{x}_1 = \boldsymbol{\Phi}_1 \boldsymbol{q}_1$ and stresses $\boldsymbol{x}_2 = \boldsymbol{\Phi}_2 \boldsymbol{q}_2$, and a Galerkin projection of the equations of motion leads to the system of ODEs that is solved in time domain. Aerodynamic forces are obtained via Generalised Aerodynamic Forces (GAFs) using a panel-based DLM solver and Roger's rational function approximation [23] to bring the forces to the time domain, resulting in a modal force component given as:

$$\boldsymbol{\eta}_a = Q_\infty \left(\boldsymbol{\mathcal{A}}_0 \boldsymbol{q}_0 + b\boldsymbol{\mathcal{A}}_1 \boldsymbol{q}_1 + b^2 \boldsymbol{\mathcal{A}}_2 \dot{\boldsymbol{q}}_1 + \boldsymbol{\mathcal{A}}_{g0} \boldsymbol{v}_g + b\boldsymbol{\mathcal{A}}_{g1} \dot{\boldsymbol{v}}_g + b^2 \boldsymbol{\mathcal{A}}_{g2} \ddot{\boldsymbol{v}}_g + \sum_{p=1}^{N_p} \boldsymbol{\lambda}_p \right) \tag{1}$$

where the $\boldsymbol{\mathcal{A}}_i s$ are real matrices, $b = \frac{c}{2U_\infty}$ with c the reference chord, $Q_\infty = \frac{1}{2}\rho_\infty U_\infty^2$ the dynamic pressure, $\boldsymbol{\lambda}_p$ the aerodynamic states and N_p the number of lags. Note these forces naturally follow the structure since the formulation is written in the material frame of reference. The coupling of the structure and aerodynamic equations combined with the aerodynamic lags, gravity forces, $\boldsymbol{\eta}_g$,

and gust disturbances, $\boldsymbol{v}_g$, gives the final ODE system in compact form:

$$\begin{aligned}
\dot{\boldsymbol{q}}_1 &= \hat{\boldsymbol{\Omega}}\boldsymbol{q}_2 - \hat{\boldsymbol{\Gamma}}_1 : (\boldsymbol{q}_1 \otimes \boldsymbol{q}_1) - \hat{\boldsymbol{\Gamma}}_2 : (\boldsymbol{q}_2 \otimes \boldsymbol{q}_2) + \hat{\boldsymbol{\eta}} \\
\dot{\boldsymbol{q}}_2 &= -\boldsymbol{\omega} \odot \boldsymbol{q}_1 + \boldsymbol{\Gamma}_2^{\top} : (\boldsymbol{q}_2 \otimes \boldsymbol{q}_1) \\
\dot{\boldsymbol{\lambda}}_p &= Q_\infty \boldsymbol{\mathcal{A}}_{p+2}\boldsymbol{q}_1 + Q_\infty \boldsymbol{\mathcal{A}}_{p+2}\dot{\boldsymbol{v}}_g - \frac{\gamma_p}{b}\boldsymbol{\lambda}_p
\end{aligned} \tag{2}$$

where $\odot$ is the Hadamard product (element-wise multiplication), $\otimes$ is the tensor product operation and $:$ is the double dot product. In this system the aerodynamic added-mass effect has been moved to the left hand side such that $\mathbf{A}_2 = (\boldsymbol{I} - \frac{\rho c^2}{8}\boldsymbol{\mathcal{A}}_2)^{-1}$, and it couples all DoF in $\boldsymbol{q}_1$. Thus the natural frequency terms become $\hat{\boldsymbol{\Omega}} = \mathbf{A}_2\text{diag}(\boldsymbol{\omega})$ and the nonlinear terms $\hat{\boldsymbol{\Gamma}} = \mathbf{A}_2\boldsymbol{\Gamma}$. The effect of all external forces, aero, $\boldsymbol{\eta}_a$, gravity, $\boldsymbol{\eta}_g$, and others, $\boldsymbol{\eta}_f$, are combined in such that $\hat{\boldsymbol{\eta}} = \mathbf{A}_2\left(\left(\boldsymbol{\eta}_a - \frac{\rho c^2}{8}\boldsymbol{\mathcal{A}}_2\dot{\boldsymbol{q}}_1\right) + \boldsymbol{\eta}_g + \boldsymbol{\eta}_f\right)$. The aerodynamic matrices $\hat{\boldsymbol{\mathcal{A}}}_{p+2}$ have also been scaled accordingly. The nonlinearities in the system are encapsulated in the modal couplings of the third-order tensors $\boldsymbol{\Gamma}_1$ and $\boldsymbol{\Gamma}_2$ (the former introduces the gyroscopic terms in the dynamics and the latter introduces the strain-force nonlinear relation). Once the nonlinear solution of the condensed model is computed, the corresponding full 3D state can be calculated as a postprocessing step if necessary [8]. Note unsteady aero is linear. Here it is obtained about undeformed geometry since we use the DLM, but it could be updated to the trim shape if needed (particularly to CFD aerodynamics).

2.2 High performance implementation

The formulation described above has been made into the codebase FENIAX (Finite Element models for Nonlinear Intrinsic Aeroelastics in JAX) [1]. It has been thoroughly tested with currently 12 different models that amount to over 200 tests that run in minutes and are part of Continuous-Integration/Development (CI/CD) workflow. Moreover, a flexible software architecture allows for automatic analysis of generic models from standard input files, which can integrated with other computational tools. The Python library JAX has been used as the numerical engine for calculations and it also manages the parallelisation, therefore some details on the library are worth describing. JAX is designed for high-performance numerical computing with focus on machine learning activities [7]. It relies on XLA (Accelerated Linear Algebra), a domain-specific compiler for linear algebra that optimizes computations for both CPUs and GPUs. In fact XLA is platform-agnostic and achieves optimised performance on the target architecture orchestrating a complex process that encompassing a series of optimizations and transformations: the source code is first converted into HLO (High-Level Optimizer) code, an specialized language derived from a graph representation of the computations; XLA performs optimisations on the HLO code (geared towards high-level mathematical operations, particularly those in linear algebra and machine learning models), and are independent of the hardware architecture, such as operation fusion. It then carries optimisations for the particular architecture in use. From there the LLVM toolkit is leveraged to produce and Intermediate Representation (IR) that the LLVM compiler can understand, perform further optimi-

[1] Both implementation and examples can be found at `https://github.com/ACea15/FENIAX`.

sations and finally output the machine code. When it comes to leveraging the computational power of (NVIDIA) GPUs, the link between XLA and CUDA kernels is critical. On the one hand, JAX utilises CUDA libraries such as cuBLAS for dense linear algebra; on the other hand, it is capable of generating custom CUDA kernels for operations that are not efficiently covered by standard libraries. In order to transform the high level Python to low level optimised code, the source code has to comply with various constraints and feature functional programming characteristics. With regards to the parallelisation, JAX follows a Single-Program Multi-Data (SPMD) parallelism, whereby a single program operates on multiple data sets in parallel. This means the same computation graph is compiled and executed across different devices. Inter-device communication and synchronization are managed internally by the library. Implementation wise, the `pmap` function maps a function across multiple input sets, distributing the workload across available GPUs. Thus being the parallel equivalent to the `vmap` function. The new standard for parallelisation is based on data sharding, either done automatically using the `shard_map` function or by sharding the data and passing it to a `jitted` function specifying input and output shape of the data to be partitioned. Inside the function, the compiler determines the necessary partitions of the data, synchronization, and communication. Collective operations like broadcasts and reductions are available within the `jax.lax` module. Internally JAX uses NVIDIA Collective Communications Library (NCCL) for low level communication across devices. The overall solution process and a description of the parallelisation strategy follow next.

2.2.1 Overall solution process

Algorithm 1 shows the main components in the solution, highlighting the time and space complexities, $O(time, space)$, of the data generated along the process. One single program is being run, for instance a dynamic simulation computing the response to multiple gusts that will be run in parallel for a total number of N_c cases. N_t time-steps are used in the integration scheme with a resolution of N_m modal shapes. The FE model has been condensed to N_n number of nodes.
The intrinsic modes, $\boldsymbol{\phi}$, $\boldsymbol{\psi}$, are computed from the condensed FE nodal positions and matrices; subsequently, the nonlinear terms, $\boldsymbol{\Gamma}$, are obtained as the integral along the reduced domain of the modal couplings; the nonlinear system of equations is built and time-marched in time to yield the solution in modal coordinates, $\boldsymbol{q}$; the intrinsic variables of the solution (velocities, $\boldsymbol{X}_1$, internal forces, $\boldsymbol{X}_2$ and strains, $\boldsymbol{X}_3$) are recovered from the modal coordinates and the intrinsic modes; finally the positional and rotational field, $\boldsymbol{r}_a$, $\boldsymbol{R}_a$, of the reduced model are computed via integration of the strain field.

2.2.2 Two-level parallelisation

Various parallelism models have been developed in the context of deep learning, for which JAX has been particularly designed, and we try to adapt here those methods to a large system of nonlinear equations solved in parallel for multiple external forces, i.e. right hand side of the equations. Data Parallel (DP) strategies consist of splitting a large batching into chunks, each fed to a single

Algorithm 1: Main components in solution process

input : Input file: settings.yaml; FE model: $\boldsymbol{K}_a$, $\boldsymbol{M}_a$, $\boldsymbol{X}_a$; Aerodynamic matrices: $\boldsymbol{\mathcal{A}}$
output: Nonlinear aeroealastic solutioxn
begin

$\boldsymbol{\phi}$, $\boldsymbol{\psi} \longleftarrow$ modes($\boldsymbol{K}_a$, $\boldsymbol{M}_a$, $\boldsymbol{X}_a$) ▷ Intrinsic modes: O($N_n^2 \times N_m$; $N_n \times N_m$)
$\boldsymbol{\Gamma} \longleftarrow$ couplings($\boldsymbol{\phi}$, $\boldsymbol{\psi}$) ▷ Nonlinear couplings O($N_n \times N_m^3$; N_m^3)
$\boldsymbol{q} \longleftarrow$ system($\boldsymbol{\Gamma}$, $\boldsymbol{\mathcal{A}}$, $\boldsymbol{\phi}$, $\boldsymbol{X}_a$) ▷ Modal coordinates: O($\frac{N_c}{N_d} \times N_t \times N_m^3$; $N_c \times N_t \times N_m$)
$\boldsymbol{X}_1$, $\boldsymbol{X}_2$, $\boldsymbol{X}_3 \longleftarrow$ ivars($\boldsymbol{q}$, $\boldsymbol{\phi}$, $\boldsymbol{\psi}$) ▷ velocity/strain fields: O($\frac{N_c}{N_d} \times N_t \times N_n \times N_m$; $N_c \times N_t \times N_n$)
$\boldsymbol{r}_a$, $\boldsymbol{R}_a \longleftarrow$ integration($\boldsymbol{X}_3$, $\boldsymbol{X}_a$) ▷ Positional/rotational fields: O($\frac{N_c}{N_d} \times N_t \times N_n \times N_m$; $N_c \times N_t \times N_n$)

device, hence scaling to very large data batches. In Large Language Models (LLMs), the number of parameters can exceed that of input data, and often do not fit in a single device. In this case a tensor parallelism (TP) strategy is employed by which the tensor of weights that are to be optimised is sharded with synchronisation at the end of each step. Hybrid strategies that mix the two are employed in production. In the engineering applications of interest here, the number of designs variables would usually be between the tens to the few thousands, so tensor parallelism becomes less relevant. However, the number of simulations for different inputs, and the size of each one of them, can be very large. Therefore we opt for a DP strategy in which our batch of data becomes the multiple inputs that are used to build the external forces for which we want to compute the response. The strategy implemented first splits the input data along the leading axis according to the total number of devices available using a data sharding approach. Each device receives the subset of inputs, a closure function that is "jitted" is called with the respective inputs, and inside the closure the high level function that computes the response (solution of the static response or time marching of the dynamic equations) is "vmapped" with respect to the subset of inputs. This last `vmap` makes the inputs that go into each device, or CPU cores, to run in parallel. Note the parallelisation happens at the final solution of the system of equations, meaning previous steps such as computation of intrinsic modes or nonlinear couplings is only carried out once before the concurrent simulations. Algorithm 2 illustrates this process with pseudo code. The process by which inputs are split and sent to each device is presented in Fig. 1, which shows the two-level parallelisation.

The inputs to the parallel simulation are tensors of arbitrary shape with the only condition that the first axis being the one over which to run the parallelisation. For the manoeuvre and gust cases below, for instance, the tensor of inputs is a matrix with the second axis being a vector with the combination of flow conditions and gust parameters. In the figure we can see each GPU has a global memory and L2 cache, and in addition cores in the GPU are packed into the so-called streaming processors, each with its own registers and L1 caches. The strength of these chips is in the large number of cores, in the thousands, that can run in parallel, thus after the inputs are initially divided,

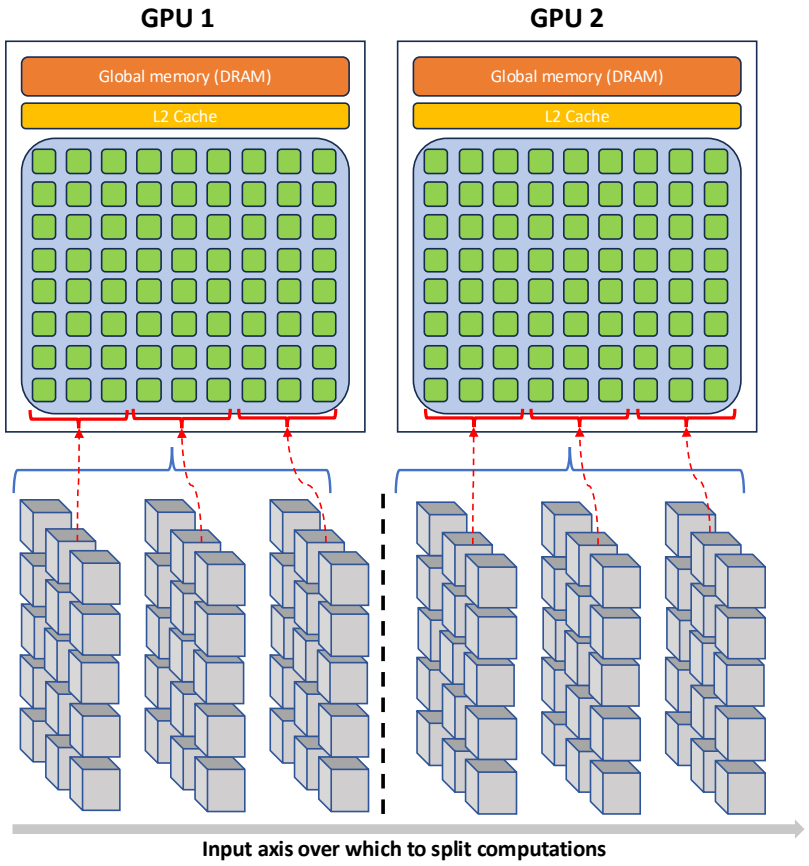

Figure 1: Input distribution example for multi-GPU runs

many computations can run in parallel even within each GPU.

Algorithm 2 has been also implemented using a `pmap` function with similar results but slightly different design.

2.3 Gradients of concurrent load-cases

Differentiable Programming encompasses AD, adjoint solutions and various concepts to produce full-fledged programs where derivatives can be taken through complex control flow, loops, and even other libraries embedded in the program such as MPI. It is the backbone of both gradient-based optimisation and Machine Learning frameworks, hence a very active field of research. New libraries for AD are either based on source-code transformation [24] or tape-based methods [25]. JAX tape-based approach showcases unique features to do AD in heterogeneous devices and across concurrent operations that are explored herein with two important applications: obtaining derivatives of statistical moments of a distribution generated as the output of a Montecarlo simulation, and a strategy to differentiate the boundaries of load envelopes representing critical aircraft loads in the certification process. The JAX-based numerical library Diffrax [26] is used as it ships with robust Newton-Raphson and ODE solvers with various adjoints options. It has been found that when computing derivatives of parallel simulations, the solvers in this library are not compatible with

Algorithm 2: Parallelisation multiple load cases

```
begin
    Function y_aeroelastic(inputs):
        Function y(input):
            ...
            (nonlinear aeroelastic computation)
            return q, X1, X2, X3, ra, Rab
        yvmap = jax.vmap(y)
        q_multi, X1_multi, X2_multi, X3_multi, ra_multi, Rab_multi ⟵
        yvmap(inputs)
        return dict(q=q_multi, X1=X1_multi, X2=X2_multi, X3=X3_multi,
        ra=ra_multi, Rab=Rab_multi)
    num_devices ⟵ jax.device_count()
    mesh ⟵ jax.sharding.Mesh(
    devices=jax.experimental.mesh_utils.create_device_mesh(
    (num_devices,)), axis_names=('x'))
    inputs = jax.device_put(inputs, jax.sharding.NamedSharding(mesh,
    jax.sharding.PartitionSpec('x')))
    y_aeroelastic ⟵ jax.jit(y_aeroelastic)
    sol ⟵ y_aeroelastic(inputs)
```

the new sharding features in JAX, however, it is able to differentiate across concurrent simulations if those we produced via a `pmap` strategy. This means that `pmap` was used for the parallelisation in the differentiation of Expectations in Sec 3.2. Most likely both strategies will be supported for differentiation in the future, therefore giving more flexibility and aligning the AD to the various parallelisation approaches.

2.3.1 Montecarlo analysis for UQ

Montecarlo analysis utilises random sampling to evaluate the effect of input uncertainties on model outputs. It allows here to propagate uncertainties in the nonlinear aerolelastic formulation outlined in Sec. 2.1. Since the samples are independent, a very simple parallelisation is possible on this algorithm: for each sample load case (right-hand side of the equations), launch a simulation, then collect outputs and perform a collective operation such as the mean across devices to calculate expectations. We can represent the system of equations in Eq. (1) as follows,

$$\dot{\boldsymbol{q}} = \boldsymbol{Q}(\boldsymbol{q}, \boldsymbol{\alpha}_1) + \boldsymbol{L}(\boldsymbol{X}, \boldsymbol{\alpha}_2) \tag{3}$$

where α_1 and α_2 are the input parameters we want to obtain the gradients with respect to, the former for internal states in $\boldsymbol{Q}$, the latter for external loads in $\boldsymbol{L}$. $\boldsymbol{X}$ is the random variable or vector of variables to sample from. In Sec. 3.2 this will be demonstrated for a nonlinear static equilibrium with $\boldsymbol{\alpha}_1$ representing the FE matrices and corresponding eigenvectors, and $\boldsymbol{\alpha}_2$ a single

parameter controlling the amount of external loading in the structure. After marching the system in time or solving the nonlinear equations for a static problem, the random variable $\boldsymbol{Y}$ emerges as some (unknown) function $\boldsymbol{F}$ of the final state $\boldsymbol{q}(t_f)$:

$$\boldsymbol{Y} = \boldsymbol{F}(\boldsymbol{q}(t_f), \boldsymbol{X}, \boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2) \tag{4}$$

The objective is to calculate derivatives of arbitrary moments n of $\boldsymbol{Y}$, $\frac{\partial \mathbb{E}[Y^n]}{\partial \boldsymbol{\alpha}}$. They can be calculated via Montecarlo by launching the concurrent simulations for each $\boldsymbol{X}_i$ and performing a collective mean on the output of interest $\boldsymbol{Y}_i$ after solving the aeroelastic system. Since all the operations in the process are being tracked in JAX, derivatives with respect to $\boldsymbol{\alpha}$ can be recovered. The slow convergence is one of the key limitations of plain Monte Carlo methods as the error goes with number of samples N: Error $\sim O\left(\frac{1}{\sqrt{N}}\right)$. Quasi Montecarlo methods using low discrepancy sequences such as Sobol numbers can be used for better convergence and multilevel methods to reduce computational cost [27]. While the method itself is not new, the combination of modern-hardware architectures and concurrent subroutines that are AD-capable can unlock optimisation problems in Engineering written in terms of expectations instead of deterministic quantities.

2.3.2 Differentiable-parallel dynamic loads

Once a parallel system is in place to compute hundreds of load cases, the next step is to obtain the derivatives of the critical loads coming from the parallel analysis. Since those are calculated using AD, all the operations need to be available in memory. While the Montecarlo analysis of the previous section made use of all the cases to compute the output expectation, in this case we are only interested on those that induce the largest loads. We encountered two major issues: the memory required for the gust cases was already in the limit of a single device (over 60 GB of RAM), to which the AD normally duplicates the requirement. As the software can now be run on multiple devices, each with its own memory, this is not a completely restrictive factor. The second issue was simply a lack of implementation of the needed collective operations in JAX. This is the case for instance with the maximum function, that can usually be differentiated but not if collective operations are involved (it is also not practical as most of the data generated by such a maximum are zeros not needed anyway). The solution found has been named the Forager Pattern and is depicted in Fig. 2. The code launches many simulations concurrently with the predefined load-cases. The solutions of all these simulations are collected (hundreds of cases, hundreds of nodes, thousands of time steps make for a single field of interest like the stress to have a size of the order of $10^7 - 10^8$). A filtering step consists of a selection of monitoring points of interest (nodes in the FEM), and then a double reduction operation in both time and load cases, for example the maximum of the selected field in time and across cases; the output is a selection of the most problematic load cases according to the predefined metric in the input file. For these critical points the program builds the inputs for the cases previously run in parallel but now with AD and on a much smaller basis, and finally more FENIAX process are spawn for the AD computations. In this way we have created a meta-program

that can automatically create programs based on the results –with limitations on the implemented possibilities.

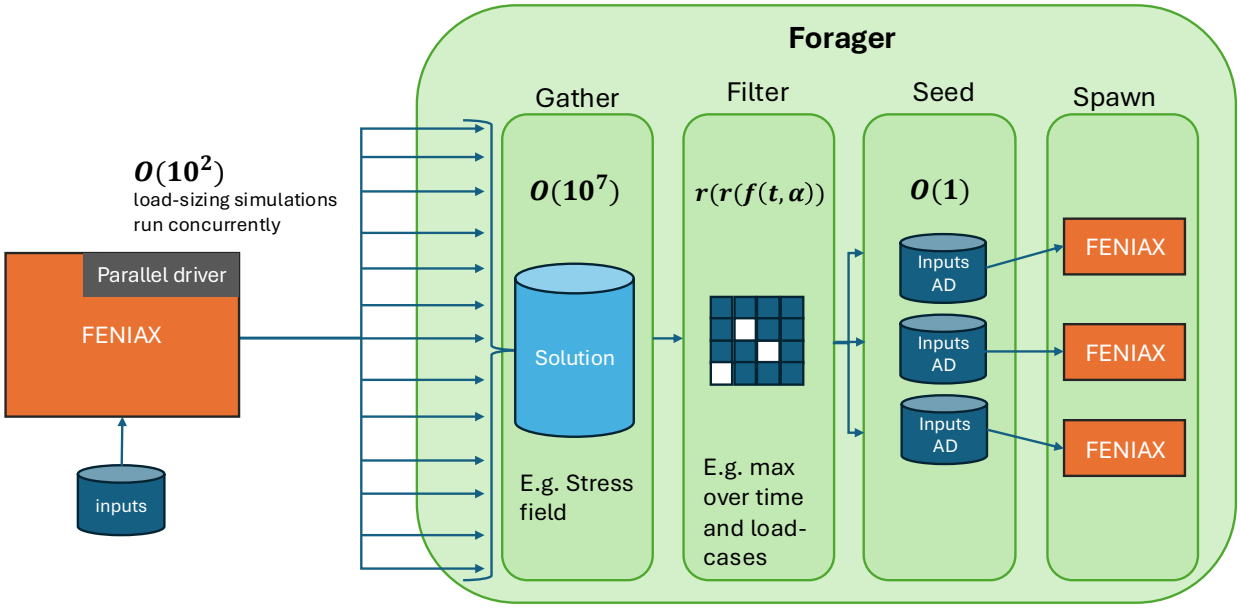

Figure 2: Forager pattern for differentiable-parallel simulations

3 Results

In this section we show the main strengths of our solvers. We run a representative aircraft model undergoing very large nonlinear displacements, the University of Bristol Ultra-Green (BUG) aircraft model [28] is the chosen platform as it is not based on proprietary data and it showcases high-aspect ratio wings and a GFEM based on beam and shell elements in MSC Nastran. The main components of the aeroelastic model have been presented in [29]. Leveraging on modern hardware architectures and a parallelisation across devices, to unlock problems such as quantifying the uncertainties in the nonlinear response given a non-deterministic loading field. We build load envelopes of static and dynamic aeroelastic simulations and differentiate across the concurrent simulations to obtain sensitivities of dynamic loads as well as moment statistics.

Structural and aeroelastic static simulations follow, solved via a Newton-Raphson solver with tolerance of 10^{-6}, as well as an assessment of the aircraft dynamics in response to various gust profiles. A high modal resolution of 100 modes is employed in all the results, more than what is necessary for most of the examples. Calculations are carried out on a CPU Intel Xeon Silver 4108 with 1.80GHz speed, 6 cores and a total 12 threads, as well as on an Nvidia GPU A100 80GB SXM.

3.1 Uncertainty quantification in nonlinear simulations

In this section uncertainty quantification is performed for both linear and nonlinear responses to a loading field that is non-deterministic. Thousands of simulations are employed in Monte Carlo type of analysis to resolve for the statistics, for which parallelisation of the independent simulations become critical. The example resembles the workflow of flight loads and wing stress analysis in an industrial setup. There will always be an element of uncertainty around computed loads, and what we show here is how for large displacements, the statistics need to be computed for every distinct loading. And for this, having a parallisation strategy as the one presented could potentially allow the computation of complex correlations and averages via Montecarlo analysis.
Considering this, a static loading field is prescribed along the wings consisting of follower forces in the normal (out-of-plane) direction, as well as torsional moments (to mimic the added aerodynamic forces on an airfoil), with the characteristic that the force follows a normal distribution:

$$N(\mu = 1.5 \times 10^4 \mu_0, \sigma = 0.15\mu) \text{ [forces]} \tag{5a}$$

$$N(\mu = 3 \times 10^4 \mu_0, \sigma = 0.15\mu) \text{ [moments]} \tag{5b}$$

Three scenarios are studied: one in which very large nonlinear deformations are induced with $\mu_0 = 1$, and two small loading with $\mu_0 = 10^{-2}$ and $\mu_0 = 10^{-3}$. The distribution of displacements is characterised by means of Montecarlo simulations that run in parallel for a total of 1600 simulations. Fig. 3 shows the equilibrium at high loading ($\mu_0 = 1$) for two of the random cases (first and last of the 1600 computed).

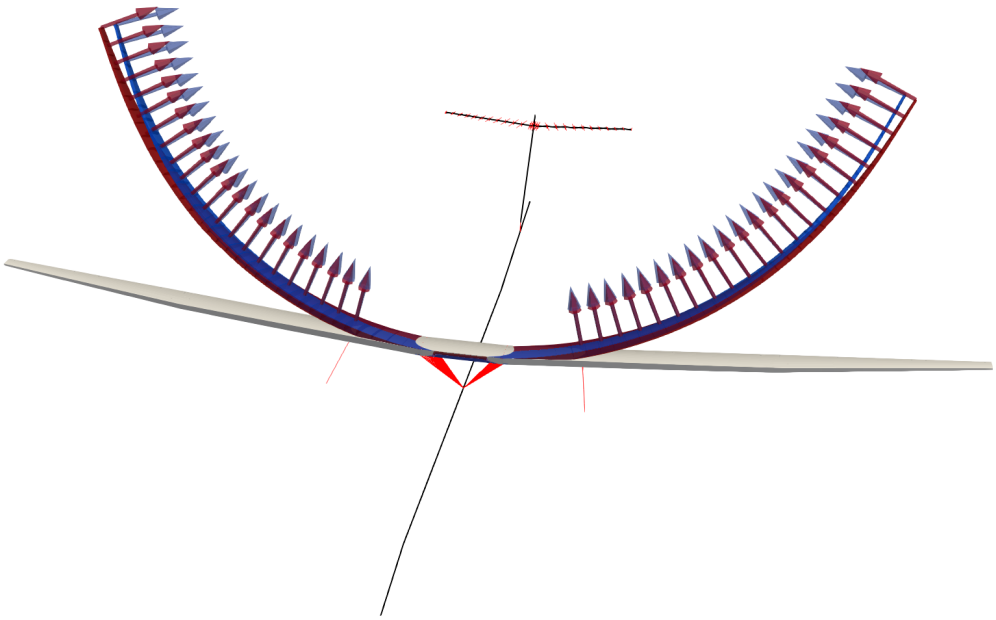

Figure 3: Static equilibrium for two cases of the random excitation ($\mu_0 = 1$)

Table 1 shows the normalized statistics gathered from the response, in this case the expectation

of tip of the wing displacements in the normal (out-of-plane) direction, μ_u, and the corresponding standard deviation, σ_u.

Table 1: Vertical wing-tip displacement statistics

Case	μ_u/μ_0	σ_u/μ_0
$(\mu_0 = 1)$	11.57	1.35
$(\mu_0 = 0.01)$	14.8	2.4
$(\mu_0 = 0.001)$	14.9	2.3

We can see the statistics of the linear response ($\mu_0 = 0.001$) are fully captured by one single Montecarlo analysis, that is, output magnitudes such as equilibrium displacements correlate with the average input load. Whereas in cases with nonlinear deformations ($\mu_0 = 1$), the whole Montecarlo analysis would need to be carried out. This is akin to deterministic linear versus nonlinear analysis. Expanding this data at the tip to the entire right wing, Fig. 4 shows the mean normal displacement along the wing, μ_u, and its standard deviation, σ_u. Note how despite the standard deviation of the input forces is the same along the wing, the uncertainty in the displacement output grows towards the tip as expected –the aircraft being clamped at the root will only showcase 0 displacements there regardless of the input forces.

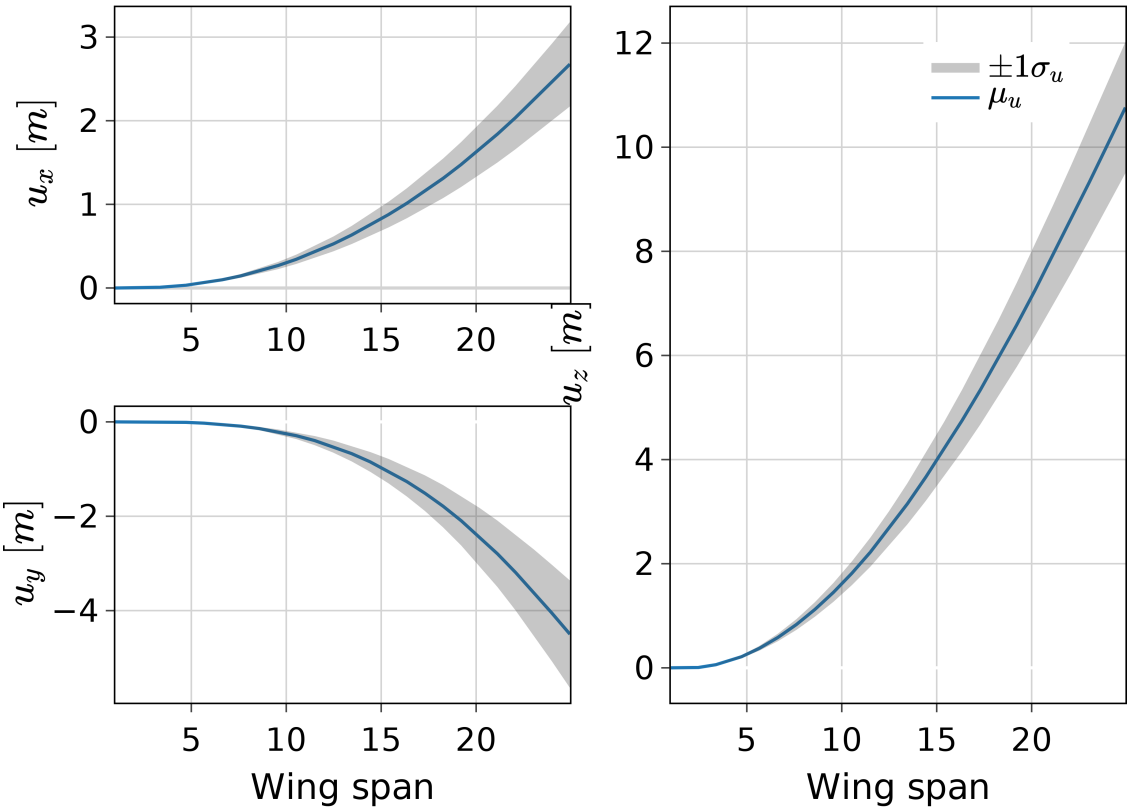

Figure 4: Wing vertical displacements expectations and 1 standard-deviation

Table 2 shows the times taken for the nonlinear case in both CPU and GPU. The computation

of 1600 independent simulations in just over a minute, involving deformations of over 40% the wing semi-span as shown in Fig. 3, highlights the potential of this methodology in more complex uncertainty quantification problems. Note that at this level of nonlinearity, our solvers are already two orders of magnitude faster than commercial solvers such as MSC Nastran even for a single simulation as demonstrated in [8]. The extension to thousands of cases with parallelisation on modern architectures is a key feature of this work with far-reaching applications in aircraft loads analysis.

Table 2: Static loading UQ computational times for 1600 paths

Device	Time [sec.]
CPU (single)	$16.8 \times 1600 = 26880$
CPU (parallel)	317.4
GPU	67.6

3.2 Computing derivatives of expectations

Now we set out to calculate the derivatives of the expectations previously computed concurrently in Sec. 2.3.1. While the Montecarlo paths are independent of each other and could therefore be run on different machines, having to do AD on the output statistics -gathered via collective operations-, forces the entire chain of operations to be within a single program. This makes for an interesting and challenging problem to propagate gradients through concurrent operations. A linear parameter α is introduced such that the follower forces and torsional moments in Eq. (5a) are $\mu = 10^4(\frac{\alpha-1.5}{4-1.5} + 1.5 \times \frac{\alpha-1}{5-1})$. The selected output is the expectation of a 3-component vector, $\boldsymbol{r}(\alpha)$ of the wing-tip positions at $\alpha = 4.5$. Fig. 5 shows a comparison between the derivative $\partial_\alpha \boldsymbol{r}^a = \partial \mathbb{E}[\boldsymbol{r}]/\partial\alpha$ using AD, and finite differences $\partial_\alpha \boldsymbol{r}^f = (\boldsymbol{r}(\alpha+\epsilon) - \boldsymbol{r}(\alpha))/\epsilon$. The relative error is calculated as $||\partial_\alpha \boldsymbol{r}^a - \partial_\alpha \boldsymbol{r}^f||/||\partial_\alpha \boldsymbol{r}^a||$, using the l_2 norm.

Another convergence metric to investigate is the number of paths in the Montecarlo analysis. We take $[8, 80, 400, 800]$ paths and using the same error metric as for the finite differences calculate its evolution taking 4000 paths as the reference. The plot in 6 shows that the convergence of the value function (expectation of the wing tip position) is faster than its derivative with respect to the loading parameter α. This may or not be an issue in optimization studies with expectations as sometimes high accuracy in the gradients is not that important as having a good direction.

While changing the input loading is good for verification studies, more realistic examples are solved in Fig. 7 where the sensitivity of the expectations of wing root loads are calculated with respect to the FE matrices and eigenvectors (only indexes corresponding to the right wing nodes are shown in the heat map as the other are 0 because the model is clamped). The derivatives in Fig. 6 were calculated in forward mode, but the condensed FE matrices of the BUG model contain 99 condensed nodes, each with 6 components, so $(99 \times 6)^2 = 352836$ entries, to what all the components in eigenvectors are also added, therefore only backward mode AD is possible in this case. The sensitivity of the loads with respect to the mass matrix increase towards the nodes at tip of

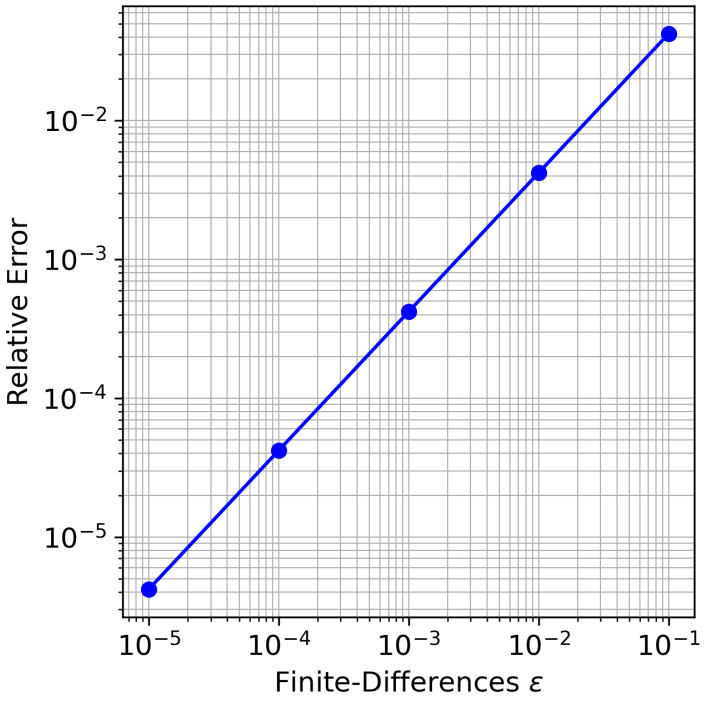

Figure 5: Relative error for the derivative of expectations of the wing-tip position with respect to α (finite-differences VS AD).

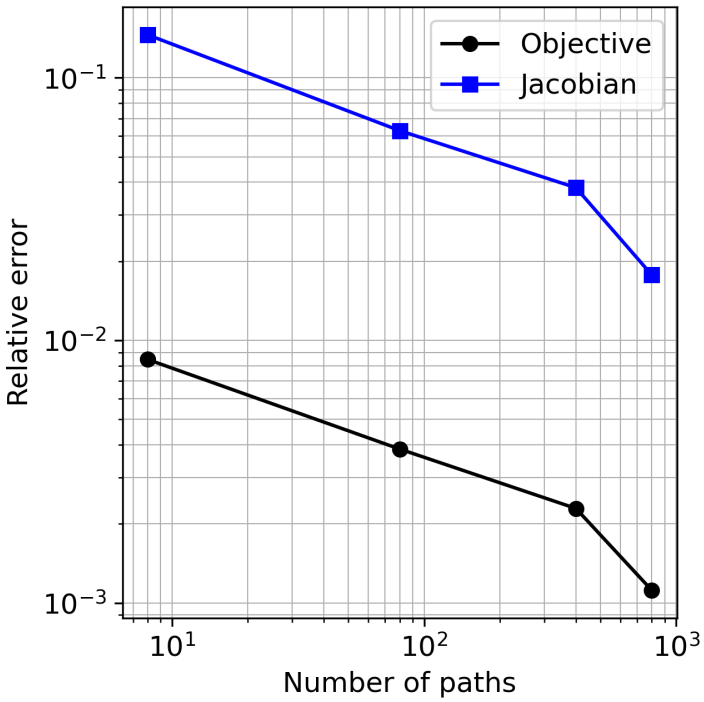

Figure 6: Convergence with number of paths of expected wing-tip position and its Jacobian with respect to α.

the wing, which makes sense if we think a unit mass at the tip will produce much larger loads than the same unit mass close to the root where the aircraft is clamped.
The wall-time to calculate the gradient of wing-root loads expectations with respect to the mass matrix using 800 paths in the Montecarlo analysis was 48.0 and 297.9 in the GPU and CPU respectively.

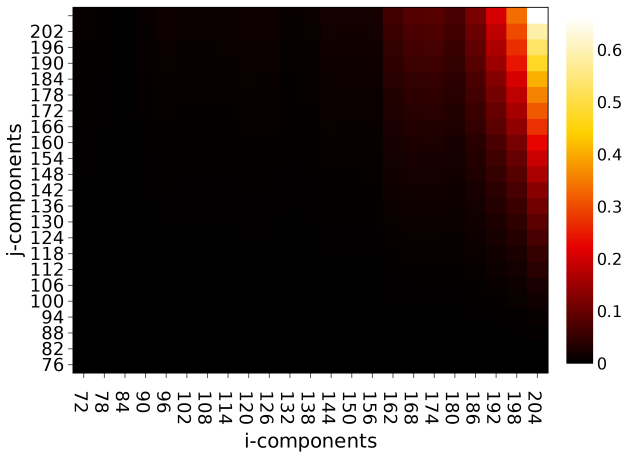

Figure 7: Weight-normalized Jacobian of wing-root bending moment with respect to Mass matrix right-wing subcomponents

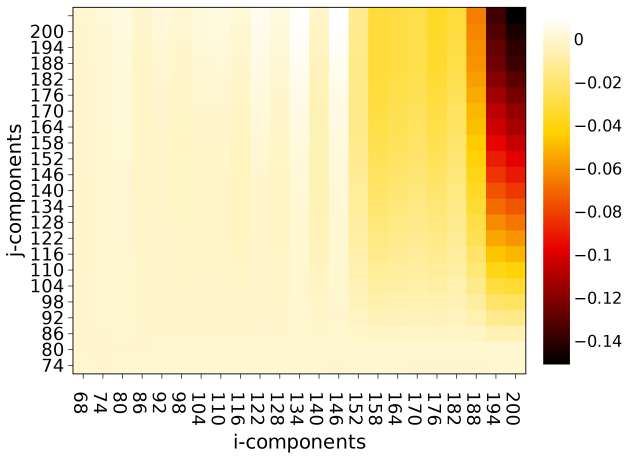

Figure 8: Weight-normalized Jacobian of wing-root shear force with respect to Mass matrix right-wing subcomponents

3.3 Steady manoeuvre loads

We extend the previous analysis to a static aeroelastic case for varying angles of attack that represent a manoeuvre scenario. We test the parallelisation by varying the flow density (±20% of the reference density 0.41 Kg/ m^3) as well and the flow velocity (±20% of the reference velocity 209.6 m/s). 16 different points for both density and velocity make a total number of 256 simulations. The Mach

number is fixed at 0.7 corresponding to the reference flow condition values. Fig. 9 illustrates the 3D equilibrium of the airframe at the reference flight conditions.

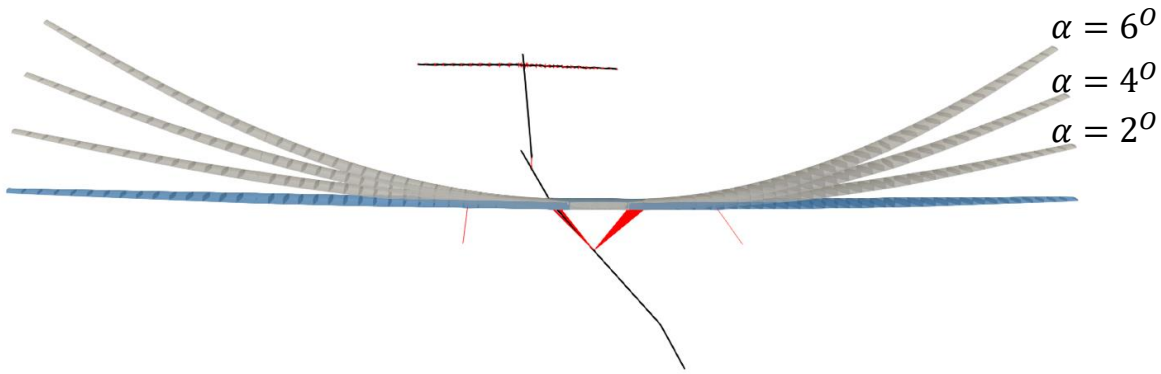

Figure 9: Aeroelastic steady equilibrium for increasing angle of attack manoeuvre

In Fig. 10 the tip of the wing in Fig. 9 is plotted for various angles-of-attach (AoA), normalized with the wing semi-span ($b = 25.9$) m. Comparison against linear analysis is carried out and the tip position in the nonlinear analysis falls down the linear counter part as expected. The flow velocities and density are selected at the maximum of the load envelope at 251.6 m/s and 0.5 kg/m^3 respectively. This highlights the potential need for geometrically nonlinear aeroelastic tools in future aircraft configurations under high loading scenarios.

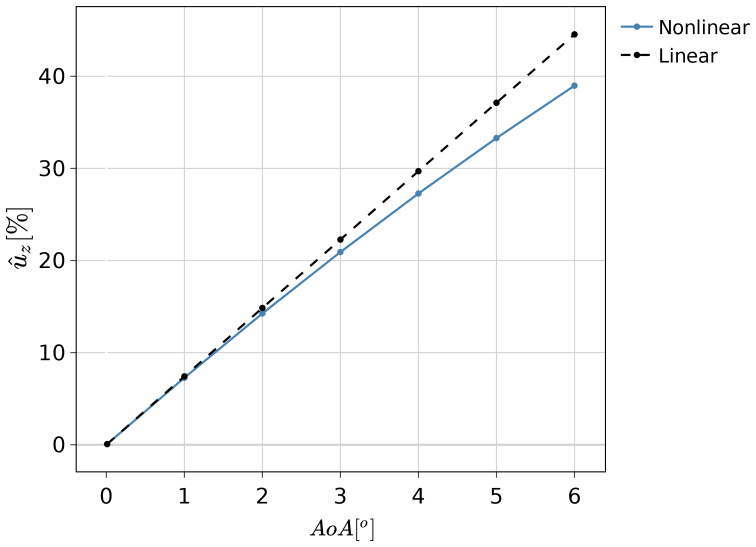

Figure 10: Wing tip position for increasing angle of attack

Table 3 shows the computational times to run these simulations, which shows near no overhead in adding a few hundred of static calculations when moving from the single load case in the CPU to the GPU (nearly 8 seconds to 14 seconds, which amounts for 6 seconds cost when adding an extra 255 cases).

Table 3: Computational times for the multiple manoeuvre problem

Device	Time [sec.]
CPU (single)	$7.71 \times 256 = 1973.8$
CPU (parallel)	52.8
GPU	14.4

3.4 Dynamic loads at large scale

In this final example we perform a dynamic aeroelastic analysis to study the response of the aircraft to multiple 1-cos gusts for varying length, intensity and the density of the airflow. The mach number is kept constant at 0.7. In the examples above the aircraft was clamped while the aircraft is free here. A Runge-Kutta solver is employed to march in time the equations with a time step of 10^{-3} s and the total number of modes used was 100. Note the large size of the aeroelastic ODE system: 2×100 nonlinear equations plus 5×100 linear equations for the aerodynamic states with 5 poles, plus 4 equations for the quaternion tracking the rigid-body motion, for a combined ODE system of 704 equations. In addition, a total of 512 gusts cases are run concurrently for all possible combinations of 8 gust lengths between 50 and 200 meters, 8 gust intensities between 5 and 25 m/s, and 8 airflow densities between 0.34 and 0.48 Kg/m^3. This means that $512 \times 704 = 360448$ equations are being marched in time, in this case for 2 seconds which is enough to capture peak loads. We have verified the concurrent implementation by satisfactory comparing single-point simulations to the same points within the parallel results. Table 4 contains the simulation times of the calculation, which shows one order of magnitude increase in performance when running in parallel in the CPU versus a complete single simulation running sequentially, and another order of magnitude when moving from the CPU to a modern GPU. This exemplifies the power of modern hardware for scientific computation.

Table 4: Computational times multiple gust problem

Device	Time [sec.]
CPU (single)	$27.8 \times 512 = 14233.6$
CPU (parallel)	922.6
GPU	38.2

In Fig. 11 the 3D reconstructed flight shape of the airframe is depicted for a gust of 150 m length, intensity of 20 m/s and flow density of 0.41 Kg/m^3

Figs. 12, 13 and 14 show the load diagrams for the maximum shear, torsion and out-of-pane bending at the wing root during the gust encounter, normalized with the aircraft weight and wing semi-span. They reflect the importance of running multiple of these simulations to assess the critical loads: maximum loads occur at different gust lengths of 65, 75, 115 m/s. This analysis would be extended to include various mass cases, flying altitudes etc. in an industrial environment, and it would be straight forward to extend our tools for this.

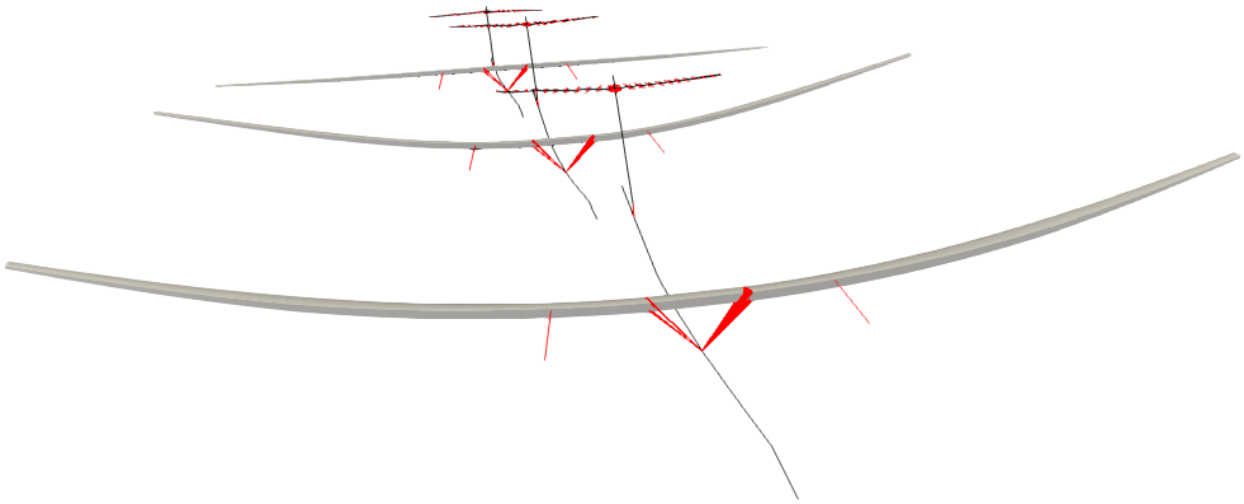

Figure 11: Full aircraft Dynamic response to 1-cos gust excitation

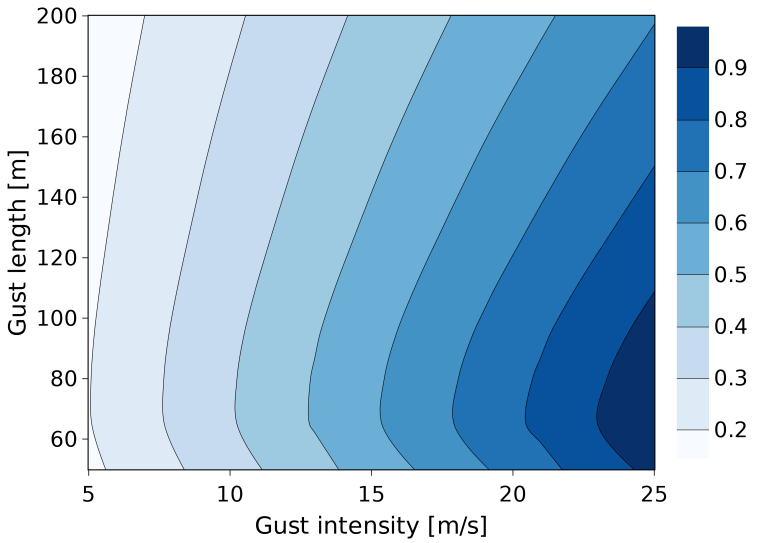

Figure 12: Normalized max. wing-root internal loading, shear force

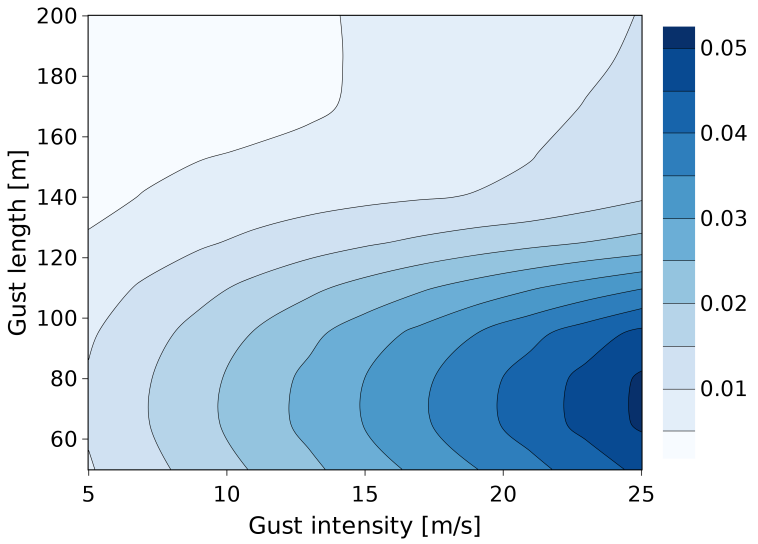

Figure 13: Normalized max. wing-root internal loading, torsional moment

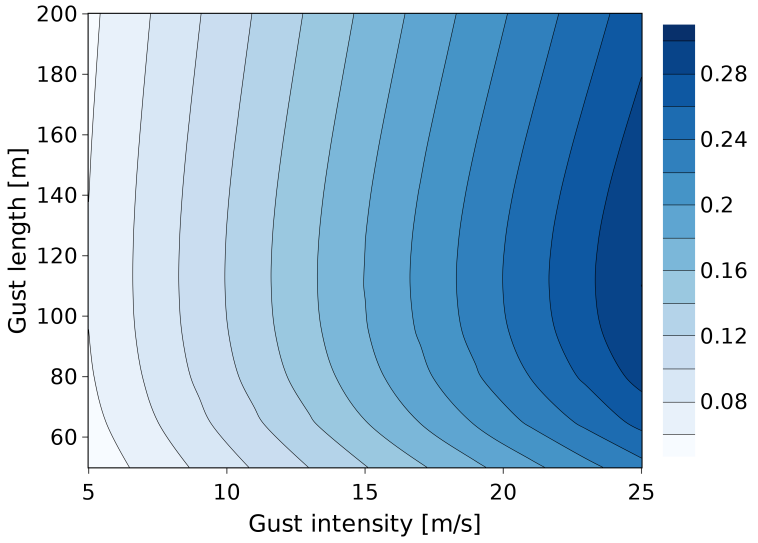

Figure 14: Normalized max. wing-root internal loading, bending moment

3.4.1 Load envelope differentiation

Since dynamic load envelopes have been constructed, the interest is to be able to obtain derivatives at the critical points as described in Sec. 2.3.2. In opposition to the derivatives of expectations where all the operations are needed in the construction of the computational graph, here only a few of the most problematic cases are automatically identifies and its derivatives computed. The metrics being tracked are wing-root shear, torsion and out-of plane bending moments. The parallelisation is set for two gust intensities, two flow densities and 16 gust lengths to cover 1-cos gusts from 50 to 200 m/s with 10 m/s separation between points. Rather than a realistic example, this is set to test the machinery of the forager pattern and verify it can indeed discover critical load cases and automatically compute gradients. The gradient of these critical cases is also calculated with respect to the flow density, gust length and intensity (thus they are not only the parameters for the parallelisation but are also chosen to be the input variables of the sensitivity analysis, though any other input such as FE matrices could have been chosen). By looking at Figs. 12 - 14, we can identify maximum loads at around 65, 75, 115 m/s gust lengths for shear, torsion and bending. Of the 64 cases analyzed, the forager step picked those with higher intensity and flow density as expected, but only 2 gust lengths of 70 m/s for the shear and torsion, and 110 m/s for the out-of-plane bending. The finer the discretization in the input of the parallelization, the closer to the actual maximum, but also the more computations that need to be run. It is therefore a balance whose best compromise is to be found on an actual optimization study. Once those peaks are found, the algorithms triggers the sensitivity analysis, for which we have shown a verification for the 70 m/s gust with maximum density and intensity in Table 5. Finite differences are computed with an $\epsilon = 10^{-4}$ and the absolute relative difference with AD is shown as Δ for each of the input parameters ($\partial\rho_{\text{inf}}$ for the flow density derivative, ∂L for the gust length, and ∂w for the gust intensity). The computational wall-time for entire forager loop (concurrent loads, filtering of critical cases and sensitivities on them) was 60.1 and 359.5 seconds on the GPU and CPU respectively.

Table 5: Comparison of peak gust loads at the critical points, AD versus FD

	objective	$\partial\rho_{\text{inf}}$	Δ	∂L	Δ	∂w	Δ
Shear	0.897	1.670	1.6×10^{-5}	-0.0026	4×10^{-6}	0.035	$1. \times 10^{-7}$
Torsion	0.025	-0.018	4.2×10^{-4}	-0.0007	3.9×10^{-7}	0.0018	1.2×10^{-6}
Bending	0.300	0.329	2.9×10^{-5}	0.00003	1.7×10^{-4}	0.012	9.2×10^{-8}

4 Conclusions

A modal-based, geometrically nonlinear formulation has been implemented with concurrent capabilities for multiple load-cases in modern hardware architectures. We have applied state-of-the-art techniques and tools to the computation of the sizing aeroelastic loads encountered during the design of commercial aircraft. These can span thousands of simulations in multiple scenarios and the

efficiency of our approach has been highlighted while accounting for nonlinear effects and rigid body dynamics. Moreover, sensitivity of the response to model inputs using automatic differentiation has been demonstrated across parallel simulations. Computational times of under a minute are achieved for 256 manoeuvres varying flow conditions and for 512 dynamic gust responses undergoing large displacements. This performance makes the solution suitable for two different applications: uncertainty quantification of the nonlinear aircraft response to a non-deterministic loading and integration of the software in larger multidisciplinary optimisation studies. The former has been presented on a problem where a field of forces with an stochastic component induces very large deformations. Montecarlo analysis then allows to build statistics of the response and we have shown how we can not only run multiple paths concurrently to build a distribution of the outputs, but also differentiate expectations of those with respect to any input in the aeroelastic analysis. For the sensitivity analysis of load envelopes, a strategy named The Forager Pattern has been introduced whereby the multiple cases are launched in parallel, critical cases are filtered out, and new simulations for AD are spawn to obtain gradients on the most demanding loads. Since the examples shown are already in the memory boundaries of a single GPU or CPU, further scale-up requires the use of multiple devices, which we have already implemented. Introduction of various mass cases, as it is done in industrial scenarios, is also a feasible target. The resulting framework thus combines prediction of sizing aeroelastic loads in a nonlinear fashion with the computation of their gradients with respect to design variables, and therefore it is intended to be part of multidisciplinary design optimization studies that include the large set of loading requirements in the certification process. Despite dynamic loads and aeroelastic instabilities being still obtained using medium fidelity tools, a future enhancement with high fidelity CFD aerodynamics is the one major addition left to achieve a versatile, production-ready framework.

Acknowledgements

This work has received funds from Innovate UK, under project 10002372, managed by the UK Aerospace Technology Institute. We acknowledge computational resources and support provided by the Imperial College Research Computing Service (`http://doi.org/10.14469/hpc/2232`). The benchmark examples for the GPU computations were carried out at the Imperial College cluster. Code and examples are open-source.

References

[1] Mayuresh J. Patil and Dewey H. Hodges. On the importance of aerodynamic and structural geometrical nonlinearities in aeroelastic behavior of high-aspect-ratio wings. *Journal of Fluids and Structures*, 19(7):905–915, 2004.

[2] Amir H. Modaress-Aval, Firooz Bakhtiari-Nejad, Earl H. Dowell, David A. Peters, and Hossein Shahverdi. A comparative study of nonlinear aeroelastic models for high aspect ratio wings. *Journal of Fluids and Structures*, 2019.

[3] Eli Livne. Aircraft Active Flutter Suppression: State of the Art and Technology Maturation Needs. *Journal of Aircraft*, 55(1):410–452, January 2018.

[4] Eirikur Jonsson, Cristina Riso, Bernardo Bahia Monteiro, Alasdair C. Gray, Joaquim R. R. A. Martins, and Carlos E. S. Cesnik. High-Fidelity Gradient-Based Wing Structural Optimization Including Geometrically Nonlinear Flutter Constraint. *AIAA Journal*, 61(7):3045–3061, July 2023.

[5] Thiemo M Kier. An integrated model for lateral gust loads analysis and dutch roll flight dynamics using a 3d panel method. International Forum on Aeroelasticity and Structural Dynamics 2013, 2017.

[6] Graeme J. Kennedy and Joaquim R.R.A. Martins. A parallel finite-element framework for large-scale gradient-based design optimization of high-performance structures. *Finite Elements in Analysis and Design*, 87:56–73, September 2014.

[7] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: Composable transformations of Python+NumPy programs, 2018.

[8] Alvaro Cea and Rafael Palacios. JAX-based aeroelastic simulation engine for differentiable aircraft dynamics. *Computer Physics Communications*, 311:109547, June 2025.

[9] Alvaro Cea, Rafael Palacios, and Lucian Iorga. Near-Real-Time Nonlinear Analysis of Free-Flying Flexible Aircraft on Modern Hardware Architectures. *AIAA Journal*, 2025 (Accepted).

[10] Timothy R. Brooks, Joaquim R.R.A. Martins, and Graeme J. Kennedy. High-fidelity aerostructural optimization of tow-steered composite wings. *Journal of Fluids and Structures*, 88:122–147, July 2019.

[11] Alvaro Cea and Rafael Palacios. Geometrically Nonlinear Effects on the Aeroelastic Response of a Transport Aircraft Configuration. *Journal of Aircraft*, 60(1):205–220, January 2023.

[12] Jens-Dominik Mueller, Jan Hueckelheim, and Orest Mykhaskiv. STAMPS: A Finite-Volume Solver Framework for Adjoint Codes Derived with Source-Transformation AD. In *2018 Multidisciplinary Analysis and Optimization Conference*, Atlanta, Georgia, June 2018. American Institute of Aeronautics and Astronautics.

[13] Ping He, Charles A. Mader, Joaquim R. R. A. Martins, and Kevin J. Maki. DAFoam: An Open-Source Adjoint Framework for Multidisciplinary Design Optimization with OpenFOAM. *AIAA Journal*, 58(3):1304–1319, March 2020.

[14] Thomas D. Economon, Francisco Palacios, Sean R. Copeland, Trent W. Lukaczyk, and Juan J. Alonso. SU2: An Open-Source Suite for Multiphysics Simulation and Design. *AIAA Journal*, 54(3):828–846, March 2016.

[15] Walter A. Silva and Robert E. Bartels. Development of reduced-order models for aeroelastic analysis and flutter prediction using the CFL3Dv6.0 code. *Journal of Fluids and Structures*, 19(6):729–745, 2004.

[16] Nikolaos Simiriotis and Rafael Palacios. A numerical investigation on direct and data-driven flutter prediction methods. *Journal of Fluids and Structures*, 117:103835, February 2023.

[17] John A. Schaefer, Vicente J. Romero, Steve R. Schafer, Brian Leyde, and Casey L. Denham. Approaches for Quantifying Uncertainties in Computational Modeling for Aerospace Applications. In *AIAA Scitech 2020 Forum*, Orlando, FL, January 2020. American Institute of Aeronautics and Astronautics.

[18] Alex Feldstein, David Lazzara, Norman Princen, and Karen Willcox. Multifidelity data fusion: Application to blended-wing-body multidisciplinary analysis under uncertainty. *AIAA Journal*, 58(2):889–906, 2020.

[19] Dewey H. Hodges. Geometrically Exact, Intrinsic Theory for Dynamics of Curved and Twisted Anisotropic Beams. *AIAA Journal*, 41(6):1131–1137, June 2003.

[20] Rafael Palacios and Bodgan Epureanu. An intrinsic description of the nonlinear aeroelasticity of very flexible wings. *52nd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*, (April):1–15, 2011.

[21] Alvaro Cea and Rafael Palacios. A non-intrusive geometrically nonlinear augmentation to generic linear aeroelastic models. *Journal of Fluids and Structures*, 101:103222, February 2021.

[22] Alvaro Cea and Rafael Palacios. Differentiable Aeroelastic Framework Suitable For Industrial Modelling Of Nonlinear Loads On Accelerators. *International Forum on Aeroelasticity and Structural Dynamics 2024, IFASD 2024*, 2024.

[23] Kenneth L Roger. Airplane math modeling methods for active control design. *Structural Aspects of Active Controls, AGARD CP-228*, pages 4.1–4.11, 1977.

[24] William S. Moses, Valentin Churavy, Ludger Paehler, Jan Hückelheim, Sri Hari Krishna Narayanan, Michel Schanen, and Johannes Doerfert. Reverse-mode automatic differentiation and optimization of GPU kernels via enzyme. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16, St. Louis Missouri, November 2021. ACM.

[25] Max Sagebaum, Tim Albring, and Nicolas R. Gauger. High-Performance Derivative Computations using CoDiPack. *ACM Transactions on Mathematical Software*, 45(4):1–26, December 2019.

[26] Patrick Kidger. *On Neural Differential Equations*. PhD thesis, University of Oxford, 2021.

[27] Michael B. Giles. Multilevel Monte Carlo Path Simulation. *Operations Research*, 56(3):607–617, June 2008.

[28] Olivia Stodieck, Jonathan E. Cooper, S. A. Neild, M. H. Lowenberg, and L. Iorga. Slender-Wing Beam Reduction Method for Gradient-Based Aeroelastic Design Optimization. *AIAA Journal*, 56(11):4529–4545, November 2018.

[29] Alvaro Cea and Rafael Palacios. Geometrically Nonlinear Aircraft Loads at Large Scale Using an Accelerator-Based Concurrent Solution. In *AIAA SCITECH 2025 Forum*, Orlando, FL, January 2025. American Institute of Aeronautics and Astronautics.