# Efficient derivative computation for unsteady fatigue-constrained nonlinear aero-structural wind turbine blade optimization

Adam Cardoza[1] and Andrew Ning[1,2]

[1]Brigham Young University, Provo, UT, 84602
[2]National Laboratory of the Rockies, Golden, CO, 80401

**Correspondence:** Adam Cardoza (adam.cardoza@byu.edu)

**Abstract.** Gradient-based optimization offers significant efficiency advantages for wind turbine blade design, but its application has often been limited by the cost and accuracy of finite difference derivative calculations, especially when fatigue constraints are considered. In this work, we systematically compare and evaluate four differentiation techniques, namely algorithmic differentiation, implicit differentiation, sparsity exploitation, and parallelization, to determine their effectiveness in computing accurate gradients through time-domain aero-structural simulations. By integrating these techniques with unsteady nonlinear aerodynamic and structural models, we develop software designed for accurate gradient computation. We show that combining these techniques addresses memory and runtime challenges associated with long simulations required by design load cases. Specifically, the most effective combination reduces derivative computation wall time by over an order of magnitude compared to finite differencing while maintaining superior accuracy. We demonstrate this approach in a proof-of-concept aero-structural optimization of a wind turbine blade that improves the cost of energy by 11.4%. This comparative study establishes a viable approach for fatigue-aware blade design that balances computational efficiency with modeling accuracy.

## 1 Introduction

Designing a cost-effective wind turbine blade is a difficult task for many reasons, but one salient challenge is including the large number of variables required to fully specify a design. Gradient-based optimization provides a systematic approach to this problem, as it typically finds optima more efficiently than gradient-free methods for design problems with many variables (Ning and Petch, 2016). Gradient-based optimization adds the additional challenge of obtaining the derivatives of the objective and constraints with respect to the design variables.

This challenge can be broken into two main categories: 1) derivative accuracy and 2) derivative computation time. Traditionally derivatives have been calculated using finite difference methods, which suffer in both accuracy and computation time. Despite the challenges, many have done excellent work in this field using finite differences. Døssing et al. (2012), Kwon et al. (2015), and Frontin et al. (2023) completed aerodynamic-only blade optimizations. Several have used traditional forms of finite differencing to complete aero-structural optimization (Bottasso et al., 2016; Zahle et al., 2016; Pavese et al., 2017; Bortolotti

et al., 2018; McWilliam et al., 2018b, a; Ingersoll, 2020; Scott et al., 2022; Serafeim et al., 2022). Kenway and Martins (2008) and Madsen et al. (2022) used a complex-step finite difference method[1].

25     Algorithmic differentiation (AD) provides a powerful alternative to finite difference methods that directly addresses the challenges of gradient-based optimization. AD computes the derivatives of the individual operations of an analysis software. Iterative application of the chain rule propagates these derivatives through the code. This approach provides the desired derivatives of function outputs (Naumann, 2011). The main added cost of using AD is that the value of the derivative and tracking information must be stored, which results in increased memory requirements. AD approaches have already been included in 30    several works such as Ning and Petch (2016), Dhert et al. (2017), and Allen et al. (2020).

    The computational cost of AD increases substantially when propagated through iterative solvers. Implicit differentiation methods offer an attractive solution by analytically computing derivatives of equations solved iteratively, bypassing the need to differentiate through the solver itself. Several studies have successfully applied implicit differentiation approaches (Dhert et al., 2017; Madsen et al., 2019; Batay et al., 2023; Mangano et al., 2025; Hermansen et al., 2025). However, manual implemen- 35   tation of implicit differentiation is often cumbersome and problem-specific. In previous work, we demonstrated how implicit differentiation can be automated by applying AD to converged solver residuals, enabling efficient derivative computation with minimal implementation effort for AD-compatible solvers (Ning and McDonnell, 2023).

    Derivative computation can be made more efficient by taking advantage of two key properties: sparsity and parallelization. In many engineering problems, Jacobians are sparse, meaning that many of the entries in the Jacobian are zero. By identifying 40   and leveraging this sparsity structure, techniques such as graph coloring can substantially reduce the number of derivative computations required. In previous work, we demonstrated how exploiting sparsity reduced the derivative computation cost in a steady aerodynamic optimization problem (Ning, 2021) and in wind farm layout optimization (Varela and Ning, 2024). Complementing this approach, parallelizing the derivative computations across multiple processors can further decrease the wall time required.

45     The design of a cost-effective wind turbine blade is inherently an aero-structural problem (Bottasso et al., 2016). One critical aspect of wind turbine design is accounting for the effects of fatigue in the design. Several different approaches have been used to include fatigue in the design such as iterative methods (Bottasso et al., 2016; Bortolotti et al., 2018; Ingersoll, 2020), simplified models (Ning and Petch, 2016), frequency-domain approaches (Pavese et al., 2017), and precomputing fatigue loads and scaling them by the static stress (Hermansen et al., 2025). Although these approaches have been successful, they do not 50   incorporate fatigue calculations based on the unsteady loads of the current design directly into the optimization loop. As a result, the optimization algorithm cannot fully explore the design space and often creates over-engineered designs.

    While fatigue has been incorporated into wind turbine optimization through various approximations, Scott et al. (2022) directly compute fatigue damage from time-series simulations within the optimization loop. Their approach evaluates unsteady

---

[1]The complex step finite difference method can achieve machine precision accuracy in derivative computation. While making code compatible with complex numbers is typically straightforward, the primary limitation is that complex step only operates in forward mode. This makes it computationally expensive for problems with many design variables, where reverse mode automatic differentiation would be more efficient. See our later discussion on derivative scaling in Sect. 3.2.

loading histories, applies the rainflow counting algorithm, and calculates fatigue damage explicitly rather than relying on
surrogate models or approximations. Their approach, however, depends on simplified modeling assumptions. Their approach
employed a linearized aerodynamic model coupled with a modal beam model that neglects geometric nonlinearities. As with
other linearized models, this formulation loses accuracy when evaluated away from the linearization point. Scott et al. report
that this leads to overpredicted axial and torsional deflections, reduced aero-elastic damping, and inflated estimates of flap-wise
damage equivalent loads.

Analyzing fatigue in the optimization loop significantly increases the difficulties already presented by gradient-based opti-
mization. The foremost challenge is obtaining accurate gradients through time. Time-domain gradient computation presents
significant challenges, as Zahle et al. (2016) observed when computing finite difference derivatives of tip-tower clearance.
More recent work (Scott et al., 2022; Frontin et al., 2023) has shown ways to improve the accuracy of unsteady gradients.
In addition to using more accurate differentiation techniques, tight convergence of the aerodynamic and structural states can
improve accurate gradient computation. A secondary challenge that arises is the length of time-domain analyses required to
compute fatigue loads. The design load cases (DLCs) require that wind turbine designs account for 600 seconds of turbulent
inflow (International Electrotechnical Commission, 2019). Long simulations not only directly increase the derivative compu-
tation time, but also increase the memory requirements of AD. These challenges make the choice of analysis software critical.
While higher fidelity software increases the accuracy of solution (and thus the gradient), it comes with time and memory re-
quirements that make it intractable to include long enough time domain simulations to conduct meaningful fatigue analysis in
the optimization loop. Furthermore, current mid-fidelity software that would not be limited by time and memory requirements
are not compatible with modern derivative computation methods.

We address the challenges of conducting fatigue analysis within the optimization loop by coupling unsteady nonlinear
aerodynamic and structural solvers designed for accurate and efficient derivative calculation. To identify the most effective
approach, we systematically compare multiple differentiation techniques: forward and reverse mode AD, implicit differentia-
tion, parallelized implementations, and sparse formulations. We first introduce the model and its sub-models, then present the
various differentiation approaches in the methods section. After describing the optimization problem, we verify the model and
compare the computational performance of each differentiation technique before discussing the optimization results.

## 2 Methods

We first describe the aeroelastic model's coupling strategy and sub-models, establishing the computational structure through
which derivatives must be propagated. The subsequent subsections detail our approach to derivative computation, including al-
gorithmic differentiation (AD), automated adjoints, parallelization, sparsity exploitation, and constraint aggregation—techniques
that enable practical gradient-based optimization of this complex system.

## 2.1 Model overview

85 Our model architecture draws inspiration from the National Laboratory of the Rockies' WEIS framework[2], which integrates OpenFAST[3], PreComp[4], and pCrunch[5]. We developed our own implementation rather than adopting these established tools for two key reasons. First, implementing everything in Julia (Bezanson et al., 2017) provides seamless access to automatic differentiation packages with minimal code modification. Julia's just-in-time compilation also delivers computational performance comparable to Fortran or other compiled languages. Second, our structural solver, GXBeam[6], employs constant property linear

90 elements with extended Milenković parameters. This formulation produces an exceptionally robust solver that avoids excessive quadrature, enabling tight convergence of structural states at each time step. Such robust convergence stabilizes the entire coupled aeroelastic solution and proves critical for obtaining accurate gradients through time. This Julia-based implementation is not intended to replace OpenFAST, but rather serves as a research platform for rapidly prototyping and evaluating differentiation techniques. Successful approaches demonstrated here could inform future implementation in OpenFAST or similar

95 established software packages.

Figure 1 summarizes the overall model, which may be partitioned into three distinct stages: a precomputation step, a time-domain analysis, and a post-processing step. The core of the time-domain analysis is a two-way coupled, partitioned model that converges across time. It uses blade element moment theory (BEMT) to predict an angle of attack and an inflow velocity. The model uses the angle of attack and inflow angle to predict time-varying loads via the Beddoes-Leishman dynamic stall

100 model. Geometrically exact beam theory (GEBT) uses those loads to predict the dynamic response of the wind turbine blade. The linear and angular deflections and velocities are used to augment the inflow conditions seen by the BEMT for the next time step. The code of the time-domain analysis is implemented in the package WATT.jl[7]. Verification of the current model will be shown in Sect. 3.1.

The precompute step uses classical laminate theory (CLT) to calculate the required mass and compliance properties. Also

105 computed before run time is a turbulent wind field using TurbSim[8]. The post-process step uses CLT to calculate the strain in each cross-section. Rainflow counting is used to separate the strain signal into load ranges and corresponding counts. Miner's rule is used to linearly accumulate damage. In the following sub-sections, we discuss each of the sub-models. Sub-models that are not referenced in their own repositories are included in this paper's companion repository.

### 2.1.1 Blade element momentum theory

110 BEMT theory is well-documented in standard aerodynamics texts (Burton et al., 2011), so we focus here on its application within our aero-structural coupling framework. For our BEMT we use Ning's (Ning, 2021) one-residual formulation inside

---

[2]https://github.com/WISDEM/WEIS
[3]https://github.com/OpenFAST/openfast
[4]https://www.nrel.gov/wind/nwtc/precomp
[5]https://github.com/NREL/pCrunch
[6]https://github.com/byuflowlab/GXBeam.jl.git
[7]https://github.com/byuflowlab/WATT.jl.git
[8]https://www.nrel.gov/wind/nwtc/turbsim
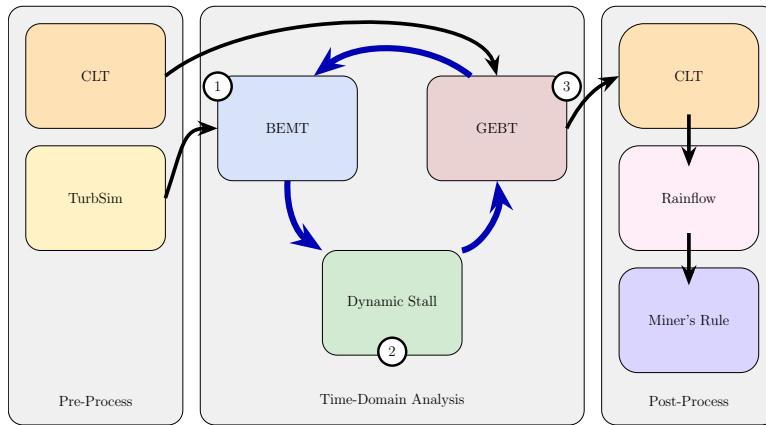
**Figure 1.** Flowchart of the coupled wind turbine blade model.

of the code package CCBlade.jl[9]. Using a BEMT with one residual allows for solvers with guaranteed convergence such as Brent's method (Brent, 2013). This reliability transfers through to the derivative calculation providing smooth derivatives of the aerodynamic solution across the design space.

115    Prandtl tip corrections are applied at both the hub and tip. The 2D airfoil polars from the wind turbine description (Resor, 2013) are provided at 17 spanwise stations and have been smoothed using B-splines. Linear interpolation between these stations produces airfoil polars at each blade node. This smoothing and interpolation is performed as a preprocessing step, while Akima splines (Akima, 1970) interpolate each polar during the solution process.

### 2.1.2 Dynamic stall model

120 To enable unsteady analysis with BEMT, we implement the Beddoes-Leishman dynamic stall model incorporating Gonzalez's modifications (Jonkman et al., 2017). Our implementation follows the discrete ODE formulation from OpenFAST v3.5.1 and is available in DynamicStallModels.jl[10]. The state predictions match those of OpenFAST to machine precision.

### 2.1.3 Turbulence model

In the pre-process step (and outside of all optimization loops), we generate a spatially uniform turbulent wind field using
125 TurbSim. The outputs used at every time step of the time-domain analysis are the three components of the wind speed ($U$, $V$, and $W$). For simplicity, the velocity is assumed to be uniform across the rotor plane; however, WATT.jl could be easily extended to accommodate a spatially varying wind field. WATT.jl is already capable of using a temporal discretization independent of the time step used in the turbulence model, but the results are co-located for the purpose of this study.

---

[9]https://github.com/byuflowlab/CCBlade.jl
[10]https://github.com/byuflowlab/DynamicStallModels.jl

### 2.1.4 Geometrically exact beam theory

130 As previously mentioned, we use the nonlinear GEBT solver GXBeam.jl (McDonnell and Ning, 2024) to predict the structural response of the wind turbine blade. The formulation of GXBeam allows large deflections and rotations, while maintaining solution robustness and derivative stability. GXBeam uses the Newmark-beta method to integrate the equations of motion across time. The inputs to the model are the loads, the blade geometry, and the material properties. The outputs used for the aero-structural coupling are the linear/angular deflections and velocities. The internal forces and moments are also available

135 for post-processing. The blade is modeled as a cantilevered beam with a fixed root and free tip. Aerodynamic loads are rotated into the root reference frame and applied as non-following distributed loads, which are assumed constant across each element.

### 2.1.5 Classical laminate theory

To prepare the cross-sectional properties for the beam model, we use CLT to calculate the mass and compliance properties of the cross-section. Since traditional classical laminate theory is designed for composite flat plates, we use the Kollar and Springer

140 formulation (Kollár and Springer, 2009) in the package GXBeamCS.jl[11]. This formulation allows for the calculation of the mass and compliance properties of arbitrary cross-sections in a similar manner to PreComp. Key differences are the implementation of the shear flow model, compatibility with AD methods, and post-processing capabilities. Although classical laminate theory has its limitations in precisely predicting cross-sectional compliance, its accuracy remains sufficient for practical purposes, and its mesh-free nature greatly simplifies analysis and parameterization for optimization. Analysis is done at the elastic center and

145 in the principal axes, so calculation of the axial strain is straightforward.

### 2.1.6 Material failure

We used three approaches for material failure analysis: ultimate strain, panel buckling, and linear accumulation of fatigue damage. The strain calculated in the cross-section analysis can be used to calculate material failure by comparing against the ultimate strain. Buckling is predicted using a simple panel buckling method (Bir, 2001).

150 To predict the fatigue damage, we use ASTM's range-pair rainflow counting algorithm (American Society for Testing and Materials, 2017) to separate the strain signal into load ranges and corresponding counts. Before applying Miner's rule, we apply the Goodman correction. Typically, fatigue damage is accumulated by analyzing the stress signal, however; we directly use the strain signal to avoid including the inaccuracies included in the compliance matrix from CLT. This action is equivalent to assuming that all of the off-diagonal terms of the compliance matrix are zero. This assumption is reasonable for our case

155 since the CLT approach does not account for these compliance values and are already assumed to be zero.

## 2.2 Derivative computation

We perform all optimizations using SNOPT, a sequential quadratic programming algorithm (Gill et al., 2005). SNOPT uses a quasi-Newton approximation of the Hessian and thus only requires the computation of first derivatives of the objective

---

[11]https://github.com/byuflowlab/GXBeamCS

function and constraints. In this work, we scale all derivatives by first normalizing function outputs to be of order one, followed
160    by scaling the design variables such that the resulting gradient entries are also of order one.

The accuracy and efficiency of these derivatives are critical. Although derivative evaluation typically governs the overall
computational cost, inaccurate derivatives can slow convergence or even prevent the optimizer from converging. Given this
dependence on accurate and efficient derivatives, the method of derivative computation should become a primary consideration
of design optimization. While calculating derivatives using finite differencing is straightforward to apply, it is not efficient
165    and is inaccurate. The inaccuracies of finite differencing stem from subtractive cancelation and truncation error (neglecting
higher-order terms in the Taylor expansion). As a result, the first order central difference of a function with an optimal step size
typically has errors on the order of 1e-6 to 1e-8 (if using 64 bit floating point numbers). Higher-order finite difference formulas
can reduce truncation error, but at the expense of additional function evaluations. More importantly, finite differencing scales
poorly with problem size: at minimum, one function evaluation is required per design variable, while a central difference
170    requires two evaluations per variable, and higher-order schemes require more.

### 2.2.1 Algorithmic differentiation

As previously mentioned, automatic differentiation (AD) provides a means of computing derivatives both more accurately and
more efficiently than finite differencing. AD calculates the derivatives of the function by calculating the partial derivatives of
each elementary operation in the code and then combining them using the chain rule (Naumann, 2011). In AD, the derivatives
175    are computed after each operation is performed and then propagated through the code to obtain the function output and its
derivative with respect to the function input. Because the derivatives are calculated analytically for each operation, the precision
of the derivatives can match the precision of the function evaluation. The exact nature of AD computed derivatives does not
mean that the approach is impervious to error. Like any floating-point computation, numerical error can still accumulate through
round-off error, particularly in complex models with many operations. This challenge is particularly poignant when iterative
180    solvers are used, such as in Brent's method for the BEMT or in the Newmark-beta method for the GEBT.

Several factors influence the efficiency of AD. There are two modes of AD: forward mode and reverse mode. In forward
mode, the derivatives are propagated from the inputs to the outputs, while in reverse mode, the derivatives are propagated from
the outputs to the inputs. Since reverse mode backpropagates the derivatives in reverse, it requires a computational graph to
be saved so that the derivative can be calculated in a reverse pass. Saving the computational graph increases memory usage,
185    which can become prohibitive if the computational graph is large. Forward mode AD, similar to finite differencing, calculates
an entire column of the Jacobian at a time; whereas reverse mode AD calculates an entire row of the Jacobian at a time. This
means that the computational cost of forward mode AD scales linearly with the number of inputs while the computational cost
of reverse mode AD scales linearly with the number of outputs. Thus, reverse mode AD is more efficient for functions with
many inputs and few outputs, while forward mode AD is more efficient for functions with few inputs and many outputs. In
190    practice, forward mode is also more efficient for square Jacobians because of the overhead cost of storing the computational
graph. Having two modes of AD to choose from allows the user to select the most efficient method for their specific problem. In

this work we use the package ForwardDiff.jl (Revels et al., 2016) and ReverseDiff.jl[12] for forward and reverse mode operator overloaded AD respectively.

One feature of ForwardDiff.jl with particular relevance to our study is chunking. ForwardDiff stores the primal (the function value) and derivative of each operation in an object called a dual number, rather than as a typical floating-point number. Each dual number can be tagged to several design variables, so the derivative of multiple design variables can be calculated in the same function call, at the expense of additional memory. These groups of design variables are called chunks. Theoretically, given enough memory, the chunk size can be increased to include all design variables, allowing the entire Jacobian to be calculated in a single function call, but typically memory limitations keep the chunk size around eight. The chunk size is determined by heuristics within ForwardDiff.jl and for this optimization problem is set to eight. While increasing the size of a chunk does not decrease the total number of computations required, it can improve the efficiency of the derivative computation process by reducing the overhead associated with running the code multiple times.

### 2.2.2 Automated implicit analytic differentiation

Implicit differentiation can be used to further improve the efficiency of derivative computation, specifically when iterative solvers are used. The operation of solving for the derivative of a nonlinear system of equations can be reduced to a single implicit analytic operation (Martins and Ning, 2021). Rather than differentiating through each solver iteration, implicit differentiation exploits the fact that the converged residual equals zero. By applying the chain rule to this converged residual equation, we obtain a linear system whose solution provides the desired derivatives. The required partial derivatives of the residual can be computed using AD. This process can be generalized to any solution process for which the residual can be evaluated with AD. In this work we automate this process by using the package ImplicitAD.jl, which we developed in previous work[13] (Ning and McDonnell, 2023).

An additional benefit of the automated adjoint, particular to using ForwardDiff and PolyesterForwardDiff, is that the nonlinear solve only need be computed once for a chunk. Because the primal solution is independent of the dual values, it can be applied to all design variables contained within that chunk. In principle, the primal solution may also be cached and reused across different chunks, provided the nonlinear solve is deterministic. This stands in contrast to finite differencing, which requires two nonlinear solves per design variable when using centered differences.

### 2.2.3 Parallelized derivative computation

Both AD and finite difference methods can be parallelized. While this does not reduce the number of computations required, it can improve the real time required for derivative computations, especially for large-scale problems. Despite these improvements, finite differencing still suffers from accuracy errors and AD should be used when it is available. In this work, we use the package PolyesterForwardDiff.jl (Mester et al., 2022) to parallelize the forward mode AD calculations.

---

[12]https://github.com/JuliaDiff/ReverseDiff.jl

[13]In this work we more fully describe the process of automating implicit analytic methods.

When applying PolyesterForwardDiff, the required number of processors can be determined from the relationship:

$$N_{proc} = \text{ceil}\left(\frac{N_{dv}}{N_{chunk}}\right), \tag{1}$$

where $N_{proc}$ is the number of processors, $N_{dv}$ is the number of design variables, and $N_{chunk}$ is the chunk size. This relationship
is the direct result of the chunking process described previously. While the parallel derivative computation cannot leverage more
processors to reduce the wall time, parallelization can still be exploited in other parts of the problem, such as running multiple
design load cases simultaneously.

### 2.2.4 Sparsity

As previously mentioned, Jacobians in many engineering problems exhibit sparsity. Sparse matrix formats store only the non-
zero entries along with their indices, rather than the entire matrix. The indices of the non-zero entries form the sparsity pattern
of the matrix. Sparse formats enable operations to be performed exclusively on non-zero entries, reducing computational cost.
Additionally, sparse storage drastically reduces memory requirements compared to dense matrix formats.

By leveraging the sparsity pattern of the Jacobian, the number of computations required to compute the derivatives can be
further reduced (Curtis et al., 1974). The process of leveraging the sparsity pattern is to 1) determine the sparsity pattern of the
Jacobian, 2) find the colorings of the Jacobian, and 3) compute the colored Jacobian. The first two steps can be done before
the optimization process begins, while the last step is done during the optimization process. We proceed by examining each of
these three steps in more detail.

Since the sparsity pattern of a Jacobian may have variations throughout the design space, we estimate the global sparsity
pattern by combining the sparsity pattern of randomly chosen designs. Sparsity patterns can be combined by summing the
absolute values of each of the elements of the Jacobian:

$$J_{global} = \sum_{i=1}^{N} |J_i|, \tag{2}$$

where $J_{global}$ is the global sparsity pattern, $J_i$ is the sparsity pattern of design $i$, and $N$ is the number of designs used to
determine the global sparsity pattern.

Coloring a Jacobian is the process of determining which columns (or rows) of the Jacobian are structurally orthogonal to
one another. Two structurally orthogonal columns are those that do not share any non-zero entries in the same row. When
computing the derivative, structurally orthogonal columns can be computed simultaneously without interference; effectively
reducing the number of columns that need to be computed. Depending on the sparsity pattern of the Jacobian, multiple columns
can be structurally orthogonal to one another. Columns that are structurally orthogonal are referred to as having the same color.
In this work, we use the Greedy D1 coloring algorithm in SparseDiffTools.jl[14] to determine the color of each of the columns of
the Jacobian of the constraints. We compute the colored Jacobian following an approach similar to SparseDiffTools.jl, but re-
implement it to enable efficient parallelization. Specifically, we use PolyesterForwardDiff.jl to compute the colored Jacobian.

---

[14]https://github.com/JuliaDiff/SparseDiffTools.jl

### 2.2.5 Constraint aggregation

While sparsity can be applied to both the columns and rows of a Jacobian, constraint aggregation offers a complementary approach for compressing dense rows. Among various aggregation techniques, the Kreisselmeier-Steinhauser method effec-

255 tively computes a smooth maximum over a constraint set. By formulating constraints as $g_i \leq 0$, multiple constraints can be combined into a single aggregate constraint. This aggregation provides minimal benefit for forward mode methods (forward mode AD, direct implicit differentiation, and finite differences) but substantially accelerates reverse mode approaches by reducing the number of required reverse passes. The trade-off is that constraint intersections are smoothed into a single curve, introducing a small offset from the true feasibility boundary that depends on the aggregation parameter. This parameter must

260 balance smoothness for differentiation against accuracy in representing the original constraints. Adaptive approaches, such as that proposed by Poon and Martins (2006), assist in selecting appropriate parameter values. We apply KS aggregation with reverse mode AD later in this work to compute constraint Jacobians in a single reverse pass.

### 2.3 Optimization formulation

This section describes the optimization problem formulation used in this work. Our objective function incorporates a subset of

265 the DLCs specified in IEC 61400-1 international standards for turbine certification. Following common practice in the literature (Zahle et al., 2016; Scott et al., 2022), we select a representative subset rather than the complete DLC suite, enabling focus on including the unsteady fatigue in the design optimization. Specifically, we include three cases: power generation (DLC 1.1), parked extreme loading (DLC 6.1), and dynamic turbulent inflow (DLC 1.2). This subset adequately demonstrates our model's capabilities and derivative computation methods, particularly the integration of fatigue analysis within the optimization loop.

270 We begin by describing the wind turbine, followed by detailed explanation of the three analysis cases.

We use the NREL 5 MW wind turbine blade as described in Resor (2013). The chord and twist are parameterized across the blade using Akima splines. The chord uses five control points, where the root control point is fixed at the original design. This is done to allow the blade to interface with the original hub. The twist angle is also parameterized using an Akima spline with four control points. The Akima splines starts after the cylindrical section because varying the aerodynamic twist of a

275 cylinder is not physically meaningful. To allow more flexibility in the peak chord of the blade, we also allowed the optimizer to vary the radial location of the second chord control point. These eight design variables account for the aerodynamic geometric parameters.

To enable structural optimization, we introduce scaling factors that control composite layer thicknesses across the blade. We divide each cross-section chordwise into five segments: leading-edge, leading-edge panel, spar cap, trailing-edge panel,

280 trailing-edge (see figure 2). Each segment contains multiple composite layers (e.g., unidirectional fibers, biaxial fabrics, core materials). A scaling factor $s_{ij}$ at radial station $r_i$ uniformly multiplies the thickness of every layer in segment $j$. The index $i$ ranges from 1 to 36 (the number of radial nodes) and $j$ ranges from one to five (corresponding to the five chordwise segments). Five Akima spline control points define the spanwise variation of each segment's scaling factors. Shear webs remain unscaled since the analyzed design load cases do not significantly stress these components. Note that in the original design, the number of

285  segments varies in cross-sections across the blade. The original design uses five to seven segments per cross-section depending on radial position. We consolidated cross-sections with more than five segments by merging adjacent regions and modifying the layup definitions accordingly. In cross-sections with trailing-edge webs to close the shell, we closed the shell with the trailing-edge segment.
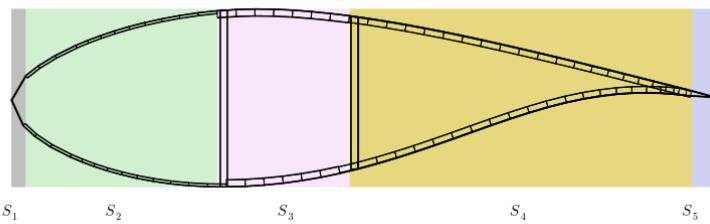
**Figure 2.** Cross-section of the NREL 5MW wind turbine blade showing the segments used for thickness scaling.

A minimum manufacturable thickness constraint was formulated by considering the thickness of each layup in a segment
290  and allowing the segment thicknesses to be scaled down until one layer reached the minimum manufacturable thickness. This constraint was applied to every cross-section, including between the control points. This added 25 design variables and 180 constraints to the optimization problem.

### 2.3.1 Power generation case

For the power generation case, we allow the optimizer to vary the tip speed ratio and the pitch of all 23 windspeeds. This
295  accounts for 24 additional design variables. In all, this problem has 57 design variables. While a comprehensive blade optimization would include additional variable types, we limit the design space to maintain focus on demonstrating the derivative computation methods.

The objective function is the cost of energy. The cost of energy is estimated using the cost model in WISDEM (Fingersh et al., 2006; Harrison and Jenkins, 1994; Malcolm and Hansen, 2006; Maples et al., 2010):

300
$$\text{COE} = \frac{\text{FCR} \cdot \text{ICC}}{AEP} + \frac{\text{OPEX}(1 - TR)}{AEP};$$
(3)

where FCR is the fixed charge rate (assumed to be 0.1158), ICC is the initial capital cost, AEP is the annual energy production, OPEX is the annual operating expenses (assumed to be 144 thousand USD per year), and TR is the tax rate (assumed to be 0.4). All of the assumed values in the cost model are based on values found in the implementation of the cost model in WISDEM[15]. The initial capital cost is calculated as

305
$$\text{ICC} = \text{BOS} + \text{TCC} + \text{WP},$$
(4)

---

[15]https://github.com/WISDEM/WISDEM

where BOS is the balance of station cost (assumed to be 2979 thousand USD), TCC is the turbine capital cost, and WP is the warranty premium (assumed to be 13 percent of the turbine capital cost). The turbine capital cost is a summation of simple emprical models for the major components of the wind turbine, including the rotor, nacelle, and tower.

These models are dependent on various aspects of the design such as the rotor diameter, the rated power, and the rated torque. The blade cost model is the most directly used in this optimization problem:

$$c_{\text{blade}} = 14.6 m_{\text{blade}}, \tag{5}$$

where $m_{\text{blade}}$ is the mass of the blade. The mass of each blade design is calculated by multiplying distributed weight (from the cross-section inertial data) of each blade element by the volume of the element (assuming that the density of each beam element is constant) and summing the result across the entire blade.

The AEP is calculated by integrating the power curve across a standard Weibull distribution with a mean wind speed of 6 m/s and a shape parameter of 2.0. Since the power curve can be calculated via steady analysis, we perform this calculation just using CCBlade.jl (not including the entire aero-elastic solver). The power is constrained to be no greater than the rated power, and the thrust is constrained to be no greater than 600 kN at every windspeed. The thrust constraint is a surrogate for modeling the structural response of the tower. We also constrain the maximum rpm at each wind speed to 12. The power generation case contributes a total of 47 constraints to the optimization problem.

### 2.3.2 Parked extreme loading case

The second case, the parked extreme loading case, assumes a 50-year return-period wind speed of 70 $ms^{-1}$. Because the loading is treated as steady, the analysis shifts from a time-domain simulation to a steady aerodynamic analysis, with the dynamic stall model omitted. The resulting aerodynamic loads are applied directly to a steady-state beam model solution. The blade is assumed to be in its worst-case orientation (horizontal) with a pitch of 0 degrees. The primary constraints for this case are buckling, ultimate strain, and tip deflection. The Bir buckling criterion is enforced at the top of each layer in the spar cap across all cross-sections, introducing 842 additional constraints. A conservative ultimate strain limit of 0.01 is applied to the top of every layer, in every segment, across all cross-sections, resulting in 10355 constraints. The safety factors $\gamma_f$ and $\gamma_m$ are 1.35 and 1.1 respectively . The tip deflection is constrained to be less than the original design's overhang distance of 5.019 m with a safety factor of 1.15.

### 2.3.3 Dynamic turbulent inflow case

The third case is a dynamic turbulent inflow case. The turbulent inflow field is generated as specified in the pre-process step. We use the IEC Kaimal spectrum to generate a turbulent wind field with a 10 m/s mean wind speed, a power law exponent of 0.2, and a surface roughness length of 0.03 m. The resulting wind field has an expected turbulence intensity of 0.16. Damage is extended from a simulation damage count to a lifetime damage by simple scaling:

$$D = D_{\text{sim}} \frac{t_{\text{lifetime}}}{t_{\text{simulation}}}. \tag{6}$$

The standard approach to calculating damage is by using a weighting of the lifetime damage from a range of different wind speeds. In this work, we calculate damage at only a single wind speed (the turbulent case at 10 m/s) to focus on demonstrating the capability to include fatigue damage in the optimization loop; full wind turbine design should consider the complete wind

340 speed distribution. Furthermore, the damage analysis is only performed at key locations along the blade. While the strain analysis is relatively quick (93 milliseconds), performing this analysis at every cross-section at every time step results in significant computational times. The damage was calculated at 5 locations along the blade: the root, the end of the cylinder section, and the three next highest damage locations in the original design (respectively 25.79% 39.14%, and 45.81% of the blade length).

345 The IEC standard requires that damage be calculated based on 600 seconds of simulation in response to turbulent inflow. Typically, in OpenFAST this is accomplished by running a 650 second simulation and removing the first 50 seconds of the simulation to allow for the system to finish responding to the initial conditions (what we call the transient response). Including a 650 second simulation analysis in the optimization loop would result in significant computational times. To determine the length of simulation for the objective function, we swept the length of the simulation and experimented with several different

350 cutoff times (see figure 3) when analyzing the inital design.
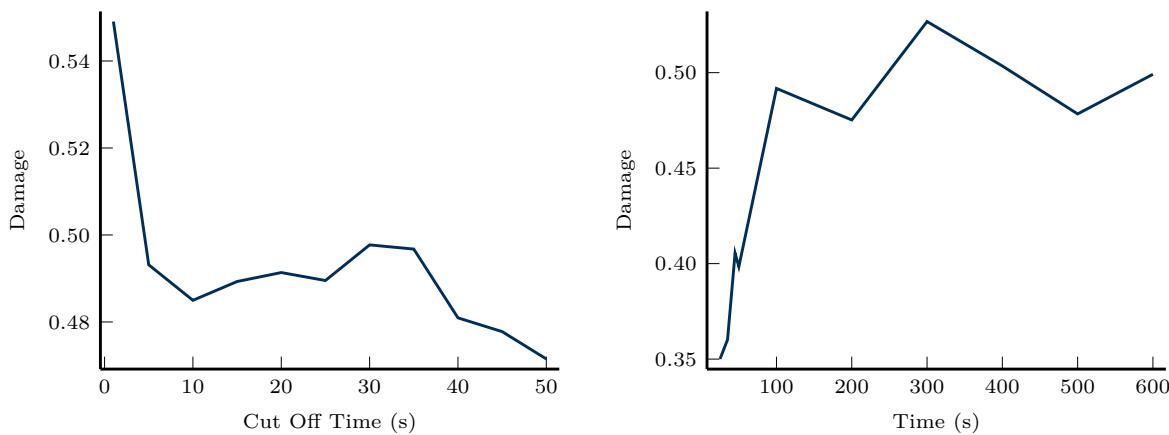
**Figure 3.** Left: varying the amount of time cut off from the beginning of the simulation; Right: varying the length of the simulation with the first 10 seconds removed.

The cutoff time was determined by running simulations of 600 seconds plus the portion removed at the beginning. Although the results did not demonstrate convergence at the standard 50-second cutoff, we selected a cutoff of 10 seconds for two reasons : 1) the relative percent difference in damage between all tested cutoff times and the standard 50-second cutoff was less than 3%, and 2) including a cutoff reduces the influence of the transient response on the damage calculation. Extended cutoff times

355 beyond 50 seconds showed no significant improvement in convergence, suggesting that the residual variation is inherent to the stochastic turbulent inflow rather than insufficient simulation duration. After fixing the cutoff at 10 seconds, we varied the total simulation length. A duration of 100 seconds was selected, as it was the shortest tested length yielding damage within

5% of the 600-second result (specifically, within 1.48%). The damage analysis introduces an additional 1811 constraints to the optimization problem. The damage metric exhibits high sensitivity to design changes, with values that can vary by multiple orders of magnitude. To improve optimizer performance, we apply a logarithmic transformation. Specifically, the damage is scaled by the safety factor ($\gamma_{\text{fatigue}} = 1.3$), logarithmically transformed, and then constrained to be less than or equal to zero.

Finally, the tip deflection is constrained to be less than 90% of the overhang distance at each time step throughout the same time period used for fatigue analysis. The simulation is run with a time step of 0.05 seconds, so the total number of time steps analyzed for fatigue and tip deflection is 1801. The size of the optimization problem is summarized in table 1. The optimization problem is formulated as follows:

minimize: COE

by varying: $\quad c_i \quad \theta_i \quad s_n \quad \beta_j \quad \lambda$

subject to:

$$
\begin{array}{c|c|c|c}
P_j < P_{\max} & T_j < T_{\max} & \Omega < \Omega_{\max} & \beta_j < \beta_{j+1} \\
B_{\text{ult}} < 1 & \dfrac{\varepsilon}{\gamma_f \gamma_m \varepsilon_{\text{ult}}} < 1 & \delta_s < d_{\text{overhang}} & \\
\delta_k < 0.9 d_{\text{overhang}} & \log(D/\gamma_{\text{fatigue}}) < 0 & s > s_{\min} &
\end{array}
\tag{7}
$$

where COE is the cost of energy, $c$ is the chord length, $\theta$ is the aerodynamic twist angle, $s$ are scaling factors of all of the layer thicknesses in a segment, $\beta$ is the pitching angle, $\lambda$ is the tip speed ratio, $B_{\text{ult}}$ is a buckling constraint, $\varepsilon_{\text{ult}}$ is a ultimate strain constraint, $\delta$ is a tip deflection constraint, $d_{\text{overhang}}$ is the maximum allowable overhang distance, $T_{\max}$ is the maximum thrust force, $\Omega$ is the rpm (and corresponding constraint), $D$ is the section damage constraint, and $s_{\min}$ is a minimum manufacturable thickness constraint. Note that the subscript $i$ refers to values that vary across the blade, $j$ refers to values that vary across wind speeds, and $k$ refers to values that vary across the time.

**Table 1.** Summary of optimization problem size. Note that the number of entries in the Jacobian is the number of non-zero entries.

| Design Variables | 57 | Jacobian Entries | 508416 |
|---|---|---|---|
| Constraints | 15080 | Simulation Time (s) | 100 |

## 3 Results and discussion

In this section, we demonstrate the capabilities of the approach. We begin with a verification case by comparing against OpenFAST. We then consider the performance of the differentiation approaches. Then we conclude with a discussion of the result of the optimization and how the differentiation approaches enable the inclusion of nonlinear fatigue analysis within the optimization loop.

### 3.1 Model verification

We verify WATT.jl by comparing its time-domain predictions with OpenFAST. The comparison consists of a 100-second
380 simulation at 0.01second time step with turbulent inflow averaging $10\ ms^{-1}$ and a rotor speed of 11.44 RPM. Other simulation
details are provided in the companion repository. The verification focuses on three key blade responses: tip aerodynamic
loading, tip deflection, and root flapwise bending moment. Figure 4 presents the first 50 seconds of these time histories, which
captures the response variations while maintaining visual clarity.
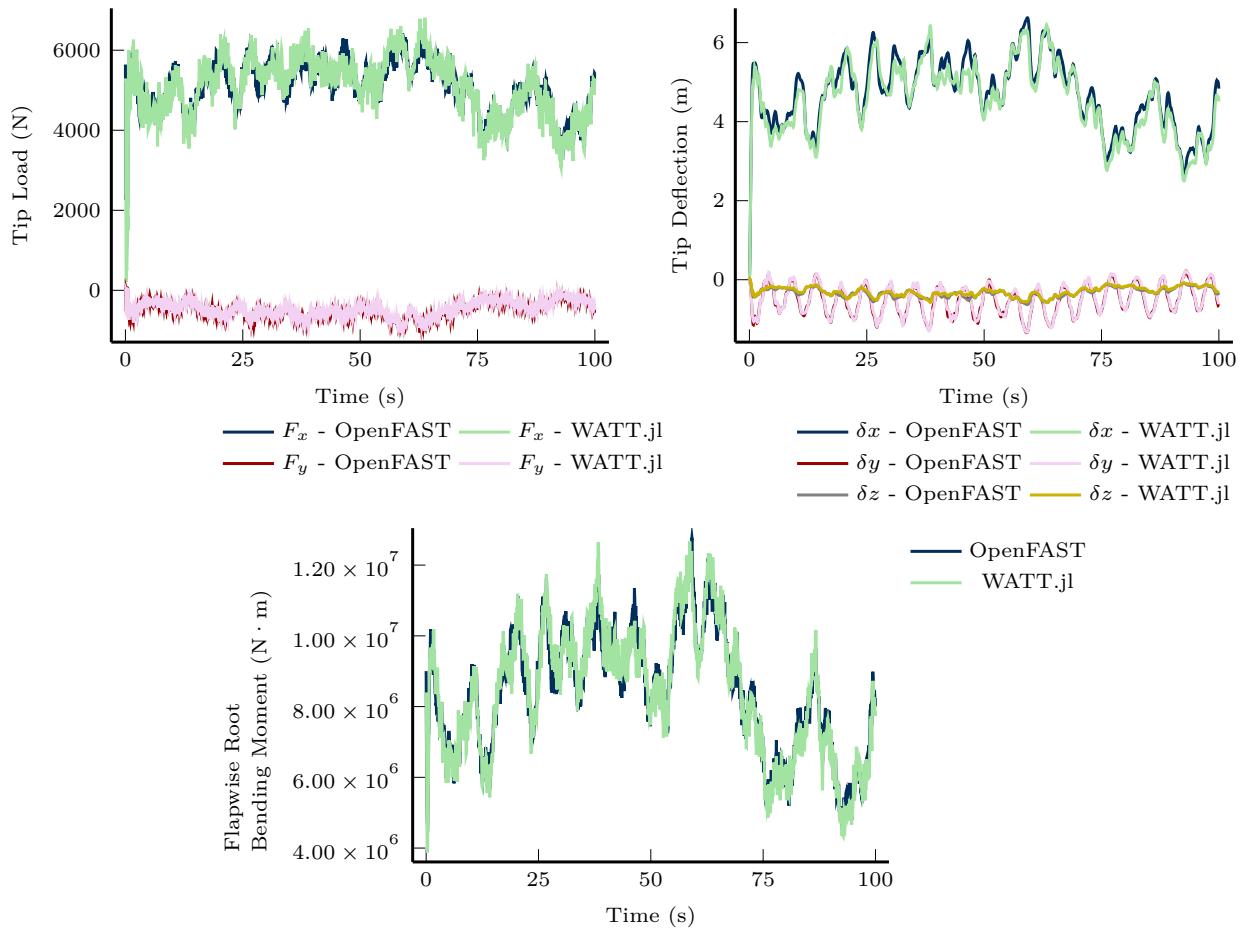
**Figure 4.** Comparison of the dynamic response to turbulent inflow of the NREL 5MW wind turbine blade using OpenFAST and WATT.jl.

In general, this figure demonstrates fairly good agreement between the two models. To quantify their performance, we
385 provide an error comparison. Because the lead–lagwise tip deflections frequently cross zero, we compare all blade response

using a maximum normalized relative error. We define the normalized error between two vector quantities as:

$$E_i(a,b) = 100 \frac{a_i - b_i}{\text{sign}(b_i) \, ||b||_\infty}, \tag{8}$$

where $E$, $a$, and $b$ are all vector quantities. The index $i$ refers to the vector element being evaluated and the infinity norm is applied to all elements of $b$. We use the normalized error for all comparisons for consistency. To provide interpretable results across time, we provide basic statistics on the normalized error, including the maximum, average, and standard deviation. The statistics of the aerostructural response are from 10 to 100 seconds to omit the transient response. Based on the results summarized in Table 2, we conclude that the two codes are in sufficient agreement for design optimization.

**Table 2.** Normalized error statistics between OpenFAST and WATT.jl.

|  | **Maximum** | **Average** | **Std Dev** |
|---|---|---|---|
| $F_{x_{\text{tip}}}$ | 9.22 % | 2.43 % | 1.74 % |
| $F_{y_{\text{tip}}}$ | 17.08 % | 3.82 % | 2.80 % |
| $M_{\text{flap}_{\text{root}}}$ | 10.51 % | 2.20 % | 1.60 % |
| $\delta_{x_{\text{tip}}}$ | 7.64 % | 2.64 % | 1.90% |
| $\delta_{y_{\text{tip}}}$ | 9.10 % | 2.27 % | 1.70 % |
| $\delta_{z_{\text{tip}}}$ | 15.24 % | 4.03 % | 2.96 % |

Before examining derivative performance, we compare the computational costs of OpenFAST and WATT.jl. In the analysis shown in Table 2 the codes perform similarly: OpenFAST averages 67.3 seconds while WATT.jl averages 65.2 seconds. The performance changes at larger time steps. When increased to 0.05 seconds, OpenFAST fails to converge, whereas WATT.jl completes the analysis in 13.1 seconds. WATT.jl's robust structural solver enables this larger time step without sacrificing accuracy—results remain within one percent of those obtained with the finer discretization. Simulations were conducted on a MacBook equipped with a 2.6 GHz six-core Intel i7 CPU and 32 GB of RAM.

### 3.2 Derivative performance

In this section, we examine how well the differentiation approaches from Sect. 2.2 perform when used in the optimization problem outlined in Sect. 2.3. We demonstrate the efficiency by first showing the derivative computation times of the different methods, then by showing how the methods scale with the number of design variables. We compare six different combinations of differentiation methods:

1. a centered finite difference,

2. forward mode AD,

3. forward mode AD with implicit differentiation around solvers,

4. reverse mode AD with implicit differentiation and constraint aggregation,

5. parallelized forward mode AD with implicit differentiation, and

6. parallelized sparse forward mode AD with implicit differentiation.

### 3.2.1 Derivative computation times

Table 3 shows the wall time and number of allocations of the six different differentiation techniques. We discuss four different features of table 3.

**Table 3.** Wall times to compute the Jacobian with a 100 second simulation for fatigue analysis.

|  | Central Difference | ForwardDiff | ForwardDiff w/ ImplicitAD | ReverseDiff w/ ImplicitAD & Constraint Aggregation | PolyesterForwardDiff w/ ImplicitAD | Sparse PolyesterForwardDiff w/ ImplicitAD |
|---|---|---|---|---|---|---|
| **Wall time (sec)** | 5521 | $\infty$ | 2797 | 2760 | 551 | 517 |
| **Allocations** | 3.29e10 | Stack Overflow | 3.83e9 | 1.21e10 | 6.06e9 | 3.79e9 |

First, we see that if we naively apply forward mode AD to the optimization problem the stack memory requirements quickly become untenable, specifically when propagating derivatives through the nonlinear solver inside of the time domain portion of the structural model. These memory issues are not inherent to forward mode AD with iterative solvers in general but arise from our specific implementation. The structural model we use, GXBeam.jl, is optimized to utilize stack memory for efficient structural analysis by storing variables on the stack rather than on the heap. ForwardDiff.jl tracks derivatives by augmenting each scalar variable with additional derivative information, effectively replacing each number with a small data structure containing both the value and its derivatives. When GXBeam.jl's many stack-allocated variables are augmented with this derivative information, the stack memory capacity is quickly exceeded. Applying implicit differentiation around the iterative solves circumvents this issue by computing derivatives without propagating them through the solver iterations, making AD feasible. GXBeam.jl was designed with implicit differentiation methods in mind so that it can benefit from both the speed of stack memory and the accuracy of AD.

Second, we see a variation of memory usage across the different methods. The number of allocations are not a reflection of how much information is held in memory at once, but how much the memory is being used. The reverse mode method offers a modest reduction in wall time compared to the forward mode method, at the cost of increased memory usage. However, the increased memory requirements are lower than what a centered finite difference demands. The parallelization of the derivatives provides the largest speedup, but requires more allocations than plain forward mode AD. This is because extra memory must be allocated to avoid data race conditions.

Third, there is opportunity for further speedup of the finite differencing by using more compute resources, specifically the number of threads used in the derivative computation. If memory is not a limiting factor, then theoretically the number of threads could be increased to the number of design variables times two (for centered finite differencing). This would reduce the wall time to be about the time it takes for a single function evaluation (about 48.43 seconds with 114 cores for this problem). Even though modern HPC resources could provide enough threads to make this cost tractable for small problems, they cannot

435 improve the limited accuracy of finite differencing. Higher-order finite difference schemes can improve derivative accuracy, but such methods are often impractical for large problems. Ultimately, the trade-off between computational resources and real-time performance is an engineering decision, where derivative accuracy remains a critical factor. Additionally, while higher order finite differencing schemes can improve the accuracy of the derivatives, they will always be limited by the truncation and round-off errors associated with finite differencing.

440 Fourth, the sparse method is the fastest method tested for the original optimization problem, over an order of magnitude faster than a centered finite difference. The sparse method is faster than the parallelized method, because the sparsity means the method is solving a smaller problem. We note that because the relative little cost to evaluate the power generation case, that we separate gradient evaluation from the Jacobian evaluation. Thus, we are able to color the Jacobian and compress the number of columns from 57 to 35 (a reduction of 22 columns or 38%). The gradient is evaluated using reverse mode differentiation,

445 which takes about 0.1 seconds. The Jacobian is evaluated using parallelized forward mode AD, which takes the rest of the time for the sparse differentiation method.

### 3.2.2 Derivative scaling

To demonstrate how the different methods scale with the number of design variables, we performed two types of sweeps: 1) varying the number of control points used in the chord and twist parameterizations, and 2) varying the number of wind speeds

450 analyzed in the power generation case. The first sweep changes the increases the number of dense columns in the Jacobian, while the second sweep increases the number of columns that can be colored (see Fig. 5). Each method was evaluated 10 times and the wall times were averaged. We compare methods by examining wall time increase per additional design variable. All methods exhibit approximately linear scaling ($O(n)$). Note that forward mode AD without implicit differentiation is omitted due to prohibitive stack memory requirements.
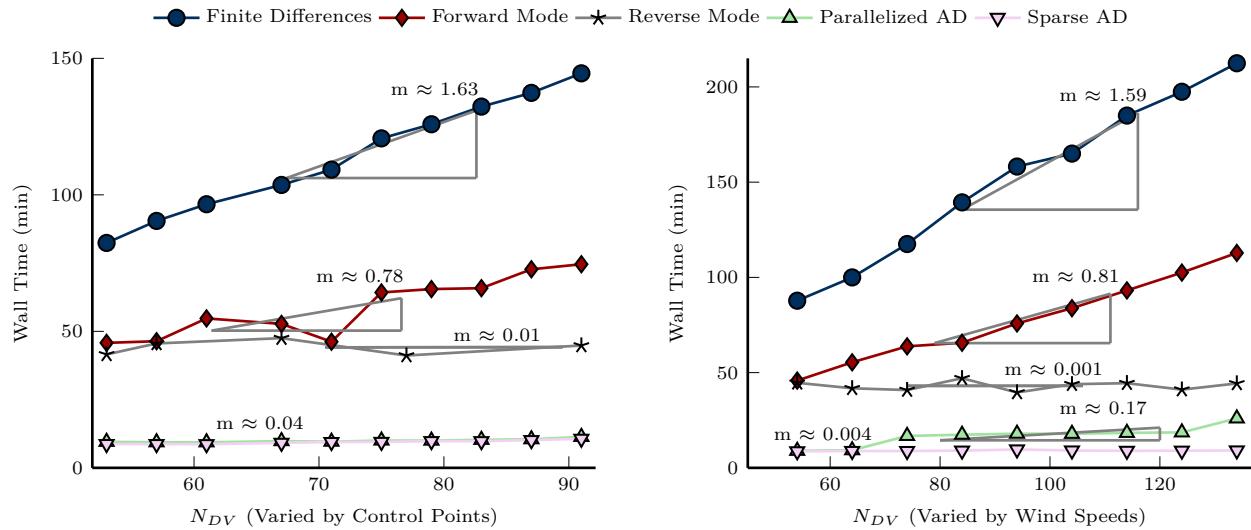
**Figure 5.** Scaling of derivative computation wall times with number of design variables. Left: varying the number of chord and twist control points; Right: varying the number of wind speeds analyzed in the power generation case.

The finite difference method had the worst scaling in both sweeps. The small difference in the slopes for the two sweeps, 1.63 and 1.59 for the control point sweep and the wind speed sweeps respectively, is likely due to the relatively small sample size used in timing. Parallelizing the finite difference would dramatically increase the scaling of the method, at the obvious increase of computational resources.

Forward mode AD with implicit differentiation demonstrates superior scaling compared to finite differences, with an average slope of 0.795. This improvement stems from the reduced number of mathematical operations required. Centered finite differences evaluate each operation $1 + 2N$ times: once for the primal and twice (forward/backward perturbation) for each of $N$ design variables. Forward mode AD evaluates each operation $1 + N$ times: once for the primal and once per derivative direction. In ForwardDiff.jl, the primal is recomputed for each chunk, so each operation is repeated $ceil\,(N/\text{chunk size}) + N$ times. With the application of implicit differentiation to the iterative solvers, the iterated operations are only evaluated for the primal solve and not for the derivative computation (although the residual is evaluated once for each derivative). Additionally, since the entire code is evaluated fewer times, there are improved memory savings from re-allocating vectors fewer times.

The reverse mode derivatives with implicit differentiation and constraint aggregation method has the best scaling in both sweep cases, 0.01 and 0.001. This is expected, because the fact that reverse mode differentiation scales with the number of outputs not the number of inputs. Since the constraints are aggregated into a single constraint, they can be calculated with a single pass. The differences in scaling between the two cases is likely due to the sample size and locations (i.e. if there were more samples, we would likely see a more consistent slope for the two methods). We do expect to see a slope (and some slight differences in the slope), because there is increased computational costs and memory overhead associated with evaluating a larger primal function. We note that in an optimization setting, instead of aggregating all constraints into a single constraint,

you may seek more accuracy in your constraints and so you may aggregate constraints by type. In this case, you can decrease
475 wall time by parallelizing constraint gradient computation.

The parallelized forward mode with implicit differentiation method has the next best scaling . The parallelized forward mode
AD method scales better than the forward mode AD method because as the number of columns increased, we increased the
number of CPU cores available. To demonstrate the effect of not using an optimal number of cores, we used a constant number
of cores in the second sweep. In the first sweep, the slope was 0.03, but the slope for the second sweep was 0.17. You can see
480 two sections where the wall time jumps up (when it jumps from 64 to 74 design variables and where it jumps from 123 to 134
design variables). Since the chunk size for is eight and the sweep is using eight CPU cores, we expect there to be jumps in the
wall time when the number of design variables surpasses multiples of 64 (i.e. 64, 128, 192...). What is happening here is that
every 64 design variables, the processors have to run another batch of derivatives in order to calculate all of the columns of the
Jacobian. You can avoid these jumps by using more processors or increasing the chunk size. Recall that increasing the chunk
485 size will increase memory requirements and may result in slower derivative computation times. Experimentation is required to
determine the optimal chunk size to processor ratio. Outside of the jumps, the scaling of the second sweep is 0.04 (calculated
by looking at just the wall times in-between the two jumps). The slope of 0.17, is still better than the plain forward mode
method with implicit differentiation because there are more CPU cores working on the problem.

Finally, we consider the sparse parallelized forward mode with implicit differentiation method for calculating derivatives.
490 Note that the slope goes from 0.04 for the first sweep to 0.004 in the second sweep. The slope improves greatly from the first
sweep to the second sweep because adding wind speeds adds structurally orthogonal columns to the Jacobian. For this method,
this means the only added cost is the added cost of calculating the primal values for the new analyses (in a similar fashion to
the reverse mode method). We note that for the second sweep, we also held the number of CPU cores constant, but because the
number of columns of the Jacobian did not effectively change, we do not see any jumps in the wall time.

495 The techniques presented here can benefit gradient-based optimization across various applications, though implementation
considerations differ depending on your starting point. For those developing new analysis tools, incorporating AD compatibility
from the outset lays the groundwork for efficient gradient-based optimization. This early consideration should be balanced
alongside other design decisions, such as ensuring numerical stability and convergence robustness. For existing codebases
where restructuring for AD compatibility may be impractical, techniques like parallelization and exploiting sparsity offer more
500 accessible performance improvements that do not require fundamental code changes. Tests are recommended as scaling is
problem dependent.

### 3.3 Optimization results

To assess the practical effectiveness of the parallelized sparse forward mode with implicit differentiation method, we present
two optimization studies. Our first study, a baseline optimization starting from the initial design, shows that the optimization
505 produces realistic results and improves upon finite difference approaches. The second study is a set of multiple optimizations
from random starting points that verify that the derivatives remain stable and consistent across different regions of the design

space. We primarily focus on the parallelized sparse forward mode with implicit differentiation (which we will occasionally refer to as the sparse method for brevity) because it was the fastest for the problem sizes we checked.

### 3.3.1 Baseline optimization

510 First, we consider the optimal design, then we discuss the performance of the baseline optimization in comparison to a finite difference approach. The optimization results are summarized in table 4 and the optimal design is shown in Fig. 6.

**Table 4.** Summary of optimization results. The number of active constraints are given at the initial ($\text{Con}_i$) and optimal ($\text{Con}_{opt}$) designs.

| Time (hours) | 33.8 | Iterations | 92 | Optimality | 5.3e-5 |
|---|---|---|---|---|---|
| Objective $\triangle$ | -11.41% | Mass $\triangle$ | - 11.90% | AEP $\triangle$ | 9.20% |
| # Active Con$_i$ | 279 | # Active Con$_{opt}$ | 94 | # F call | 123 |

We performed the baseline optimization in 33.8 hours using eight 2.3 GHz Intel Haswell CPU cores with seven GB of RAM from the Fulton HPC resources at Brigham Young University. The optimization converged in 92 iterations to an optimality of 5.3e-5, representing the infinity norm of the Lagrangian gradient. The objective function (COE) decreased by 11.41% through an 11.90% mass reduction and 9.20% increase in annual energy production (AEP). The initial design had 279 active constraints, while the optimal design had 94 active constraints. The optimizer required 123 function calls, including line search evaluations.
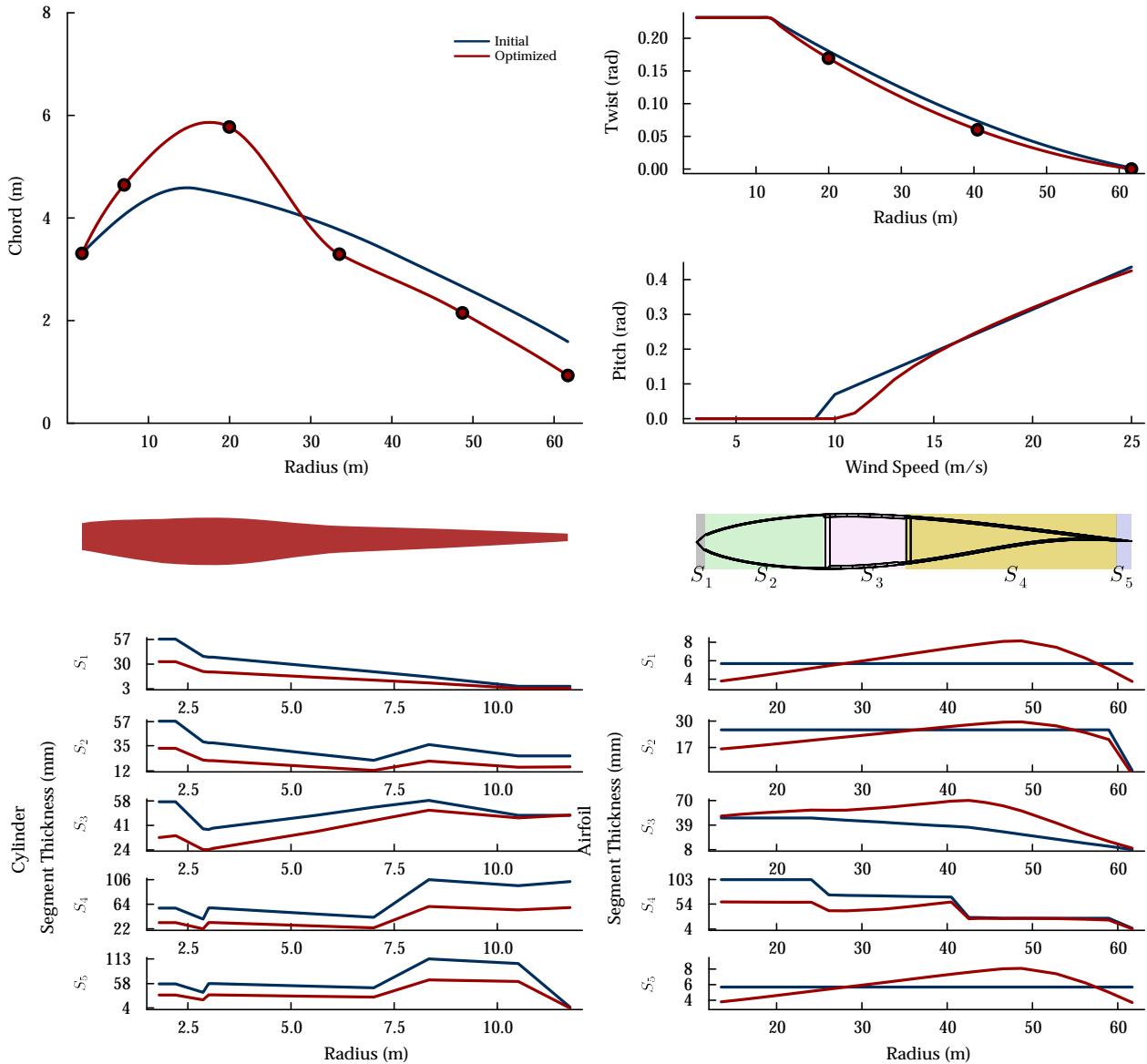
**Figure 6.** Reference optimization optimal design. Markers indicate the locations of the control points.

When considering the optimal design, we highlight three design trends. We note that since the focus of our work is the optimization efficiency and robustness, we only provide a brief discussion of these trends. First, in table 4, we see that the change in blade mass outweighs the change in annual energy production. Consistent with other studies (Yao et al., 2022), the material in the blade (and thus the mass) is the primary driver of decreasing the cost of energy. Second, in Fig. 6, we see a trade-off between the chord and the twist. We see that the chord is increased at the inboard region of the blade, whereas it is decreased near the outboard region. In the outboard regions the twist is also decreased. We see a similar trend in analytic aerodynamic-

only optimal solutions (see Burton et al. (2011) for an example of the analytic BEMT optimal solution). However, as seen in Fig. 7, the solution does not reach the aerodynamic-only solution because the objective function prioritizes cost reduction

525 rather than pure aerodynamic performance. Third, we see a trade-off between the chord and layer thicknesses. We see that in the inboard region, especially in the cylinder region, that the layer thicknesses shrink in response to the increased chord. The larger chord is better able to resist the loads applied there, so the thicknesses can be reduced there to decrease the mass.

Of the 94 active constraints at the optimal design, 22 were minimum manufacturable thickness constraints, 1 was the static deflection constraint, 7 were pitch monotonicity constraints, 61 were buckling constraints, and 3 were fatigue constraints. The

530 buckling constraints that were active in 3 sections near the hub ranging from 1.9 m to 2.1 m. The active fatigue constraints were all located in the trailing edge segment of the cross-section at the end of the cylinder section. This is typical (see Bortolotti et al. (2016)), as the trailing edge typically experiences the largest amplitude strain oscillation.
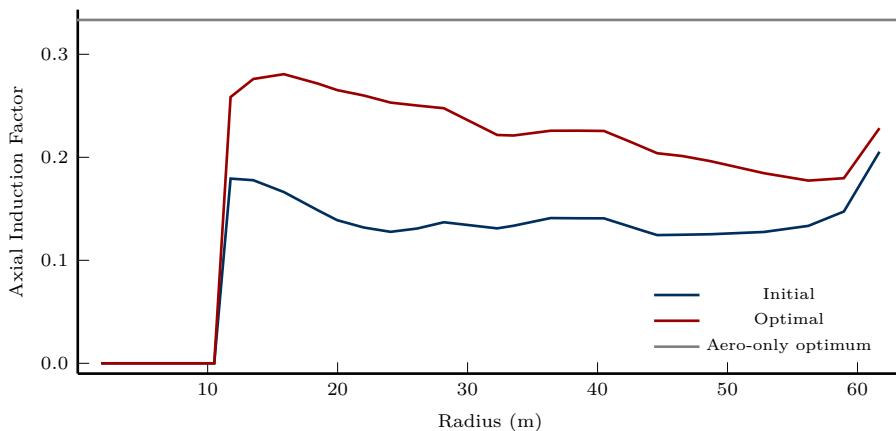
**Figure 7.** Axial induction factor distribution along the blade

Comparing our approach to finite difference highlights significant practical advantages. The AD-based optimization converged to an optimality of 5.3e-5 in 33.8 hours. From Sect. 3.2, we see that finite difference gradients are approximately 10x

535 slower than the parallelized sparse approach. Assuming that the optimizer would follow the same trajectory as when using the sparse approach, this implies similar convergence would require roughly 338 hours, or about 14 days. In practice, however, a centered finite difference optimization encounters numerical issues almost immediately. The inherent error in finite differenced derivatives exceeds the optimizer's default tolerances for gradient noise, causing premature termination. While careful tuning might enable some degree of convergence, reaching the same optimality achieved with accurate AD derivatives would

540 be unlikely. The baseline optimization establishes two key findings: the derivative methods yield physically realistic designs in an optimization, and they significantly outperform finite difference. Single-point convergence alone, however, cannot establish derivative stability throughout the design space.

### 3.3.2 Multi-start optimizations

The second study evaluates derivative stability across the design space through multi-start optimization. Consistent convergence
545 from diverse starting points indicates reliable derivatives, whereas chaotic or inaccurate gradients would prevent the optimizer
from reaching the same solution repeatedly. We generated 50 starting designs by randomly perturbing all initial design vari-
ables within ±10 percent of their baseline values. 29 optimizations converged smoothly, 4 experienced slow convergence near
tolerance thresholds, and 17 failed due to structural analysis instabilities. The following analysis addresses converged cases
first, followed by the non-converged cases.

550 The converged optimizations shown in Fig. 8 demonstrate remarkable consistency. The optimal designs cluster tightly to-
gether, with the optimal chord exhibiting a standard deviation of just 5 mm. Such tight convergence indicates that the derivatives
are both accurate and consistent across a sizeable portion of the design space that the optimizer explored. While a stricter op-
timality tolerance could reduce this variability, doing so could substantially increase iteration counts and might necessitate
stopping and rescaling design variables. If the optimizations were conducted with noisy or inaccurate gradients, we would
555 expect to see a wider spread in the final designs.
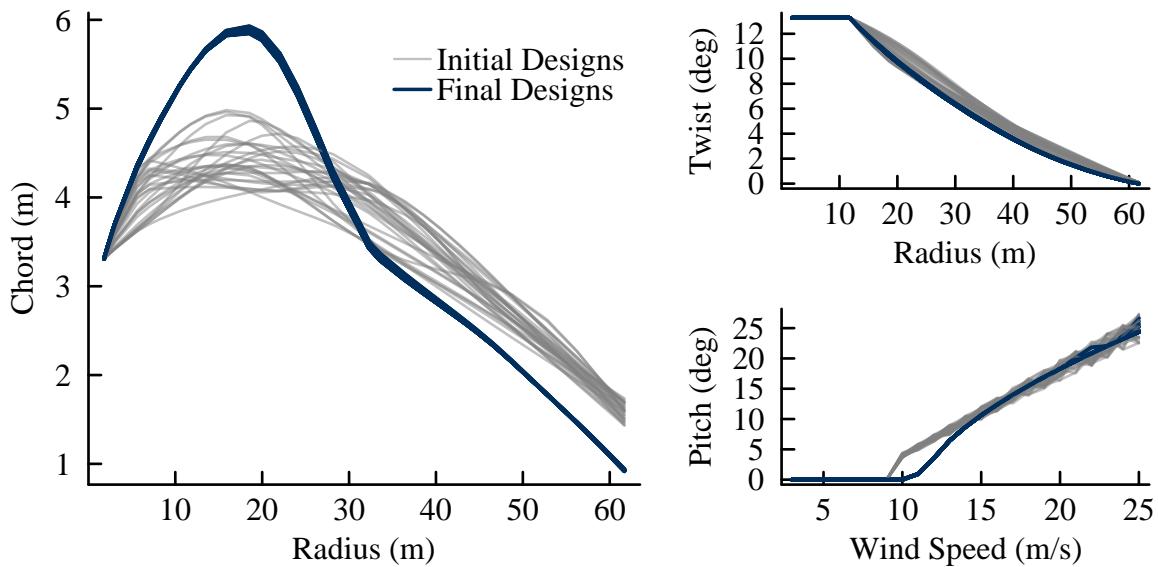
**Figure 8.** Optimizations from different starting points converging to one solution.

17 randomly generated starting points failed to converge due to numerical instabilities in the unsteady aero-structural analy-
sis, particularly within the structural solver. These failures typically stem from highly elastic blade designs that create positive
feedback loops with the aerodynamic model. A finer temporal mesh generally resolves such instabilities. The four designs
struggling to meet convergence criteria occupied a saddle point adjacent to this failure region. These failure modes are charac-
560 teristic of the underlying physics and solver performance rather than deficiencies in the derivative computation, the gradients

accurately reflect the challenging landscape near infeasible designs. The 29 converged cases validate derivative consistency in the feasible and near-optimal regions.

## 4 Conclusions

Gradient-based optimization is an effective approach to creating wind turbine blade designs that not only meet performance requirements but also minimize cost. The cost of calculating derivatives has historically made it infeasible to conduct nonlinear time-domain simulations to include fatigue analysis within the optimization loop. We use efficient derivative computation methods to allow for the inclusion of fatigue analysis within the optimization loop.

We evaluate six differentiation approaches: finite differences, forward mode AD, forward mode AD with implicit differentiation, reverse mode AD with implicit differentiation, parallelized forward mode AD with implicit differentiation, and sparse parallelized forward mode AD with implicit differentiation. Using a proof-of-concept optimization problem, we compare computational costs and examine how each approach scales with problem size. The sparse parallelized forward mode approach proves over an order of magnitude faster than traditional finite differences while delivering superior accuracy. This approach also exhibits favorable scaling characteristics as problem size increases.

To allow for AD compatibility, we present a new aeroelastic solver: WATT.jl. Similar to OpenFAST, the solver incorporates blade element momentum theory, the Beddoes-Leishman dynamic stall model, and a geometrically exact beam theory structural solver in a fully nonlinear two-way coupled partitioned model. The analysis method is compared to OpenFAST and is within an average of 2-4% of the time domain loads and deflections. Additionally, classical laminate theory, the rainflow counting algorithm, and linear damage accumulation methods are integrated into the analysis framework.

The derivatives are used in a representative optimization problem to demonstrate feasibility. The optimization converges to an optimality of 5.3e-5 in 33.8 hours using only eight CPU cores and 7GB of RAM. The cost of energy objective is reduced 11.41% by reducing the blade mass 11.90% and increasing the AEP 9.20%. Future work should include the exploration of additional design variables and constraints to better capture the complexities of the full wind turbine design problem. Because of the few resources required, it should be straightforward to scale up the problem to include more design load cases and a more complex design space.

*Code availability.* Scripts to reproduce results can be found at https://github.com/byuflowlab/Cardoza2025_Efficient_aeroelastic_wind_gradients.git

*Author contributions.* AN conceived the approach and developed the CLT code. AC developed the dynamic stall code, the coupling code, the optimization framework with supervision from AN. AC also prepared the manuscript with feedback from AN.

*Competing interests.* The authors declare that they have no conflict of interest.

590   *Acknowledgements.* The authors would like to thank Jeff Allen, Jonathan Maack, Pietro Bortolotti, Garret Barter, Denis-Gabriel Caprace, and Taylor McDonnell for their advice and input on this work. This work was authored in part by the National Laboratory of the Rockies for the U.S. Department of Energy (DOE), operated under Contract No. DE-AC36-08GO28308. The views expressed in the article do not necessarily represent the views of the DOE or the U.S. Government. The U.S. Government retains and the publisher, by accepting the article for publication, acknowledges that the U.S. Government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish
595   or reproduce the published form of this work, or allow others to do so, for U.S. Government purposes ChatGPT and Claude were used for assistance in copy-editing. Copilot was used for code completion.

*Financial support.* This work was authored under funding support provided by the National Laboratory of the Rockies subcontract ASIQ 2023005151. Original funding from the U.S. Department of Energy Office of Science, Advanced Scientific Computing Research, Exploratory Research 595 for Extreme-Scale Science program under DE-FOA-0002717.

# References

Akima, H.: A New Method of Interpolation and Smooth Curve Fitting Based on Local Procedures, Journal of the Association for Computing Machinery, 17, 589–602, https://doi.org/10.1145/321607.321609, 1970.

Allen, J., Young, E., Bortolotti, P., King, R., and Barter, G.: Blade planform design optimization to enhance turbine wake control, Journal of Wind Energy, https://doi.org/10.1002/we.2699, 2020.

American Society for Testing and Materials: Standard Practices for Cycle Counting in Fatigue Analysis, Tech. rep., 2017.

Batay, S., Kamalov, B., Zhangaskanov, D., Zhao, Y., Wie, D., Zhou, T., and Su, X.: Adjoint-Based High Fidelity Concurrent Aerodynamic Design Optimization of Wind Turbine, Fluids, https://doi.org/10.3390/fluids8030085, 2023.

Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. B.: Julia: A fresh approach to numerical computing, SIAM review, 59, 65–98, https://doi.org/10.1137/141000671, 2017.

Bir, G. S.: Computerized Method for Preliminary Structural Design of Composite Wind Turbine Blades, Journal of Solar Energy Engineering, 123, 372–381, https://doi.org/10.1115/1.1413217, 2001.

Bortolotti, P., Botasso, C. L., and Croce, A.: Combined preliminary-detailed design of wind turbines, Wind Energy Science, pp. 71–88, https://doi.org/10.5194/wes-1-71-2016, 2016.

Bortolotti, P., Botasso, C. L., Croce, A., and Sartori, L.: Integration of multiple passive load mitigation technologies by automated design optimization - The case study of a medium-size onshore wind turbine, Wind Energy, 22, 65–79, https://doi.org/10.1002/we.2270, 2018.

Bottasso, C., Bortolotti, P., Croce, A., and Gualdoni, F.: Integrated aero-structural optimization of wind turbines, Multibody Syst Dyn, 38, 317–344, https://doi.org/10.1007/s11044-015-9488-1, 2016.

Brent, R. P.: Algorithms for Minimization Without Derivatives, Dover Publications, ISBN 0-486-41998-3, 2013.

Burton, T., Jenkins, N., Sharpe, D., and Bossanyi, E.: Wind Energy Handbook, John Wiley & Sons, Ltd, 2 edn., https://doi.org/10.1002/9781119992714, 2011.

Curtis, A. R., Powell, M. J. D., and Reid, J. K.: On the estimation of sparse Jacobian matrices, IMA Journal of Applied Mathematics, 13, 117–119, https://doi.org/10.1093/imamat/13.1.117, 1974.

Dhert, T., Ashuri, T., and Martins, J. R. R. A.: Aerodynamic shape optimization of wind turbine blades using a Reynolds-averaged Navier-Stokes model and an adjoint method, Wind Energy Science, 20, 909–926, https://doi.org/10.1002/we.2070, 2017.

Døssing, M., Madsen, H. A., and Bak, C.: Aerodynamic optimization of wind turbine rotors using a blade element momentum method with corrections for wake rotation and expansion, Wind Energy Science, 15, 563–574, https://doi.org/10.1002/we.487, 2012.

Fingersh, L., Hand, M., and Laxson, A.: Wind Turbine Design Cost and Scaling Model, Tech. rep., National Renewable Energy Laboratory, https://doi.org/10.2172/897434, 2006.

Frontin, C., Bortolotti, P., Branlard, E., Martinez, L., Vijayakumar, G., and Barter, G.: Design space exploration for novel reduced-vortex turbine rotors using free wake vortex methods, Rotor Design and Manufacturing, NAWEA WindTech 2023, 2023.

Gill, P. E., Murray, W., and Saunders, M. A.: SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization, SIAM Review, 47, 99–131, https://doi.org/10.1137/S0036144504446096, 2005.

Harrison, R. and Jenkins, G.: Cost modelling of horizontal axis wind turbines, Tech. Rep. ETSU-W-34/00170/REP, University of Sunderland, 1994.

Hermansen, S. M., Macquart, T., and Lund, E.: Gradient-based structural optimization of a wind turbine blade root section including high-cycle fatigue constraints, Engineering Optimization, https://doi.org/10.1080/0305215X.2024.2428678, 2025.

Ingersoll, B.: Efficient Incorporation of Fatigue Damage Constraints in Wind Turbine Blade Optimization, Wind Energy, 23, 1063–1076, https://doi.org/10.1002/we.2473, 2020.

International Electrotechnical Commission: Wind energy generation systems - Part 1: Design requirements, IEC 61400-1:2019, https://webstore.iec.ch/en/publication/29360, 2019.

Jonkman, J. M., Hayman, G. J., Jonkman, B. J., Damiani, R. R., and Murray, R. E.: AeroDyn v15 User's Guide and Theory Manual, National Laboratory of the Rockies, National Laboratory of the Rockies, Golden, CO, draft edn., 2017.

Kenway, G. and Martins, J. R. R. A.: Aerostructural shape optimization of wind turbine blades considering site-specific winds, AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, 12, https://doi.org/10.2514/6.2008-6025, 2008.

Kollár, L. P. and Springer, G. S.: Mechanics of Composite Structures, Cambridge University Press, https://doi.org/10.1017/CBO9780511547140, 2009.

Kwon, H. I., Yi, S., Choi, S., and Kim, K.: Design of efficient propellers using variable-fidelity aerodynamic analysis and multilevel optimization, Journal of Propulsion and Power, 31, 1057–1072, https://doi.org/10.2514/1.B3509, 2015.

Madsen, M. H. A., Zahle, F., Sorensen, N. N., and Martins, J. R. R. A.: Multipoint high-fidelity CFD-based aerodynamic shape optimization of a 10MW wind turbine, Wind Energy Science, 4, 163–192, https://doi.org/10.5194/wes-4-163-2019, 2019.

Madsen, M. H. A., Zahle, F., Horcas, S. G., Barlas, T. K., and Sørensen, N. N.: CFD-based curved tip shape design for wind turbine blades, Wind Energy Science, 7, 1471–1501, https://doi.org/10.5194/wes-7-1471-2022, 2022.

Malcolm, D. J. and Hansen, A. C.: WindPACT Turbine Rotor Design Study: June 2000–June 2002 (Revised), Tech. rep., National Renewable Energy Laboratory, Golden, CO., https://doi.org/10.2172/15000964, 2006.

Mangano, M., He, S., Liao, Y., Caprace, D.-G., Ning, A., and Martins, J. R. R. A.: Wind Turbine Rotor Design Using High-Fidelity Aerostructural Optimization, AIAA Journal, 63, 3493–3513, https://doi.org/10.2514/1.J064556, 2025.

Maples, B., Hand, M., and Musial, W.: Comparative Assessment of Direct Drive High Temperature Superconducting Generators in Multi-Megawatt Class Wind Turbines, Tech. Rep. NREL/TP-5000-49086, National Renewable Energy Laboratory, 2010.

Martins, J. R. R. A. and Ning, A.: Engineering Design Optimization, Cambridge University Press, https://doi.org/10.1017/9781108980647, 2021.

McDonnell, T. and Ning, A.: Geometrically exact beam theory for gradient-based optimization, Computers and Structures, 298, 107–373, https://doi.org/10.1016/j.compstruc.2024.107373, 2024.

McWilliam, M. K., Barlas, T. K., Madsen, H. A., and Zahle, F.: Aero-elastic wind turbine design with active flaps for AEP maximization, Wind Energy Science, 3, 231–241, https://doi.org/10.5194/wes-3-231-2018, 2018a.

McWilliam, M. K., Zahle, F., Dicholkar, A., Verelst, D., and Kim, T.: Optimal Aero-Elastic Design of a Rotor with Bend-Twist Coupling, Journal of Physics: Conference Series, IOP Conference Series, 1037, 042 009, https://doi.org/10.1088/1742-6596/1037/4/042009, 2018b.

Mester, R., Landeros, A., Rackauckas, C., and Lange, K.: Differential methods for assessing sensitivity in biological models, PLoS Computational Biology, 18, e1009–598, https://doi.org/10.1371/journal.pcbi.1009598, 2022.

Naumann, U.: The art of differentiating computer programs: an introduction to algorithmic differentiation, SIAM, https://doi.org/10.1137/1.9781611972078, 2011.

Ning, A.: Using Blade Element Momentum Methods with Gradient-Based Design Optimization, Structural and Multidisciplinary Optimization, 64, 994–1014, https://doi.org/10.1007/s00158-021-02883-6, 2021.

Ning, A. and McDonnell, T.: Automating Steady and Unsteady Adjoints: Efficiently Utilizing Implicit and Algorithmic Differentiation, arXiv, https://doi.org/10.48550/arXiv.2306.15243, 2023.

675 Ning, A. and Petch, D.: Integrated design of downwind land-based wind turbines using analytic gradients, Wind Energy, 19, 2137–2152, https://doi.org/10.1002/we.1972, 2016.

Pavese, C., Botasso, C. L., Zahle, F., and Kim, T.: Aeroelastic multidisciplinary design optimization of a swept wind turbine blade, Wind Energy, 20, 1941–1953, https://doi.org/10.1002/we.2131, 2017.

Poon, N. M. K. and Martins, J. R. R. A.: An adaptive approach to constraint aggregation using adjoint sensitivity analysis, Structural and
680 Multidisciplinary Optimization, 34, 61–73, https://doi.org/10.1007/s00158-006-0061-7, 2006.

Resor, B. R.: Definition of a 5MW/61.5m Wind Turbine Blade Reference Model, Tech. rep., Sandia National Laboratories, Albuquerque, New Mexico, https://doi.org/10.2172/1095962, 2013.

Revels, J., Lubin, M., and Papamarkou, T.: Forward-Mode Automatic Differentiation in Julia, arXiv, https://doi.org/10.48550/arXiv.1607.07892, 2016.

685 Scott, S. J., Greaves, P., Macquart, T., and Pirrera, A.: Comparison of blade optimisation strategies for the IEA 15MW reference turbine, Journal of Physics: Conference Series, IOP Conference Series, 2265, 032 029, https://doi.org/10.1088/1742-6596/2265/3/032029, 2022.

Serafeim, G., Manolas, D., Riziotis, V., and Chaviaropoulos, P.: Multidisciplinary aeroelastic optimization of a 10MW-scale wind turbine rotor targeting to reduced LCoE, Journal of Physics: Conference Series, IOP Conference Series, 2265, 04 051, https://doi.org/10.1088/1742-6596/2265/4/042051, 2022.

690 Varela, B. and Ning, A.: Sparsity Applications for Gradient-Based Optimization of Wind Farms, (in review), 2024.

Yao, S., Chetan, M., Griffith, D. T., Mendoza, A. S. E., Selig, M. S., Martin, D., Kianbakht, S., Johnson, K., and Loth, E.: Aero-structural design and optimization of 50 MW wind turbine with over 250-m blades, Wind Engineering, 46, 273–295, https://doi.org/10.1177/0309524X211027355, 2022.

Zahle, F., Tibaldi, C., Pavese, C., McWilliam, M. K., Blasques, J. P. A. A., and Hansen, M. H.: Design of an Aeroelastically Tailored 10
695 MW Wind Turbine Rotor, Journal of Physics: Conference Series, IOP Conference Series, 753, 062 008, https://doi.org/10.1088/1742-6596/753/6/062008, 2016.