Handling Iterative Solvers in an Algorithmic Differentiation Framework using Implicit Methods

Jeffery M. Allen[a], Olga Doronina[a], Jon Maack[a], Ethan Young[a], Andrew Ning[b], Adam Cardoza[b], Eric Green[b]

[a]Computational Science Center, National Renewable Energy Laboratory, Golden, CO 80401, USA
[b]Mechanical Engineering, Brigham Young University, Provo, UT 84602, USA

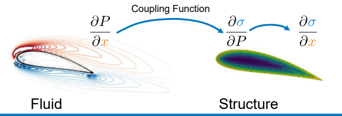
Abstract

Differentiable programming is a powerful concept as it enables the seemly **propagation of gradients through functions, algorithms, and/or whole physics simulations**. These gradients are useful for a wide variety of applications, including sensitivity studies and machine learning, but one of particular interest is optimization. Gradient-based optimization, enabled through **automatic/algorithmic differentiation (AD)**, can be used on predictive physical models to efficiently optimize a set of design variables. AD methods are a particularly promising approach to complex physics simulations because they can be shown to scale well with an increasing number of design variables; however, care must be taken when coupling between different models or different states of a single model.

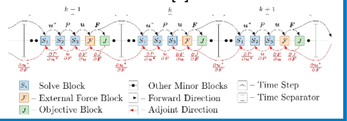
Motivation

This project fills a gap in simulation and optimization research by developing solutions to (1) propagate AD gradients between very disparate simulations and (2) maintain problem tractability even when long sequences of such operations are necessary (e.g., time stepping or iterative solves)

1. **Propagating AD Gradients**
   For coupled approaches, we are developing a framework to pass AD gradients between different algorithms with different structures, e.g., a partitioned fluid-structure interaction simulation using the concept of algorithmic differentiation and unsteady adjoints.


Coupling Function
$\frac{\partial P}{\partial x}$
$\frac{\partial \sigma}{\partial P}$
$\frac{\partial \sigma}{\partial x}$
Fluid
Structure

2. **Managing Iterative Simulations**
   Iterative solution methods pose a unique, related challenge in that many such couplings must be recorded, requiring prohibitive amounts of memory—custom rules to re-use repeated operations can eliminate this barrier [1].



Approach

If AD is used to natively compute gradients with respect to a solve, the resulting computational graph will be unnecessarily complex. Instead, the **solve is treated as a special node** on the graph, which relies on the fact that the solution of a system does not depend on the solver path. This results in **trading a complex computational graph with an additional linear solve**.

1. Define residual

$$r(x,u(x)) = 0$$

2. Apply chain rule:

$$\frac{dr}{dx} = \frac{\partial r}{\partial x} + \frac{\partial r}{\partial u}\boxed{\frac{\partial u}{\partial x}} \leftarrow \text{We want the action of this Jacobian!}$$

4. Since $r = 0$ at the solution, $\frac{dr}{dx} = 0$, yielding:

$$\text{The tangent system} \rightarrow \frac{\partial r}{\partial u}\frac{\partial u}{\partial x} = -\frac{\partial r}{\partial x}$$

3. Set $\frac{\partial r}{\partial u} = A$, $\frac{\partial u}{\partial x} = J$, and $\frac{\partial r}{\partial x} = B$:

$$\underbrace{A\, J = -B}_{\text{These can be found using Forward or Reverse AD}}$$

Forward Mode

In Forward mode, we have access to the dual input, $\dot{x}$, which is the local derivative of the special node's input with respect to the design variables. The goal of the special node is to return $\dot{u}$, which can be computed using:

$$\dot{u} = \frac{du}{dx}\dot{x} = J\dot{x}.$$

Multiplying both sides of the tangent system by $\dot{x}$, yields:

$$A J\dot{x} = -B\dot{x},$$
$$A\dot{u} = -B\dot{x}.$$

The RHS, $-B\dot{x} = g$, can be computed as a Jacobian vector product to get:

$$A\dot{u} = g$$

Solving this linear system for $\dot{u}$ allows the special node to output its gradient information.

Reverse Mode

In Reverse mode, we have access to $\bar{u}$, the local derivative of design variables with respect to the special node's outputs, u. The goal of the special node is to compute $\bar{x}$ using:

$$\bar{x} = J^T\bar{u}.$$

First, transpose the tangent system:

$$J^T A^T = -B^T.$$

Next, multiply by an unknown vector, z:

$$J^T A^T z = -B^T z.$$

Solving the linear system $A^T z = \bar{u}$, for z allows for the following substitutions:

$$J^T A^T z = -B^T z,$$
$$J^T\bar{u} = -B^T z,$$
$$\bar{x} = -B^T z.$$

Now, $\bar{x}$ can be computed using the Jacobian vector product, $-B^T z$.

Software Information:

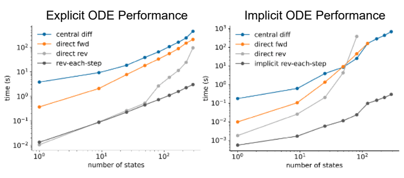
Source: https://github.com/byuflowlab/ImplicitAD.jl
Docs: https://docs.juliahub.com/ImplicitAD/KnMWN/0.2.2/tutorial/

Preliminary Results

We are currently focused on verifying the scaling and timing performance of the **Implicit AD** approach relative to commonly-used alternatives—including finite differences and direct forward/reverse mode—for benchmark problems and test cases like those shown below.

Explicit/Implicit ODEs Benchmark

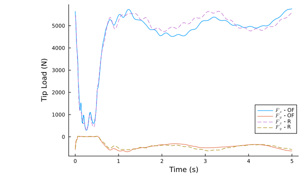
The benchmark ODE problem is a simple **2D heat transfer** analysis on a thin plate with convection and radiation. For the explicit case, the **Tsitouras 5/4 Runge-Kutta** method is used, and the implicit case uses the **Backward Euler** method. In both cases, the number of inputs (states) is much greater than the number of outputs, so reverse mode AD should perform better. For the full details of the problem setup and results, see [2].



Time to compute gradient for the explicit (left) and implicit (right) ODE problems as a function of the number of states for finite differencing, forward AD, reverse AD, and implicit reverse AD applied over each time step.

Unsteady Loads for Turbine Blades

One of the preliminary test cases involves computing **load forces on a turbine blade** or, more specifically, the gradients of these forces for use in optimizations of turbine blades. We use Rotor.jl (built in Julia) to compute the gradients and have compared the resulting forces with OpenFAST. Using reverse unsteady adjoint method takes only 24.8 seconds, which is **25 times faster** than finite differences.



Comparing the loading forces on the tip of the blade computed using OpenFAST (OF) and our new method using Rotors.jl (R).

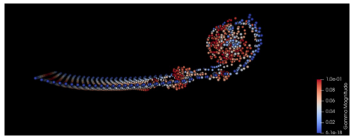
A meshed turbine blade used for computing loads.

Simulation Time	Time Steps	Finite Diff	Implicit Rev
14.0 s	250	621.7 s	24.8 s

A table showing the performance of computing these derivatives.

Vortex Particle Method

The **vortex particle method (VPM)** is a mesh-free, Lagrangian method in which particles representing vorticity at a point are used to solve the Navier-Stokes equations in their velocity-vorticity form. We use the **fast multipole method (FMM)** to calculate particle-particle interactions and their associated derivatives in a scalable manner as the number of particles increases.



Snapshot of particles released from the trailing edge of a blade after 40 time steps; as the simulation proceeds and more particles are introduced, enabling efficient inter-particle calculations becomes increasingly important.

Particle Count	Fast Multipole Method		Forward Mode		Reverse Mode		Implicit Reverse Mode	
10	0.002 s	0.56 MB	0.003 s	0.84 MB	1.174 s	577.37 MB	0.030 s	13.52 MB
100	0.005 s	2.95 MB	0.010 s	3.54 MB	4.390 s	1,329.00 MB	0.130 s	67.61 MB
1,000	0.110 s	34.29 MB	0.240 s	43.93 MB	Memory Overflow		2.122 s	878.18 MB
10,000	0.257 s	255.00 MB	0.425 s	279.76 MB	Memory Overflow		30.820 s	8,398.00 MB

Computation time and memory usage of the fast multipole method and the gradient calculation using various AD methods.

Conclusion

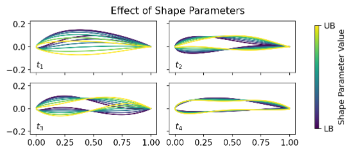
The **Implicit AD package significantly outperforms other state-of-the-art methods** for calculating derivatives across a variety of different use cases. These performance increases are evident for benchmark examples of root finding, ODE solvers, and an application for vortex particle methods over a wide range of degrees of freedom. As we begin to apply these tools to domain problems in earnest, we will focus on **coupling fluid and structural solvers** to demonstrate the ability to efficiently and accurately compose complicated derivatives across models and continue to advance our **applications to stochastic models** to further refine how similar computational steps can be aggregated and accessed for efficient memory usage.

Future Work

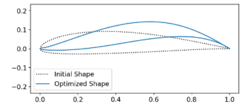
We are currently working on applying these computational and theoretical advances to more domain-specific problems from aerodynamics and physics. This demonstrates the broad applicability of the Implicit AD tools while motivating further developments.

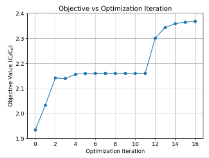
Airfoil Optimization

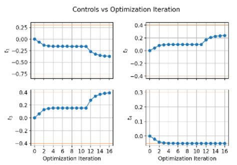
In this problem, we seek to **maximize the simulated lift-to-drag ratio** [3] by controlling airfoil shape. In this preliminary study, we restrict ourselves to modifications of **4 shape parameters** [4], which capture the principally important modes of airfoil shape change, that is:



Maximize	Lift-to-drag ratio (C_L/C_D)
with respect to	$\mathbf{t} = [t_1, t_2, t_3, t_4]$
subject to	$\mathbf{LB} \le \mathbf{t} \le \mathbf{UB}$ constraints







Stochastic Simulations

In this problem, we seek to design a tower with minimal cost. The tower is subject to stochastic forces due to wind, and we want it to remain close to neutral while large magnitude displacement and momentum is heavily penalized.

$$dz = (Az + f_0)dt + BdW$$

$$z = \begin{pmatrix} q \\ p \end{pmatrix}, \quad A = \begin{pmatrix} 0 & 1/m \\ -k & -\gamma/m \end{pmatrix}$$

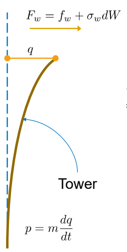
$$f_0 = \begin{pmatrix} 0 \\ f_w \end{pmatrix}, \quad B = \begin{pmatrix} 0 \\ \sigma_w \end{pmatrix}$$

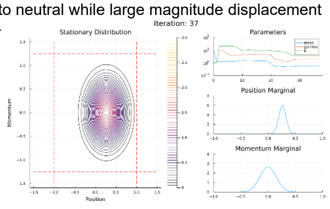
$$\min_{x \in \mathcal{X}} c(x) + c_L \mathbb{E}\left[|z|^2\right] + c_I P(z \in I)$$

$$x = (m, \gamma, k), \quad z \sim N(\mu, C)$$

$$\mu = -A^{-1} f_0, \quad AC + CA^T + BB^T = 0$$

$$I = \{z \notin [-q_b, q_b] \times [-p_b, p_b]\}$$





Joint probability distribution of position and momentum of optimized state (left). Parameter values through the optimization (top right). Position and Momentum marginal probability distributions (middle and bottom right).

Potential Impact

The impact of this project is incredibly wide ranging: by enabling the algorithmic calculation of gradients across disparate models and arbitrarily constructed iterative solvers, we will unlock sensitivity and optimization studies of models and simulations that would otherwise be intractable. The derivatives provided by this framework will decrease the burden on computational scientists to painstakingly construct derivatives manually, reduce unnecessary memory usage, and scale to a new class of problems featuring more controls and degrees of freedom than are possible with the current state of the art.

Collaboration Opportunities

The general nature of our project and the extremely widespread use case of gradient calculation means that there are many avenues for collaboration and opportunities to use our methods in other research areas. We are particularly interested in **physics-informed machine learning, uncertainty quantification, and extensions of this method to stochastic algorithms** such as Monte Carlo algorithms and Bayesian optimization approaches.

References

[2] Charles C. Margossian, “A review of automatic differentiation and its efficient implementation,” WIREs Data Mining and Knowledge Discovery, 9(4):e1305, 2019.
[3] Andrew Ning and Taylor McDonnell. Automating Steady and Unsteady Adjoints: Efficiently Utilizing Implicit and Algorithmic Differentiation. arXiv. 2023.
[4] J. D. Eldredge, “A method of immersed layers on Cartesian grids, with application to incompressible flows,” Journal of Computational Physics 448: 110716, 2022.
[5] Olga A. Doronina, Zachary J. Grey, and Andrew Glaws. “G2Aero: A Python Package for Separable Shape Tensors,” Journal of Open Source Software 8, no. 89 (September 20, 2023): 5408

This work was authored by the National Renewable Energy Laboratory, operated by Alliance for Sustainable Energy, LLC, for the U.S. Department of Energy (DOE) under Contract No. DE-AC36-08GO28308. Funding provided by the U.S. DOE Office of Science, Advanced Scientific Computing Research, Exploratory Research for Extreme-Scale Science program under DE-FOA-0002717. 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 or reproduce the published form of this work, or allow others to do so, for U.S. Government purposes.

DOE ASCR Computational Science Principal Investigators Meeting
Atlanta, GA. February 4-7, 2024
NREL/PO-2C00-88610