

Fast automated adjoints for spectral PDE solvers

Calum S. Skene¹ and Keaton J. Burns^{2,3}

¹Department of Applied Mathematics, University of Leeds, Leeds LS2 9JT, UK

²Department of Mathematics, Massachusetts Institute of Technology, Cambridge MA 02139, USA

³Center for Computational Astrophysics, Flatiron Institute, New York NY 10010, USA

Abstract

We present a general and automated approach for computing model gradients for PDE solvers built on sparse spectral methods, and implement this capability in the widely used open-source Dedalus framework. We apply reverse-mode automatic differentiation to symbolic graph representations of PDEs, efficiently constructing adjoint solvers that retain the speed and memory efficiency of this important class of modern numerical methods. This approach enables users to compute gradients and perform optimization for a wide range of time-dependent and nonlinear systems without writing additional code. The framework supports a broad class of equations, geometries, and boundary conditions, and runs efficiently in parallel using MPI. We demonstrate the flexibility and capabilities of this system using canonical problems from the literature, showing both strong performance and practical utility for a wide variety of inverse problems. By integrating automatic adjoints into a flexible high-level solver, our approach enables researchers to perform gradient-based optimization and sensitivity analyses in spectral simulations with ease and efficiency.

Numerical modeling has become a ubiquitous tool across every discipline of science and engineering. However, systematically using forward models to enhance scientific understanding often requires costly optimizations over model inputs and parameters. *Adjoint state methods* provide an efficient and systematic way to compute model gradients needed for such model-based optimization. Thus, the ability to compute adjoints forms the foundation of many scientific studies that would otherwise be infeasible. Indeed, the recent rise of machine learning and artificial intelligence as a transformative technology would not have been possible without adjoint methods, which form the basis of the backpropagation algorithms needed to train models. Beyond machine learning, adjoint methods have a rich history in scientific computing. They are an indispensable tool in non-modal stability theory [83, 72], parametric sensitivity analysis [46, 54], and weather forecasting via four-dimensional data assimilation (4D-Var) [14, 45]. Adjoint techniques have also been applied in countless optimization settings – for example, to design aerodynamic structures [34], mitigate thermoacoustic instabilities [50], optimize stellarator designs for nuclear fusion reactors [65, 40], and solve inverse problems in geophysics [66], among many other applications.

Despite their evident utility, adjoint-based computations remain underutilized in many fields, largely due to the difficulty of computing the adjoint operator, which is often a tedious and error-prone process. For numerical solvers of partial differential equations (PDEs) (e.g., fluid flow or

electromagnetics), there are two main strategies for obtaining adjoints. One approach is to analytically derive an adjoint system by integrating the governing equations by parts in both space and time; the resulting continuous adjoint equations are then discretized and solved numerically. Alternatively, the PDE can be discretized first, and then the adjoint of the resulting discrete system can be computed using tools like automatic differentiation (AD). These are known as the continuous and discrete adjoint approaches, respectively, each with its own advantages and disadvantages (see [23, 47] for detailed discussions).

Several open-source simulation codes now include built-in adjoint functionality (often termed “differentiable” codes), using a mix of continuous and discrete approaches. This includes SU² [64], FEniCS [44], Firedrake [25], Φ_{Flow} [30], simsopt [40], OpenFOAM [22], Exponax [38], JaxFluids [7, 6], Jax-CFD [37, 16], Trixi.jl [70], and Julia’s SciML ecosystem [69]. The majority of these PDE solvers focus on specific equation sets and are tied to particular applications or geometries. Notably, FEniCS and Firedrake are different in that they are high-level finite-element libraries designed for a wide variety of models. Using these packages, the governing equations are specified in variational (weak) form via the “unified form language” [1], and then automatically discretized by the library. Adjoint computations in these libraries are executed automatically using the dolfin-adjoint framework [55]. This high-level approach has led to numerous adjoint-based studies across a wide range of PDE applications.

While finite-element methods excel for complex geometries, global spectral methods are a popular and complementary alternative for problems in simpler geometries, such as turbulent fluid flow in a box or a sphere. These methods offer exponential accuracy as resolution is increased, and they facilitate efficient elliptic solves for differential-algebraic equations as well as implicit time-stepping for stiff problems [9]. They are widely used in fundamental research and underpin the world’s largest turbulence simulations [92] and state-of-the-art weather forecasting models [90]. Beyond classical Fourier-based spectral methods, recent advances have introduced large-scale sparse polynomial spectral algorithms that provide Fourier-like speed and accuracy in simple geometries (Cartesian, cylindrical, and polar coordinates). These developments have enabled cutting-edge simulations of biophysical, geophysical, and astrophysical flows (e.g., [33, 2, 86]).

Here we present a numerical approach to automatically generate discrete adjoints for the entire family of fast global spectral methods, implemented in the open-source **Dedalus** framework [12]. Like **FEniCS** and **Firedrake**, **Dedalus** is a high-level library that discretizes and solves user-defined PDEs. However, **Dedalus** allows users to specify systems of equations in strong form via a simple interface that is both flexible and accessible to computational scientists across many disciplines. **Dedalus** includes many spectral bases and has solver paths for eigenvalue problems (EVPs), linear boundary value problems (LBVPs), nonlinear boundary value problems (NLBVPs), and initial value problems (IVPs). **Dedalus** is written in *Python*, with compiled extensions for performance-critical routines, and it automatically handles distributed-memory parallelism via MPI. These features allow **Dedalus** simulations to be easily prototyped on a laptop and then ran at scale on a high-performance computing clusters.

In this article, we outline our efficient approach to automating adjoint computations for sparse spectral methods. We demonstrate its capabilities using **Dedalus** on several representative adjoint-based optimization problems drawn from a variety of scientific domains. These examples highlight the efficiency, flexibility, and ease of use of our differentiable extension of **Dedalus**. By working with discrete adjoints, our method automatically handles any boundary conditions, constraints, or gauge choices, eliminating the difficulties these pose for continuous-adjoint approaches. Furthermore, our approach allows sparse spectral solvers to be seamlessly integrated with machine learning frameworks, which inherently require differentiable simulators for training. Importantly, in all the examples below the gradient computation requires minimal user intervention: only a few additional lines of code are needed to obtain the gradient of any cost functional evaluated on a **Dedalus** solution. Overall, our work makes adjoint-based optimization readily accessible to computational scientists using spectral methods across disciplines.

Spectral adjoints

Continuous vs. discrete

The fundamental definition of an adjoint is mathematically simple: an adjoint operator \mathcal{A}^\dagger to a linear operator \mathcal{A} is one that satisfies $\langle x, \mathcal{A}y \rangle = \langle \mathcal{A}^\dagger x, y \rangle$ for all x and y , where $\langle \cdot, \cdot \rangle$ is a suitably chosen inner product. Efficiently solving the adjoint of a numerical model’s Jacobian yields the model’s gradients. Continuous methods discretize the infinite dimensional adjoint, while discrete methods discretize the forward operator and then take its finite-dimensional adjoint.

The continuous adjoint has the advantage of producing the true adjoint of the governing equations. However, it generally does not provide the exact gradient of the discrete forward model (even if the forward model is well-resolved), and deriving it – either manually or automatically – is difficult for constrained systems or complex boundary conditions. Implementing a continuous adjoint also requires writing a separate solver in addition to the original model. The discrete adjoint, on the other hand, yields the correct gradient for the discrete forward model. This advantage often translates into better convergence in outer-loop optimizations, although it can introduce instabilities when the forward model is under-resolved.

Several studies have manually implemented continuous adjoints for specific PDEs in **Dedalus** using its symbolic equation-entry interface. For example, adjoint methods have been used to explore the similarities between minimal seeds and instantons [41], to solve inverse problems [60], and to identify subcritical geodynamo solutions [74]. In a recent study [51], continuous and discrete adjoints were compared across several Cartesian problems in **Dedalus**, highlighting the advantages of the discrete approach – especially when used with optimization methods that benefit from the discrete adjoint’s superior convergence. In each of these cases, implementing the adjoint solver demanded significant user effort (analytical derivations or manual coding of the discrete adjoint), requiring specialist knowledge – particularly for bounded domains or high-order time-stepping schemes.

Importantly, the use of AD means discrete adjoints can be generated automatically from the forward model code, contributing to their rising popularity in recent years [26, 19, 10, 93]. However, current AD toolchains have limitations in programming language compatibility and hardware support. Moreover, one often needs to implement custom derivatives or “chain rules” for many optimized library functions (such as FFTs and linear algebra routines), since manually providing the Jacobian’s action can be more efficient and stable than tracing through the full forward computation. In fact, for the highly structured computations in many PDE solvers, custom rules might be required for nearly the entire code path, limiting the benefits of AD toolchains under such constraints.

Here we take the approach of leveraging the existing flexible, high-level code structures that enable **Dedalus** to forward model wide varieties of PDEs. We compute the adjoints of the various solver types at the outer level, and use the code’s internal computational graphs to effectively perform a highly structured, efficient, and platform-independent form of AD to evaluate the adjoints of the low-level operator implementations using their associated chain rules. Our approach eliminates the barriers associated with manually deriving continuous adjoint systems and produces discrete adjoints without requiring any modifications of the direct code or relying on external AD libraries. This combination enables **Dedalus** to automatically compute the discrete adjoint for every available solver type, geometry, and time-stepping scheme in an efficient and platform-independent manner.

Sparse spectral methods

Sparse spectral methods form finite-dimensional discretizations of PDEs by expanding the unknown solutions in a *trial basis* and projecting the equations against a *test basis*, enforcing the strong-form PDE via a weighted-residual method. Recent advances in the field have established optimal choices for the test and trial bases to produce maximally sparse discretizations for wide ranges of equations and domains [61, 84, 85, 42, 62, 17]. Linear operators can be directly discretized into sparse (often narrowly banded) matrices, which are fast to apply or invert. Nonlinear terms can be efficiently evaluated on a collocation grid using fast transforms (e.g. FFTs). Boundary conditions and other constraints can be applied by modifying the discretized systems, or at the continuous level through the use of Lagrange multipliers and tau terms [11].

Generic solvers for wide classes of BVPs and IVPs can then be readily automated as a series of sparse linear system solves against explicitly computed right-hand sides (RHSs); for instance, this encapsulates both Newton iterations for BVPs and mixed implicit-explicit (IMEX) timestepping for IVPs. For notational brevity, we will denote such a forward solution as a vector of solution coefficients \mathbf{X} satisfying the equation $\mathbf{F}(\mathbf{X}, \mathbf{p}) = 0$, where \mathbf{p} is a vector of problem parameters. In the Methods section, we provide more detail on how EVPs and IVPs are handled, but we will illustrate the process using this simple notation that more naturally matches the form of BVPs. The goal of the discrete adjoint program is to efficiently compute gradients of solution properties with respect to the parameters, and to do so by reusing the flexible and efficient computational motifs (sparse direct solvers and fast transforms) that enable the automated forward solution of PDEs with spectral methods.

Efficiently calculating gradients

Here we summarize how discrete adjoints are used to obtain the derivative with respect to parameters \mathbf{p} – which could be forcing terms, initial conditions, or equation parameters – of a functional applied to the spectral solution of a PDE. We denote the target functional as $J(\tilde{\mathbf{X}}, \mathbf{p})$, where $\tilde{\mathbf{X}}$ is a generic vector in the finite-dimensional solution space. We seek to compute the derivative at a point $\tilde{\mathbf{X}} = \mathbf{X}(\mathbf{p})$ which solves the discretized PDE with the specified parameters, i.e. $\mathbf{F}(\mathbf{X}(\mathbf{p}), \mathbf{p}) = 0$. As a functional purely of the parameters, we define $\mathcal{J}(\mathbf{p}) = J(\mathbf{X}(\mathbf{p}), \mathbf{p})$. Directly differentiating this functional by the chain rule yields

$$\frac{d\mathcal{J}}{d\mathbf{p}} = \frac{\partial J}{\partial \mathbf{p}} + \frac{\partial J}{\partial \tilde{\mathbf{X}}} \cdot \frac{d\mathbf{X}}{d\mathbf{p}}. \quad (1)$$

where the partial derivatives are evaluated at $(\mathbf{X}(\mathbf{p}), \mathbf{p})$. The problem in computing this gradient explicitly is the last term, $d\mathbf{X}/d\mathbf{p}$, since directly calculating this Jacobian by e.g. finite differences requires solving the forward equation for every element of \mathbf{p} , which becomes prohibitively expensive as the number of parameters increases.

To obtain the gradient with a tractable computational procedure, the adjoint state method approaches this as a constrained optimization problem (for a detailed review, see [66]). The associated Lagrangian is

$$\mathcal{L}(\tilde{\mathbf{X}}, \tilde{\mathbf{Y}}, \mathbf{p}) = J(\tilde{\mathbf{X}}, \mathbf{p}) - \langle \tilde{\mathbf{Y}}, \mathbf{F}(\tilde{\mathbf{X}}, \mathbf{p}) \rangle, \quad (2)$$

where $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{Y}}$ are generic vectors. Applying the chain rule directly to \mathcal{L} gives

$$\frac{d\mathcal{L}}{d\mathbf{p}} = \frac{\partial \mathcal{L}}{\partial \mathbf{p}} + \frac{\partial \mathcal{L}}{\partial \tilde{\mathbf{X}}} \cdot \frac{\partial \tilde{\mathbf{X}}}{\partial \mathbf{p}} + \frac{\partial \mathcal{L}}{\partial \tilde{\mathbf{Y}}} \cdot \frac{\partial \tilde{\mathbf{Y}}}{\partial \mathbf{p}}. \quad (3)$$

By construction the last term in this expression will be zero when evaluating the gradient at $\tilde{\mathbf{X}} = \mathbf{X}(\mathbf{p})$ because $(\partial \mathcal{L} / \partial \tilde{\mathbf{Y}})|_{\mathbf{X}} = -\mathbf{F}(\mathbf{X}, \mathbf{p}) = 0$. Calculating $\partial \mathcal{L} / \partial \tilde{\mathbf{X}}$ yields

$$\frac{\partial \mathcal{L}}{\partial \tilde{\mathbf{X}}} = \frac{\partial J}{\partial \tilde{\mathbf{X}}} - \left(\frac{\partial \mathbf{F}}{\partial \tilde{\mathbf{X}}} \right)^{\dagger} \tilde{\mathbf{Y}}, \quad (4)$$

where \dagger indicates the Hermitian adjoint. This term can be set to zero by choosing $\tilde{\mathbf{Y}} = \mathbf{Y}(\mathbf{p})$, where $\mathbf{Y}(\mathbf{p})$ satisfies the *adjoint state equation*

$$\left(\frac{\partial \mathbf{F}}{\partial \tilde{\mathbf{X}}} \right)^{\dagger} \mathbf{Y}(\mathbf{p}) = \frac{\partial J}{\partial \tilde{\mathbf{X}}}, \quad (5)$$

where again all partials are evaluated at $(\mathbf{X}(\mathbf{p}), \mathbf{p})$. Hence, by setting $\tilde{\mathbf{X}} = \mathbf{X}(\mathbf{p})$, $\tilde{\mathbf{Y}} = \mathbf{Y}(\mathbf{p})$, and by using the fact that $\mathcal{L}(\mathbf{X}(\mathbf{p}), \cdot, \mathbf{p}) = \mathcal{J}(\mathbf{p})$, we obtain

$$\frac{d\mathcal{J}}{d\mathbf{p}} = \frac{d\mathcal{L}}{d\mathbf{p}} = \frac{\partial \mathcal{L}}{\partial \mathbf{p}} = \frac{\partial J}{\partial \mathbf{p}} - \left\langle \mathbf{Y}, \frac{\partial \mathbf{F}}{\partial \mathbf{p}} \right\rangle. \quad (6)$$

(6) is key to the adjoint method and shows that by solving a linear equation for $\mathbf{Y}(\mathbf{p})$ we can efficiently obtain the

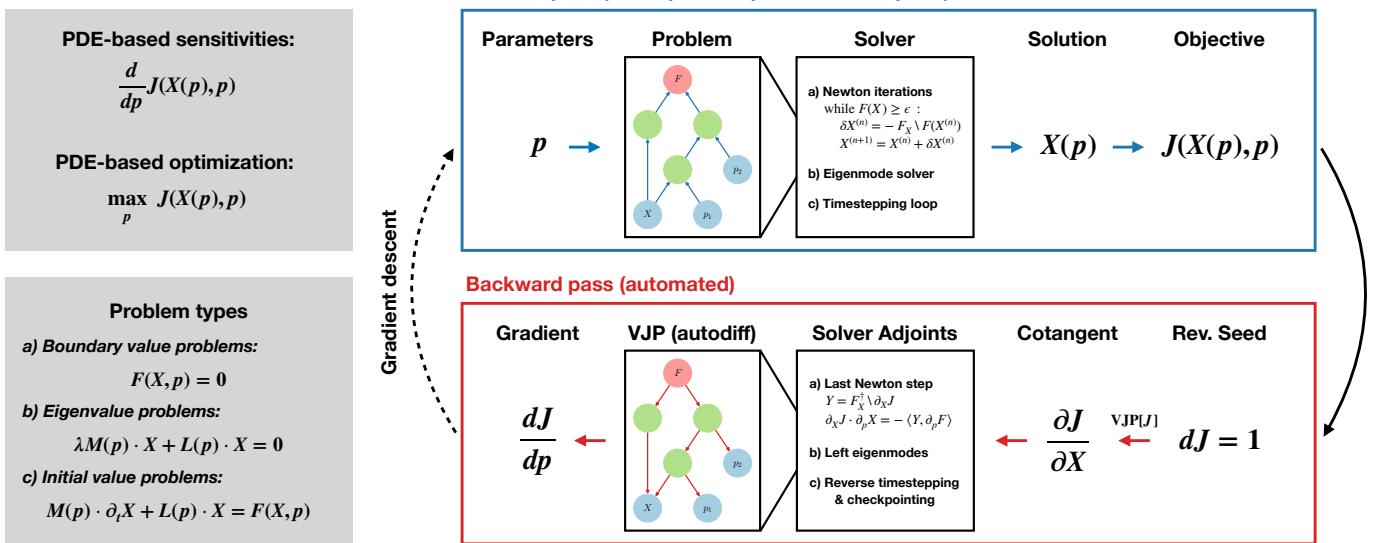


Figure 1: Depiction of the adjoint-looping process to perform efficient PDE-based optimization using automatic discrete adjoints in **Dedalus**. Left: we seek to optimize a functional J of the solution \mathbf{X} of a PDE (BVP, IVP, or EVP) over parameters \mathbf{p} . Top right: **Dedalus** already enables solving generic PDEs with fast spectral methods with operator graphs representing the PDE (forward pass). Bottom right: we have now automated the adjoint (backward pass) by reversing the high-level solver control flow and implementing efficient vector-Jacobian products (VJPs) on the operator graphs, akin to reverse-mode automatic differentiation. The result is an optimally fast computation of the discrete model gradient.

gradient of a functional with respect to parameters, since the dimension of the adjoint state is independent of the number of parameters.

To summarize, in the adjoint state method:

1. The forward problem is first solved to determine $\mathbf{X}(\mathbf{p})$.
2. The adjoint problem ((5)) is then solved for $\mathbf{Y}(\mathbf{p})$.
3. Finally, the right-hand side of (6) is computed to determine the gradient $\partial J / \partial \mathbf{p}$.

The first (forward) step is already automated in **Dedalus** using optimally sparse spectral methods. We have now automated the second and third (adjoint) steps, enabling automatic PDE-based optimization through a simple user interface. The principal technical requirements are:

- Implementing fast direct solvers for the Jacobian adjoint systems appearing on the left-hand side of (5), based on reusing sparse factorizations from the forward solver.
- Adding fast evaluations of the derivative terms on the right-hand side of (5) and (6) using matrix-free vector-Jacobian products for arbitrary nonlinear operator trees.

The workflow of the adjoint looping process using these tools is illustrated in Fig. 1. More details about the technical implementations are in the Methods section and the Supplementary Information.

Results

To demonstrate the simplicity and flexibility of our approach to computing discrete adjoints for sparse spectral solvers, we have created fast implementations using **Dedalus** of several canonical problems in PDE-based optimization. We have chosen examples using a variety of solver types, illustrating different scenarios in which gradient information is utilized. They are also chosen to cover a range of application areas and domain geometries, showcasing the flexibility of our framework and highlighting its suitability for enhancing a wide range of PDE-based modeling studies. The scripts for these examples are available online¹ and include instructions for installing the necessary software.

Parametric sensitivity and numerical continuation

Since adjoints provide efficient access to gradient information, they are an ideal tool for parametric sensitivity analyses where we seek the direct impact of problem parameters on the PDE solution. Here we consider the process of computing the neutral stability curve for plane Poiseuille flow [63], a canonical fluid dynamics problem that describes pressure-driven flow between two flat plates situated at

¹https://github.com/csskene/dedalus_adjoint_examples

$y = 0$ and $y = 2$. In non-dimensional variables, the stability problem can be written as an eigenvalue problem involving the velocity and pressure perturbations \mathbf{u} and pressure p as

$$\begin{aligned} \lambda \mathbf{u} + \mathbf{u}_0 \cdot \nabla \mathbf{u} + \mathbf{u} \cdot \nabla \mathbf{u}_0 &= -\nabla p + \frac{1}{\text{Re}} \nabla^2 \mathbf{u}, \\ \nabla \cdot \mathbf{u} &= 0, \\ \mathbf{u}(y = \pm 1) &= 0, \end{aligned} \quad (7)$$

where $\mathbf{u}_0 = y(2-y)\hat{\mathbf{e}}_x$ is the base flow and Re is the Reynolds number. The real part of the eigenvalue λ is the growth rate ($\gamma = \Re \lambda$), and the imaginary part gives the oscillation frequency. As the base flow is independent of x , this stability problem is separable over streamwise wavenumbers α . For what follows, we consider only 2D perturbations (with no z -dependence), and seek to find the sets of parameters (α, Re) such that the maximum growth rate is zero, indicating neutral stability.

To accelerate the process of computing the neutral curve, we utilize discrete model adjoints in **Dedalus** in two ways. First, we guess a point close to the neutral curve and use the EVP solver and its adjoint to compute the largest growth rate $\gamma_{\max}(\text{Re}, \alpha)$ and its parametric gradient at this point. Using a Newton solver, we can then efficiently find a point on the neutral curve where $\gamma_{\max} = 0$. Second, to construct the guess for a new point on the neutral curve, we use the parametric gradient to find the parametric tangent to the neutral curve, from which a new guess can be estimated. By repeating this process, we can efficiently trace the neutral curve without difficulties continuing around turning points.

Figure 2 shows the result of this procedure. Each point indicated on this curve has a growth rate of zero to machine precision and was calculated with approximately five eigenvalue solves. This leads to a very efficient overall calculation without requiring any multidimensional grid searches. While this equation only contains two parameters, the adjoint method allows quick calculations of eigenvalue sensitivities with respect to potentially many more. Furthermore, the adjoint implementation reuses the LU decomposition from the forward problem, providing significant computational savings over, e.g., a finite-difference-based approach which would require new forward solves and factorizations.

Nonlinear optimization

Propagating sensitivities through nonlinear evolutionary equations is possible with *adjoint looping*, where the backward pass corresponds to a reverse-time integration using the time-dependent linearization of the forward model. This technique allows optimizing, e.g., final-time functionals with respect to equation parameters and initial conditions. To demonstrate the capabilities of automatic differentiation in **Dedalus** as a tool for nonlinear optimization, we now consider the problem of optimizing dynamo action in a ball with insulating boundary conditions, following the

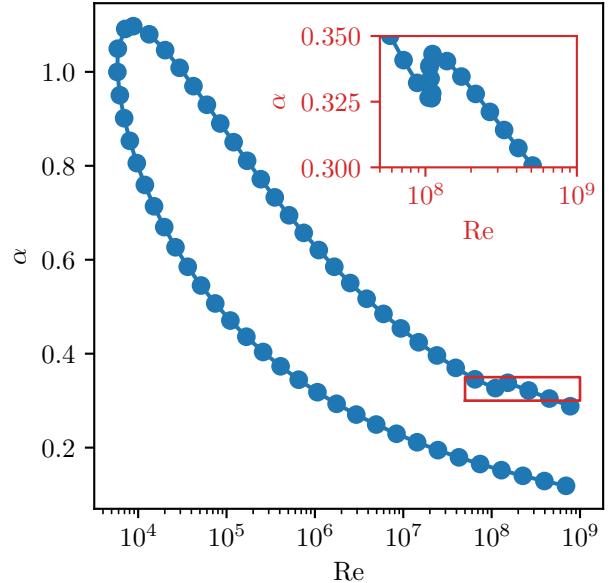


Figure 2: Neutral stability curve for plane Poiseuille flow obtained with adjoint-based eigenvalue sensitivities and numerical continuation. Each point on the curve has a growth rate of zero to machine precision. The parameters are the horizontal wavenumber α and the base-flow Reynolds number Re .

work of *Chen et al.* [13]. This is an important and challenging example of using nonlinear optimization to assess the stability of fluid flows (see the review [36] and references therein).

In the astrophysical context, dynamo theory deals with the growth of magnetic fields via the motion of a conducting fluid – an important physical process underlying magnetic field generation in planets, stars, accretion disks, and galaxies (for a review, see [81]). This problem is governed by the (non-dimensional) induction equation

$$\begin{aligned} \frac{\partial \mathbf{B}}{\partial t} - \eta \nabla^2 \mathbf{B} &= \nabla \times (\mathbf{u} \times \mathbf{B}), \\ \nabla \cdot \mathbf{B} &= 0, \end{aligned} \quad (8)$$

where the fluid velocity is \mathbf{u} and the magnetic diffusivity is η . Although this is a linear equation for \mathbf{B} when the velocity field is prescribed, as a joint optimization task it is nonlinear (in both parameter and state), and obtaining growing solutions can be a highly nontrivial and subtle task.

Scaling length with the domain radius R , velocity with ΩR for a prescribed vorticity scale Ω , time with R^2/η , and magnetic field with an arbitrary strength B , the induction equation includes a single non-dimensional parameter, the *vorticity*-based magnetic Reynolds number $\text{Rm} \equiv R^2\Omega/\eta$. In order to ensure the magnetic field remains solenoidal, we formulate the problem using a magnetic vector potential \mathbf{A} ,

where $\mathbf{B} = \nabla \times \mathbf{A}$, as

$$\begin{aligned}\frac{\partial \mathbf{A}}{\partial t} - \nabla^2 \mathbf{A} + \nabla \psi &= \text{Rm } \mathbf{u} \times \mathbf{B}, \\ \nabla \cdot \mathbf{A} &= 0,\end{aligned}\quad (9)$$

These equations include a scalar field ψ that enables us to satisfy the Coulomb gauge condition (that the vector potential also be solenoidal). Since the temporal eigenmodes of \mathbf{B} correspond to temporal eigenmodes of \mathbf{A} with the same growth rate, optimizing the growth of \mathbf{A} beyond initial transients will yield the same optimal velocity field \mathbf{u} as directly optimizing the growth of \mathbf{B} .

Defining the volume-weighted norm $\|\mathbf{X}\|^2 = (1/V) \int \mathbf{X} \cdot \mathbf{X} dV$ and following [13], we formulate the optimization problem as maximizing

$$J = \log \|\mathbf{A}(T)\|^2 \quad (10)$$

with constraints $\|\mathbf{A}(0)\| = 1$ and $\|\boldsymbol{\omega}\| = 1$, where $\boldsymbol{\omega} = \nabla \times \mathbf{u}$ is the vorticity field. Note that the norm of the vorticity, rather than the velocity, is fixed due to the results of [68]. We specify vacuum boundary conditions for the magnetic field, which can be written in terms of the spherical harmonic decomposition of \mathbf{A} as

$$\frac{\partial \mathbf{A}_{\ell,\sigma}}{\partial r} - \frac{\ell + \sigma}{r} \mathbf{A}_{\ell,\sigma} = 0 \quad \text{at } r = 1, \quad (11)$$

where ℓ is the spherical harmonic degree and we have used the regularity component index $\sigma \in \{-1, 1, 0\}$ [85]. Since the norm of $\boldsymbol{\omega}$ is constrained, we solve for \mathbf{u} by solving the auxiliary LBVP

$$\begin{aligned}\nabla \times (\nabla \times \mathbf{u}) + \nabla \chi &= \nabla \times \boldsymbol{\omega}, \\ \nabla \cdot \mathbf{u} &= 0,\end{aligned}\quad (12)$$

together with no-slip conditions on \mathbf{u} . This ensures that the velocity field remains divergence-free and satisfies appropriate boundary conditions. The norm constraints are maintained by using the Pymanopt [82] library to perform the optimization on the product manifold $(\mathbf{A}(0), \boldsymbol{\omega}) \in \Pi = V_{\mathbf{W}} \times V_{\mathbf{W}}$, where $V_{\mathbf{W}} = \{\mathbf{x} \mid \mathbf{x}^T \mathbf{W} \mathbf{x} = 1\}$ is a generalized Stiefel manifold with a weight matrix \mathbf{W} corresponding to the discretized L^2 norm. Although the IVP is a linear equation for \mathbf{A} , it becomes nonlinear when including \mathbf{u} as a parameter, and therefore forms a nonlinear optimization problem requiring checkpointing.

The results of the optimization procedure are shown in Fig. 3. From the time series, we see that for all magnetic Reynolds numbers there is a short period of transient growth (up to $t \approx 0.1$), followed by an exponentially growing period in which only the mode with the largest growth rate remains. For $\text{Rm} \lesssim 64.45$, the flow is stable and this growth rate is negative, whereas for $\text{Rm} \gtrsim 64.45$ the growth rate is positive and exponential growth is observed. For $\text{Rm} \approx 64.45$ the growth rate is approximately zero, indicating that this is the critical magnetic Reynolds number

(agreeing with [13]). The figure also shows the streamlines and field lines of the optimal velocity and magnetic fields, respectively. We see that the flow is mainly concentrated near the center of the ball, where a large increase in flow velocity occurs with a twisting motion. Again, this agrees with the results of [13].

To highlight the flexibility of our method, we also perform the optimization problem of maximizing

$$J = \log \|\mathbf{B}(T)\|^2 \quad (13)$$

subject to $\|\mathbf{B}(0)\|_2 = 1$. This is done by solving the induction equation directly for \mathbf{B} . Fig. 3 shows the resulting magnetic energy under this optimization compared to the initial optimization for \mathbf{A} . As expected, both energies show the same growth rates after the initial transient. However, the amount of transient growth is notably different, with the magnetic-energy-based optimization leading to substantially more absolute growth. This example emphasizes the flexibility of our automated adjoint capabilities for spectral solvers, allowing the user to easily examine problems from different viewpoints using different equation formulations.

Resolvent analysis

Resolvent analysis is a powerful modal analysis technique [78], whose applications to turbulence were pioneered by McKeon and Sharma [53]. In their setup, the flow field is decomposed into a statistically stationary mean flow $\bar{\mathbf{u}}$ and fluctuations \mathbf{u}' , such that $\mathbf{u} = \bar{\mathbf{u}} + \mathbf{u}'$. Substituting this into the Navier-Stokes equations and Fourier transforming in time results in the following coupled systems of equations:

$$\begin{aligned}i\omega \hat{\mathbf{u}}' + \bar{\mathbf{u}} \cdot \nabla \hat{\mathbf{u}}' + \hat{\mathbf{u}}' \cdot \nabla \bar{\mathbf{u}} + \nabla \hat{p} - \frac{1}{\text{Re}} \nabla^2 \hat{\mathbf{u}}' &= \mathcal{F}[\mathbf{u}' \cdot \nabla \mathbf{u}'](\omega), \\ \nabla \cdot \hat{\mathbf{u}}' &= 0,\end{aligned}\quad (14)$$

and

$$\begin{aligned}\bar{\mathbf{u}} \cdot \nabla \bar{\mathbf{u}} + \nabla \bar{p} - \frac{1}{\text{Re}} \nabla^2 \bar{\mathbf{u}} &= \mathcal{F}, \\ \nabla \cdot \bar{\mathbf{u}} &= 0,\end{aligned}\quad (15)$$

where we denote the Fourier transform at frequency ω as $\mathcal{F}[\cdot](\omega)$. (14) show that fluctuations at frequency ω are driven by triadic interactions, represented by the right-hand side. In turn, (15) show that the zero-frequency component of these triadic interactions sustains the mean flow.

Instead of solving the fully coupled system, mean flow data can be obtained from numerical simulations or experiments in order to directly study the fluctuating components. In doing so, (14) can be reduced to the form:

$$(i\omega \mathbf{M} - \mathbf{L}) \hat{\mathbf{u}}' = \hat{\mathbf{f}}, \quad (16)$$

where \mathbf{M} is a mass matrix, \mathbf{L} is a linear operator, and the triadic interactions are written as a forcing term $\hat{\mathbf{f}}$.



Figure 3: Optimal kinematic dynamo solutions in the ball using nonlinear optimization via adjoint looping. Top left: Time series showing the result of optimizing the vector potential norm at different magnetic Reynolds numbers. Bottom left: Time series comparing optimization of the magnetic field norm versus the vector potential norm. Middle: Streamlines of the optimal velocity field for $Rm = 64.45$. Right: Field lines of the optimal magnetic field at $t = 1$ for $Rm = 64.45$.

Resolvent analysis treats $\hat{\mathbf{f}}$ as a generic forcing term arising from turbulence and examines (16) as an input-output system in which a transfer function $\mathcal{H} = (i\omega\mathbf{M} - \mathbf{L})^{-1}$ maps a forcing $\hat{\mathbf{f}}$ to its driven response $\hat{\mathbf{u}}'$. This transfer function \mathcal{H} is known as the resolvent. Of interest are the leading singular values and vectors of this operator, i.e., $(\sigma_i, \hat{\mathbf{u}}'_i, \hat{\mathbf{f}}_i)$ such that $\mathcal{H}\hat{\mathbf{f}}_i = \sigma_i\hat{\mathbf{u}}'_i$, and the vectors $\hat{\mathbf{u}}'_i$ and $\hat{\mathbf{f}}_i$ are orthonormal under the L^2 (energy) norm. The gains σ_i measure the amount of amplification a particular forcing induces in the flow response. In situations where the largest gain σ_1 is much greater than the next largest σ_2 , the flow is deemed low-rank, and a generic forcing $\hat{\mathbf{f}}$ is likely to produce a response that resembles $\hat{\mathbf{u}}'_1$ due to the orthogonality of the singular basis.

Hence, a resolvent analysis reveals two key insights. First, it systematically identifies frequencies at which turbulence is likely to manifest, via the magnitude of the gains at each frequency. Second, for frequencies at which the flow is found to be low-rank, the leading response mode resembles coherent structures expected to be found in fully developed turbulence. Furthermore, by examining the corresponding forcing mode, one can gain important information relevant for flow control (see, e.g., [79, 71]). While we focus here on resolvent analysis for turbulent flows (about the mean flow), it is worth noting that resolvent analysis for steady flows (about fixed points) is also an important concept in non-modal stability analysis – predating its use in turbulence [83, 73] – and serves as the driven counterpart to transient growth analysis.

Using our automatic adjoint routines in **Dedalus**, we reproduce the resolvent analysis for turbulent pipe flow fol-

lowing [53]. Since the mean flow is obtained by averaging over ϕ and z , it depends only on the radial coordinate, allowing us to solve for the forcing and response independently for each azimuthal wavenumber m and streamwise wavenumber k . We use **Dedalus** to compute the action of both \mathcal{H} and \mathcal{H}^\dagger for given m and k in a cylindrical geometry. This is done using a linear boundary value problem (LBVP) to apply \mathcal{H} via solving the associated matrix pencil, and applying the adjoint via the vector-Jacobian product of the LBVP. Together, these routines are used to compute the leading singular vectors via the **SciPy** sparse SVD. The mean flow is taken from experimental data [52] at $Re = 74345$.

Fig. 4 shows the resolvent results for azimuthal wavenumber $m = 10$ and streamwise wavenumber $k = 1$, in excellent agreement with [53]. The figure illustrates that over a wide range of frequencies, the flow is low-rank, with an order-of-magnitude separation between σ_1 and σ_2 . The optimal response near the peak gain at $\omega = 0.5$ is concentrated near the wall and resembles very large-scale motions observed in turbulent pipe experiments [53]. This example demonstrates the ease of setting up and performing a resolvent analysis using a differentiable spectral code. Since many turbulent experimental configurations – such as pipe flow, channel flow, and Taylor-Couette flow – can be modeled spectrally, there is a clear advantage to complementing these experiments with resolvent analysis. In particular, we envision that resolvent analyses for fluids not modeled by the incompressible Navier-Stokes equations (e.g., compressible, conducting, non-Newtonian, or viscoelastic fluids) will be made more accessible by these developments, which significantly ease the numerical burden.

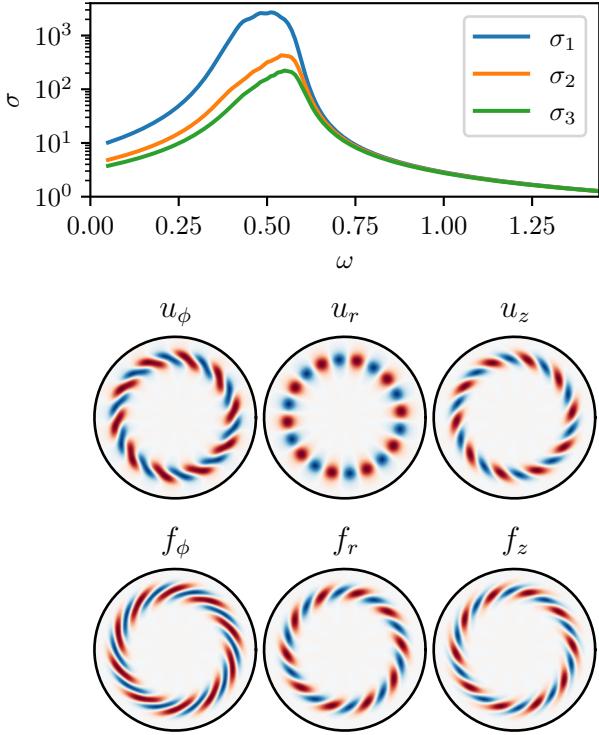


Figure 4: Resolvent analysis for turbulent pipe flow at $\text{Re} = 74345$ following *McKeon et al.* [52], for axial wavenumber $k = 1$ and azimuthal wavenumber $m = 10$. Top: Leading gains (singular values) as a function of frequency, showing the optimal response at $\omega \approx 0.5$. Bottom: Optimal forcing and response modes, showing structures similar to those observed in the experimental fluctuations.

Phase reduction analysis

Phase-reduction analysis is a technique with origins in mathematical biology and neuroscience, used to study phase dynamics and synchronization in oscillating systems [91, 18]. More recently, it has gained traction in the fluid mechanics community [77], where it has been applied to assess the synchronization properties of fluid-structure interactions [43], thermoacoustic instability [75], and to design flow actuation strategies in aeronautical engineering applications [57, 24].

For a stable limit-cycle solution with period T , the notion of phase can be defined as follows. The phase θ is first defined for states on the limit cycle as $2\pi t/T$ for $t \in [0, T]$. In other words, for a state on the limit cycle, the phase variable $\theta \in [0, 2\pi]$ assigns a scalar label to each state, and satisfies the simple ODE $\dot{\theta} = 2\pi/T$ on the cycle. This definition can be extended to states in the vicinity of the limit cycle by using the fact that the cycle is stable, and therefore any small perturbation returns to it. Hence, for a state near the limit cycle, we define its phase as the phase of the point on the limit cycle whose trajectory it matches

asymptotically as $t \rightarrow \infty$. Under this definition, the phase dynamics near the limit cycle are governed by the ODE

$$\dot{\theta} = \frac{2\pi}{T} + \mathbf{z}(\theta) \cdot \mathbf{h}(\theta), \quad (17)$$

where $\mathbf{h}(\theta)$ represents a small perturbation to the system. The term $\mathbf{z}(\theta)$ is the phase sensitivity function, whose computation is the central task in a phase-reduction analysis. By obtaining \mathbf{z} , the phase response can be determined independently of the specific form of the perturbation, making this a powerful technique for analyzing the phase and synchronization dynamics of oscillators.

As an example of how adjoints enable spectral methods for phase-reduction analysis, we consider the FitzHugh-Nagumo model [20, 56]:

$$\begin{aligned} \frac{du}{dt} &= u - \frac{u^3}{3} - v + I, \\ \frac{dv}{dt} &= \epsilon(u + a - bv), \end{aligned} \quad (18)$$

a simplified version of the Hodgkin-Huxley model [29]. These equations model the firing dynamics of a neuron, with membrane potential u and recovery variable v . The parameters include ϵ (which sets the timescale separation between u and v), the input current I , and constants a and b describing the activation dynamics. For $I \neq 0$, the system exhibits slow-fast repetitive firing for a range of parameter values, yielding a stable limit-cycle solution.

Of interest is the phase response of the dynamics about this limit cycle. This includes how perturbations can delay or advance the phase of the solution, synchronization of neurons to external stimuli [39], collective synchronization of coupled neurons [31], and synchronization of oscillators to common white noise [59]. The key to understanding all of these phenomena lies in computing the phase sensitivity function, which can be efficiently obtained via adjoint methods (see [18], for example), as detailed below for the FitzHugh-Nagumo model. This connection between the adjoint problem and the phase sensitivity function is essential for efficiently conducting a phase-reduction analysis, particularly in PDE systems, as recently demonstrated for fluid flows [35].

To compute the phase sensitivity function using *Dedalus*, we first compute the limit-cycle solution (u_0, v_0) . To do this, we discretize (18) in time using a Fourier basis, and solve it as a nonlinear boundary value problem (NLBVP) to obtain a spectrally accurate periodic solution. Floquet theory [21] then gives that perturbations to this limit cycle (u', v') satisfy the eigenvalue problem

$$\begin{aligned} \frac{du'}{dt} + \lambda u' &= u' - u_0^2 u' - v', \\ \frac{dv'}{dt} + \lambda v' &= \epsilon(u' - bv'), \end{aligned} \quad (19)$$

for eigenvalue λ , where u' and v' are again discretized using a periodic (Fourier) basis in t (the Floquet-Fourier-Hill

method [28]). As the system is autonomous, it has a neutral eigenvalue $\lambda = 0$, with corresponding eigenvector (\dot{u}_0, \dot{v}_0) representing phase shifts of the limit cycle. The phase sensitivity function, which allows the projection of dynamics onto this phase shift, is obtained as the corresponding adjoint eigenvector, normalized such that $\langle \mathbf{z}, (\dot{u}_0, \dot{v}_0)^T \rangle = 1$. Thus, the phase sensitivity function can be computed in **Dedalus** simply by solving the adjoint of (19).

The results are shown in Fig. 5 for parameters $a = 0.7$, $b = 0.8$, $\epsilon = 0.08$, and $I = 0.8$, demonstrating excellent agreement with [58]. This approach can be easily adapted to other equation sets, including PDEs, providing a systematic route to phase-reduction analysis for a wide range of oscillators.

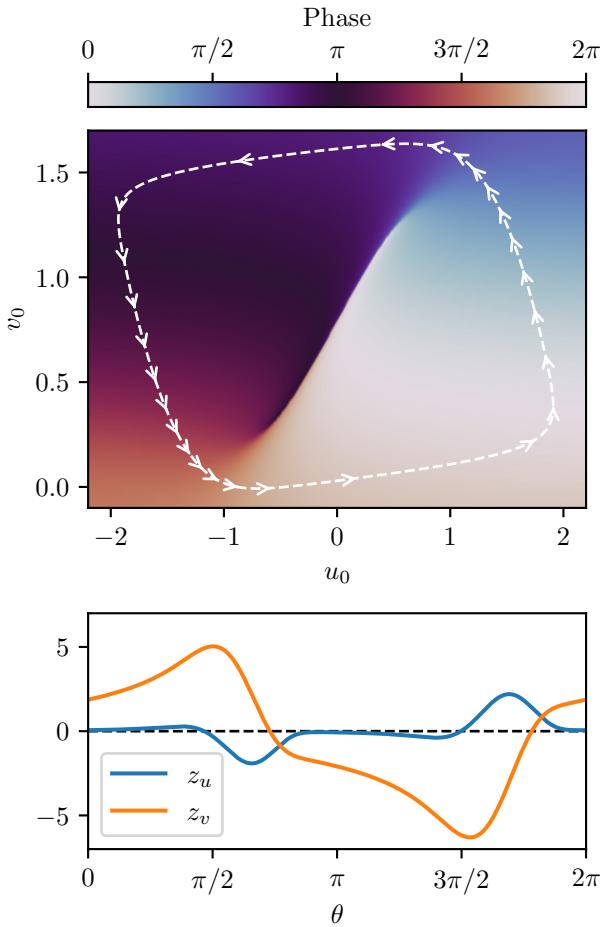


Figure 5: Phase sensitivity analysis of the FitzHugh-Nagumo model via discrete eigenvalue problem adjoints. Top: Limit cycle and explicitly computed phase function. Arrows on the limit cycle are equispaced in time, illustrating the slow-fast dynamics of the oscillator. Bottom: The phase sensitivity components measuring the gradient of the phase function along the limit cycle.

Discussion

We have presented a framework for efficiently producing adjoint solvers for modern sparse methods and have demonstrated its implementation in the **Dedalus** library. By leveraging the structure of spectral solvers, we have developed a hybrid approach in which the adjoint is explicitly provided for core routines and determined via high-level automatic differentiation for others. By individually handling components such as linear system solves, spectral transforms, timestepping routines, eigenvalue solvers, and Newton solvers, we are able to unobtrusively compute the adjoint of many different PDE solution techniques.

Furthermore, we reuse data structures from the forward code – such as matrix decompositions and transform plans – so that the resulting adjoint code is both computationally and memory efficient. We complement this with automatic differentiation, using it to evaluate vector-Jacobian products of nonlinear terms based on their graph representation in **Dedalus**. This enables users to obtain adjoint information for general PDEs directly from the forward model implementation, without requiring additional libraries or restructuring of the original code.

To demonstrate the capabilities of this approach, we have implemented canonical applications of adjoint methods in problems from classical fluid mechanics, dynamo theory, modal analysis of turbulence, and mathematical neuroscience. These examples span a range of problem types and geometries, showcasing the versatility and advantages of a generic differentiable spectral code. Specifically, our examples illustrate how such a code can be used for parametric sensitivity analysis, numerical continuation, nonlinear optimization, non-modal stability theory, and phase sensitivity analysis. These methods are applicable across scientific disciplines and utilize gradient information in diverse ways. By extending the availability of open-source differentiable solvers to include spectral methods, we provide broad new opportunities for adjoint-based studies in areas where spectral solvers are critical, and do so without requiring custom code development.

While we have focused here on first-order derivative information, a natural next step is the automation of higher-order derivatives [49]. This would enable techniques such as Newton methods for optimization, as well as uncertainty quantification and Bayesian inference, which can benefit from efficient Hessian-vector products for posterior generation (see, e.g., [80, 32]). Additionally, it is now possible to combine **Dedalus** models with machine learning libraries by manually linking gradient computations. Creating a dedicated interface to simplify this integration – similar to the interface between **Firedrake** and PyTorch [8] – would enable seamless end-to-end model-based training for a wide range of applications.

Finally, a larger – but critically important – future direction is enabling **Dedalus** solvers to run on graphical and tensor processing units. Beyond accelerating tradi-

tional solvers, this would greatly expand the applicability of **Dedalus** to machine learning, statistical inference, and nonlinear optimization problems, all of which require solving large numbers of forward and reverse problems. These exciting research directions build on the strong foundation we have established in rendering **Dedalus** fully differentiable in a platform-independent fashion.

Methods

Solving the adjoint state equation, (5), requires adjoint linear-system solves – e.g., for a sparse matrix A – and the evaluation of vector-Jacobian products for the right-hand-side (RHS) terms, such as a nonlinear operator $N(\mathbf{X}, \mathbf{p})$. Here, we outline the implementation approach used to efficiently automate these processes in **Dedalus**. For details on how different problem types map to this structure, see the Supplementary Information.

Linear system solves

In each forward iteration, the linear system is solved using a fast direct solver, typically via a sparse LU factorization of A . The corresponding adjoint solve is then obtained by applying the adjoints of the LU factors of A .

Nonlinear operator graphs

Generic nonlinear terms, such as N , are represented in **Dedalus** as directed acyclic graphs (DAGs) of operators acting on the fields and parameters of the PDE. These graphs are evaluated via depth-first recursion, which triggers spectral transforms as needed to compute intermediate operators.

Forward-mode sensitivity propagation (tangent mode) through an operator graph is implemented as a matrix-free Jacobian-vector product (JVP) evaluated concurrently with the primal graph. This only requires each operator to provide its discrete derivative.

Reverse-mode sensitivity propagation (adjoint mode, or cotangent propagation) is implemented as a matrix-free vector-Jacobian product (VJP) utilizing the existing DAG operator structure. During the forward evaluation of the operator tree, a tape records the evaluation order of intermediate operators, forming a topological sort of the DAG. The VJP is computed by propagating the cotangents backward through this recorded evaluation order and accumulating at the root nodes. This approach is equivalent to reverse-mode automatic differentiation but is performed in a reliably memory- and compute-efficient manner by leveraging the high-level structure of the forward operator DAG.

Spectral transforms

During the evaluation of nonlinear operator graphs, spectral transforms are employed to efficiently apply differential operators to field coefficients in their sparse spectral representations, while nonlinearities are evaluated on a collocation grid. This pseudospectral approach is substantially faster than either pure-collocation or pure-spectral approaches when fast transforms are available.

When evaluating operator VJPs, the adjoint of each spectral transform must therefore be applied. To automate this process, we have implemented specialized “adjoint field” classes that represent cotangent quantities and overload transformation behaviors to automatically apply the appropriate adjoint transforms.

Let \mathbf{f} denote a field’s values on the collocation grid and $\hat{\mathbf{f}}$ its spectral coefficients. Let \mathbf{T} denote the forward transform, such that $\hat{\mathbf{f}} = \mathbf{T}\mathbf{f}$ and $\mathbf{f} = \mathbf{T}^{-1}\hat{\mathbf{f}}$. Denoting an adjoint field as \mathbf{f}^\dagger , we require that $\langle \mathbf{f}^\dagger, \mathbf{f} \rangle = \langle \hat{\mathbf{f}}^\dagger, \hat{\mathbf{f}} \rangle$. Using the above identities, we obtain the relationships:

$$\hat{\mathbf{f}}^\dagger = \mathbf{T}^{-\dagger}\mathbf{f}^\dagger, \quad \mathbf{f}^\dagger = \mathbf{T}^\dagger\hat{\mathbf{f}}^\dagger. \quad (20)$$

This shows that the forward transform of an adjoint field is the adjoint of the original backward transform, and vice versa.

For transforms implemented directly as matrix multiplications, the adjoints are computed using the Hermitian transpose of the matrix. For fast transforms (e.g., FFTs), the adjoint is implemented using a corresponding fast transform, depending on the transform type.

Acknowledgements

We thank Steve Tobias, Jeff Oishi, Geoff Vasil, Daniel Lecoanet, Ben Brown, Rich Kerswell, Patrick Farrell, Colm Caulfield, Andre Souza, Greg Wagner, James Maddison, and Peter Schmid for helpful discussions.

This work was undertaken on ARC4, part of the High Performance Computing facilities at the University of Leeds, UK. CSS acknowledges partial support from a grant from the Simons Foundation (Grant No. 662962, GF). CSS would also like to acknowledge support of funding from the European Union Horizon 2020 research and innovation programme (grant agreement no. D5S-DLV-786780). This work was supported by a grant from the Simons Foundation (SSRFA-6826, K.J.B.).

A Problem types

Here we give further details of the adjoint approach used for each solver type.

A.1 Eigenvalue problems

Dedalus eigenvalue problems take the generalized form

$$(\lambda \mathbf{M}(\mathbf{p}) + \mathbf{L}(\mathbf{p})) \mathbf{X} = 0, \quad (21)$$

meaning the adjoint eigenvalue problem is simply

$$(\bar{\lambda} \mathbf{M}^\dagger(\mathbf{p}) + \mathbf{L}^\dagger(\mathbf{p})) \mathbf{Y} = 0. \quad (22)$$

For dense solves, the left eigenvectors and eigenvalues can be returned by the eigenvalue library routines wrapped in `scipy.linalg`. For sparse solves which find a small number of eigenmodes near a target eigenvalue, a shift-invert method is used, requiring an LU decomposition of a matrix pencil of \mathbf{M} and \mathbf{L} . For solving the adjoint eigenvalue problem, this LU decomposition can be reused, leading to significant computational savings for the adjoint solve.

Once the adjoint eigenvalue problem is solved, the sensitivity of the eigenvalue λ with respect to parameters is obtained with the equation (see [46], for example)

$$\frac{\partial \lambda}{\partial \mathbf{p}} = -\frac{\langle \mathbf{Y}, \left(\lambda \frac{\partial \mathbf{M}}{\partial \mathbf{p}} + \frac{\partial \mathbf{L}}{\partial \mathbf{p}} \right) \mathbf{X} \rangle}{\langle \mathbf{Y}, \mathbf{M} \mathbf{X} \rangle}. \quad (23)$$

The sensitivity of the eigenvector with respect to parameters is then

$$(\lambda \mathbf{M}(\mathbf{p}) + \mathbf{L}(\mathbf{p})) \frac{\partial \mathbf{X}}{\partial \mathbf{p}} = -\left(\mathbf{M} \frac{\partial \lambda}{\partial \mathbf{p}} + \lambda \frac{\partial \mathbf{M}}{\partial \mathbf{p}} + \frac{\partial \mathbf{L}}{\partial \mathbf{p}} \right) \mathbf{X}. \quad (24)$$

This is a singular equation when λ is a generalized eigenvalue, but from the Fredholm alternative theorem, it is solvable when the right hand side is orthogonal to the adjoint eigenvector \mathbf{Y} . This directly gives the eigenvalue sensitivity equation (23), and therefore there exists a unique solution for the eigenvector sensitivity that is orthogonal to \mathbf{X} . This sensitivity analysis naturally carries forward to higher order derivatives, as outlined by [54].

A.2 Linear boundary value problems

Dedalus LBVPs take the form

$$\mathbf{L} \mathbf{X} = \mathbf{F}(\mathbf{p}), \quad (25)$$

so the adjoint state equation is simply

$$\mathbf{L}^\dagger \mathbf{Y} = \mathbf{G}, \quad (26)$$

for a right-hand-side \mathbf{G} (that depends on the cost functional or upstream solves). The LU decomposition used for the forward solve can be reused for the adjoint solve. A VJP of \mathbf{F} with cotangents \mathbf{Y} then propagates sensitivities to the parameters \mathbf{p} on which \mathbf{F} depends.

A.3 Nonlinear boundary value problems

Dedalus nonlinear boundary value problems can be written as

$$\mathbf{L} \mathbf{X} = \mathbf{F}(\mathbf{X}, \mathbf{p}), \quad (27)$$

and is solved via Newton-Kantorovich iterations. The adjoint of this solve is obtained via the linearization of the final solution stage, solving

$$\underbrace{\left(\mathbf{L} - \frac{\partial \mathbf{F}}{\partial \mathbf{X}} \right)}_{\mathbf{H}}^\dagger \mathbf{Y} = \mathbf{G}, \quad (28)$$

for a right-hand-side \mathbf{G} (that depends on the cost functional or upstream solves). The factorization of \mathbf{H} from the final Newton iteration is reused for the adjoint state equation. A VJP of \mathbf{F} with cotangents \mathbf{Y} then propagates sensitivities to the parameters \mathbf{p} and state \mathbf{X} .

A.4 Initial value problems

Dedalus IVPs take the form

$$\mathbf{M} \partial_t \mathbf{X} + \mathbf{L} \mathbf{X} = \mathbf{F}(\mathbf{X}, \mathbf{p}, t), \quad (29)$$

which are either solved via a multistep IMEX scheme [89] or a Runge-Kutta IMEX scheme [3]. For both of these timestepping schemes, each step requires computing the explicit terms \mathbf{F} , solving linear systems, and updating the state \mathbf{X} . To provide the adjoint of this sequence, we have manually implemented the adjoint of the timestepping base classes (for an example of how to derive the adjoint timestepping schemes, see [51]). An s -step multistep IMEX scheme takes the form

$$(a_0^n \mathbf{M} + b_0^n \mathbf{L}) \mathbf{X}_n = \sum_{i=1}^s [c_i^n \mathbf{F}(\mathbf{X}_{n-i}) - (a_i^n \mathbf{M} + b_i^n \mathbf{L}) \mathbf{X}_{n-i}], \quad (30)$$

where the coefficients (a_i^n, b_i^n, c_i^n) depend on the timestep at step n , and revert to lower order multistep IMEX schemes for early iterations where not enough steps are computed for higher order schemes. The adjoint of this scheme is

$$(a_0^n \mathbf{M} + b_0^n \mathbf{L})^\dagger \mathbf{Y}_n = \sum_{i=1}^s \left(c_i^{n+i} \frac{\partial \mathbf{F}(\mathbf{X}_n)}{\partial \mathbf{X}_n} - (a_i^{n+i} \mathbf{M} + b_i^{n+i} \mathbf{L}) \right)^\dagger \mathbf{Y}_{n+i}. \quad (31)$$

Similarly, a Runge-Kutta IMEX scheme with s stages is

$$(\mathbf{M} + dt^n H_{i,i} \mathbf{L}) \mathbf{X}_{n,i} = \mathbf{M} \mathbf{X}_{n,0} + dt^n \sum_{j=0}^{i-1} [A_{i,j} \mathbf{F}_{n,j} - H_{i,j} \mathbf{L} \mathbf{X}_{n,j}], \quad (32)$$

with $\mathbf{X}_n = \mathbf{X}_{n,0} = \mathbf{X}_{n-1,s}$ and where dt^n is the timestep at iteration n . This has the adjoint

$$(\mathbf{M} + dt^n H_{i,i} \mathbf{L})^\dagger \mathbf{Y}_{n,i} = \begin{cases} \mathbf{Y}_{n+1,0}, & \text{if } i = s, \\ \sum_{j=i+1}^s dt^n \left(A_{j,i} \frac{\partial \mathbf{F}_{n,i}}{\partial \mathbf{X}_{n,i}} - H_{j,i} \mathbf{L} \right)^\dagger \mathbf{Y}_{n,j}, & \text{if } 0 < i < s, \end{cases} \quad (33)$$

with

$$\mathbf{Y}_{n,0} = \sum_{j=1}^s \left(\mathbf{M}^\dagger + dt^n \left[A_{j,0} \frac{\partial \mathbf{F}_{n,0}}{\partial x_{n,0}} - H_{j,0} \mathbf{L} \right]^\dagger \right) \mathbf{Y}_{n,j}, \quad (34)$$

and where $\mathbf{Y}_n = \mathbf{Y}_{n,0} = \mathbf{Y}_{n-1,s}$.

In both cases we see that an adjoint step requires VJPs of \mathbf{F} with cotangents, solving linear systems given via transposes of the forward problem matrices, and updating the cotangents. The LU decompositions in the direct solve loop are reused where possible, i.e. unless they must be rebuilt due to a change in the timestep. The VJPs of \mathbf{F} are performed using the automatic differentiation technique discussed in the manuscript. As the state at each iteration of the forward solve is required to compute VJPs in the adjoint solve, checkpointing is used using the `checkpoint_schedules` library [15], which includes many state-of-the-art checkpointing schemes [76, 5, 4, 67, 27, 48]. This allows the user to easily choose between memory and disk based checkpointing.

The adjoint timestepping schemes propagate sensitivities backwards in time from the cotangents defined at the final-time. These final-time-condition cotangents stem from VJPs with upstream calculations which can be the cost functional, or other solvers such as LBVPs, NLVPs or other IVPs. Although it seems that this formulation only provides the sensitivities with respect to each direct state \mathbf{X}_n , we can obtain the sensitivity with respect to constant parameters \mathbf{p} by adding them to the state \mathbf{X} . Similarly, if the cost functional depends on a quantity \mathcal{J}_I integrated over t , we can include this in the equation set by including the ODE $\partial_t \mathcal{J}_I = \mathcal{J}_I$. This means that the integrated quantity at the final iteration \mathcal{J}_n is obtained with the same accuracy as the forward integration, and its influence on the cotangent calculation is now naturally included through the adjoint timestepping routine.

B Example implementation details

Here we give further details of the implementations for each example.

B.1 Parametric sensitivity and numerical continuation

The problem is discretized in the wall-normal direction with Chebyshev polynomials with 256 modes. The streamwise and spanwise directions are parameterized by wavenumbers α and β . A sparse eigenvalue solve is performed with the `scipy.sparse.eigs` routine which wraps ARPACK.

B.2 Nonlinear optimization

The ball is discretized with spin-weighted spherical harmonics in the longitudinal and latitudinal dimensions and one-sided Jacobi polynomials in the radial direction [88]. We use spherical harmonics up to degree $\ell = 15$, and a radial resolution up to polynomials of maximum degree 31. Nonlinear terms are dealiased using a 3/2-dealiasing rule. A second order semi-implicit multistep BDF scheme is used with fixed timestep $\Delta t = 5 \times 10^{-4}$. We have checked that increasing the resolution and decreasing the timestep does not significantly affect the results. The direct solution is provided by solving an LBVP to obtain \mathbf{u} from $\boldsymbol{\omega}$, followed by an IVP to evolve either \mathbf{A} or \mathbf{B} . The adjoint solve requires propagating cotangents back from the cost functional through the IVP and then through the LBVP.

B.3 Resolvent analysis

The radial direction in the disk is discretized with Zernike polynomials [87] up to a maximum degree of 127, with the annular and spanwise directions parameterized by wavenumbers m and k . A spectrally accurate weight matrix \mathbf{W} that discretizes the L^2 -norm is generated with Zernike quadrature. The action of the resolvent matrix \mathcal{H} on a forcing vector $\hat{\mathbf{f}}$ is obtained via an LBVP. The optimal basis that maximizes the Rayleigh quotient

$$\sigma = \frac{\|\mathcal{H}\hat{\mathbf{f}}\|_{\mathbf{W}}}{\|\hat{\mathbf{f}}\|_{\mathbf{W}}}, \quad (35)$$

is found by computing the singular value decomposition (SVD) of $\mathbf{R}_M = M^{-1}HM$, where M is the Cholesky decomposition of the weight matrix, $\mathbf{W} = M^\dagger M$ (trivial here since \mathbf{W} is diagonal). Finding the SVD requires matrix-free application routines for both \mathcal{H}_M and \mathcal{H}_M^\dagger , the latter of which requires the adjoint of the LBVP.

B.4 Phase reduction analysis

Time is discretized with a complex-Fourier basis with 512 modes. Dealiasing is performed on a grid twice the size to properly account for cubic nonlinearities.

C Verification

In this section we perform verification tests to ensure that our computed gradients are consistent with the direct problem. For the plane-Poiseuille and optimal dynamo examples, we use the Taylor-remainder test (see [19], for example). This test checks that

$$\mathcal{J}(\mathbf{p} + \epsilon\mathbf{p}') = \mathcal{J}(\mathbf{p}) + \epsilon\langle \mathbf{Y}, \mathbf{p}' \rangle + \mathcal{O}(\epsilon^2), \quad (36)$$

where \mathbf{p}' is a perturbation, and \mathbf{Y} is the computed cotangent. This test confirms the correctness of \mathbf{Y} by showing that

$$|\mathcal{J}(\mathbf{p} + \epsilon\mathbf{p}') - \mathcal{J}(\mathbf{p}) - \epsilon\langle \mathbf{Y}, \mathbf{p}' \rangle| \approx 0, \quad (37)$$

and decays at a second order in ϵ . This provides a simple and effective test for verifying cotangents and can be applied for linear and nonlinear problems. The results of these Taylor-remainder tests are shown in table 1, showing excellent accuracy of the computed gradients. We also plot the remainders for a range of values of ϵ in Fig. 6, showing that the remainders decay at second order over a range of perturbation sizes.

For the linear tests, direct error metrics are shown in Table 2, and again indicate that the discrete adjoint is computed correctly. For the pipe-flow example, we use the fact that \mathcal{H}_M is a linear operator to directly verify the adjoint relation

$$\langle \mathbf{Y}, \mathcal{H}_M \mathbf{X} \rangle = \langle \mathcal{H}_M^\dagger \mathbf{Y}, \mathbf{X} \rangle. \quad (38)$$

For the FitzHugh-Nagumo equation, we check the tangential phase tendency

$$Z_u \frac{du_0}{dt} + Z_v \frac{dv_0}{dt} = 1, \quad (39)$$

for all times. This test requires that the phase sensitivity function is the gradient of the phase function $\Phi((u, v)^T)$ evaluated on the limit cycle [18].

Problem	Taylor-remainder
Plane-Poiseuille	1.999
Optimal dynamo (I)	2.016
Optimal dynamo (II)	1.993

Table 1: Taylor-remainder tests demonstrating the correctness of the discrete adjoint calculations. For the optimal dynamo problem two setups were tested. Setup I: a second-order IMEX BDF scheme with no checkpointing. Setup II: a second-order Runge-Kutta IMEX scheme using the “H-revolve” checkpointing schedule [27] with 400 checkpoints in RAM and 50 checkpoints on disk.

Problem	Error
Pipe-flow	1.05×10^{-14}
FitzHugh-Nagumo	2.15×10^{-13}

Table 2: Direct errors for the linear adjoint examples. For the pipe-flow problem, we report the inner-product test error. For the FitzHugh-Nagumo problem, we report the L^∞ error over time of the tangential phase tendency.

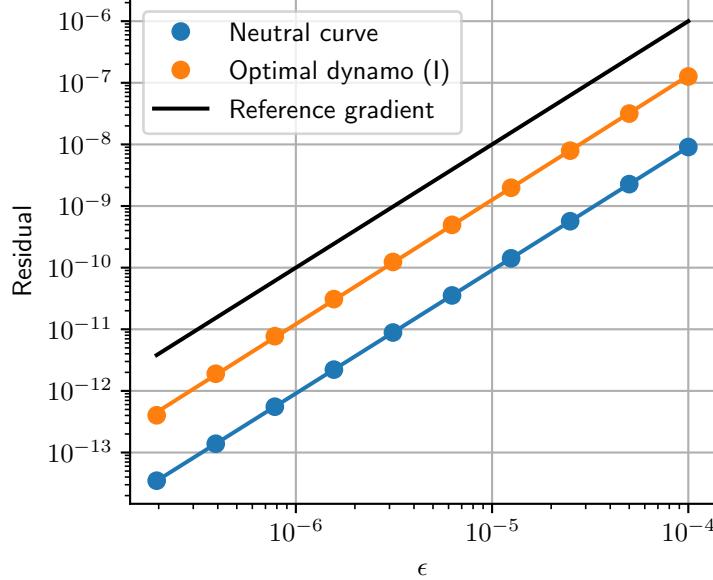


Figure 6: Taylor remainders demonstrating the convergence of finite difference estimates of the adjoint to our automatically computed values. Proper implementation of the discrete adjoint results in second-order convergence (black line).

References

- [1] M. S. Alnaes, A. Logg, K. B. Ølgaard, M. E. Rognes, and G. N. Wells. “Unified Form Language: A domain-specific language for weak formulations of partial differential equations”. In: *ACM Transactions on Mathematical Software* 40 (2014).
- [2] E. H. Anders et al. “The photometric variability of massive stars due to gravity waves excited by core convection”. In: *Nature Astronomy* 7.10 (2023), pp. 1228–1234.
- [3] U. M. Ascher, S. J. Ruuth, and R. J. Spiteri. “Implicit-explicit Runge-Kutta methods for time-dependent partial differential equations”. In: *Applied Numerical Mathematics* 25.2 (1997). Special Issue on Time Integration, pp. 151–167.
- [4] G. Aupy and J. Herrmann. “Periodicity in optimal hierarchical checkpointing schemes for adjoint computations”. In: *Optimization Methods and Software* 32.3 (2017), pp. 594–624.
- [5] G. Aupy, J. Herrmann, P. Hovland, and Y. Robert. “Optimal Multistage Algorithm for Adjoint Computation”. In: *SIAM Journal on Scientific Computing* 38.3 (2016), pp. C232–C255.
- [6] D. A. Bezgin, A. B. Buhendwa, and N. A. Adams. “JAX-Fluids 2.0: Towards HPC for differentiable CFD of compressible two-phase flows”. In: *Computer Physics Communications* 308 (Mar. 2025), p. 109433.
- [7] D. A. Bezgin, A. B. Buhendwa, and N. A. Adams. “JAX-Fluids: A fully-differentiable high-order computational fluid dynamics solver for compressible two-phase flows”. In: *Computer Physics Communications* 282 (Jan. 2023), p. 108527.
- [8] N. Bouziani and D. A. Ham. *Physics-driven machine learning models coupling PyTorch and Firedrake*. 2023. arXiv: 2303.06871.
- [9] J. P. Boyd. *Chebyshev and Fourier spectral methods*. Courier Corporation, 2001.
- [10] J. Bradbury et al. *JAX: composable transformations of Python+NumPy programs*. Version 0.3.13. 2018.
- [11] K. J. Burns, D. Fortunato, K. Julien, and G. M. Vasil. “Corner Cases of the Generalized Tau Method”. In: *preprint* (2022).
- [12] K. J. Burns, G. M. Vasil, J. S. Oishi, D. Lecoanet, and B. P. Brown. “Dedalus: A flexible framework for numerical simulations with spectral methods”. In: *Physical Review Research* 2.2, 023068 (Apr. 2020), p. 023068.
- [13] L. Chen, W. Herreman, K. Li, P. W. Livermore, J. W. Luo, and A. Jackson. “The optimal kinematic dynamo driven by steady flows in a sphere”. In: *Journal of Fluid Mechanics* 839 (2018), pp. 1–32.
- [14] F.-X. L. Dimet and O. Talagrand. “Variational algorithms for analysis and assimilation of meteorological observations: theoretical aspects”. In: *Tellus A* 38A.2 (1986), pp. 97–110.
- [15] D. I. Dolci, J. R. Maddison, D. A. Ham, G. Pallez, and J. Herrmann. “checkpoint_schedules: schedules for incremental checkpointing of adjoint simulations”. In: *Journal of Open Source Software* 9.95 (2024), p. 6148.
- [16] G. Dresdner et al. *Learning to correct spectral methods for simulating turbulent flows*. 2022.
- [17] A. C. Ellison, K. Julien, and G. M. Vasil. “A gyroscopic polynomial basis in the sphere”. In: *Journal of Computational Physics* 460 (2022), p. 111170.
- [18] G. B. Ermentrout and D. Terman. *Mathematical Foundations of Neuroscience*. Vol. 35. Interdisciplinary Applied Mathematics. New York, NY: Springer, 2010.
- [19] P. E. Farrell, D. A. Ham, S. W. Funke, and M. E. Rognes. “Automated Derivation of the Adjoint of High-Level Transient Finite Element Programs”. In: *SIAM Journal on Scientific Computing* 35.4 (2013), pp. C369–C393.
- [20] R. Fitzhugh. “Impulses and Physiological States in Theoretical Models of Nerve Membrane.” In: *Biophys J* 1.6 (July 1961), pp. 445–466.
- [21] G. Floquet. “Sur les équations différentielles linéaires à coefficients périodiques”. In: *Annales scientifiques de l’École Normale Supérieure* 12 (1883), pp. 47–88.
- [22] K. Giannakoglou, E. Papoutsis-Kiachagias, and K. Gkaragkounis. *adjointOptimisationFoam*, an OpenFOAM-based optimisation tool. the Parallel CFD & Optimization Unit, School of Mechanical Engineering, National Technical University of Athens. June 2019.
- [23] M. B. Giles and N. A. Pierce. “An introduction to the adjoint approach to design”. In: *Flow, turbulence and combustion* 65 (2000), pp. 393–415.
- [24] V. Godavarthi, Y. Kawamura, and K. Taira. “Optimal waveform for fast synchronization of airfoil wakes”. In: *Journal of Fluid Mechanics* 976 (2023), R1.
- [25] D. A. Ham et al. *Firedrake User Manual*. First edition. Imperial College London et al. May 2023.
- [26] L. Hascoet and V. Pascual. “The Tapenade Automatic Differentiation tool: principles, model, and specification”. In: *ACM Transactions on Mathematical Software* 39.3 (2013).
- [27] J. Herrmann and G. P. Aupy. “H-Revolve: A Framework for Adjoint Computation on Synchronous Hierarchical Platforms”. In: *ACM Trans. Math. Softw.* 46.2 (June 2020).
- [28] G. W. Hill. “On the part of the motion of the lunar perigee which is a function of the mean motions of the sun and moon”. In: *Acta Mathematica* 8.1 (1886), pp. 1–36.
- [29] A. L. Hodgkin and A. F. Huxley. “A quantitative description of membrane current and its application to conduction and excitation in nerve.” In: *The Journal of physiology* 117.4 (Aug. 1952), pp. 500–544.
- [30] P. Holl and N. Thurey. “ΦFlow (PhiFlow): Differentiable Simulations for PyTorch, TensorFlow and Jax”. In: *International Conference on Machine Learning*. PMLR. 2024.
- [31] T. Ichinomiya. “Frequency synchronization in a random oscillator network”. In: *Phys. Rev. E* 70 (2 Aug. 2004), p. 026116.
- [32] T. Isaac, N. Petra, G. Stadler, and O. Ghattas. “Scalable and efficient algorithms for the propagation of uncertainty from data through inference to prediction for large-scale problems, with application to flow of the Antarctic ice sheet”. In: *Journal of Computational Physics* 296 (2015), pp. 348–368.

- [33] J. A. Jackson et al. “Scaling behaviour and control of nuclear wrinkling”. In: *Nature physics* 19.12 (2023), pp. 1927–1935.
- [34] A. Jameson. “Aerodynamic Shape Optimization”. In: *Computational Aerodynamics*. Cambridge Aerospace Series. Cambridge University Press, 2022, pp. 465–495.
- [35] Y. Kawamura, V. Godavarthi, and K. Taira. “Adjoint-based phase reduction analysis of incompressible periodic flows”. In: *Phys. Rev. Fluids* 7 (10 Oct. 2022), p. 104401.
- [36] R. Kerswell. “Nonlinear Nonmodal Stability Theory”. In: *Annual Review of Fluid Mechanics* 50. Volume 50, 2018 (2018), pp. 319–345.
- [37] D. Kochkov, J. A. Smith, A. Alieva, Q. Wang, M. P. Brenner, and S. Hoyer. “Machine learning-accelerated computational fluid dynamics”. In: *Proceedings of the National Academy of Sciences* 118.21 (2021).
- [38] F. Koehler, S. Niedermaier, R. Westermann, and N. Thuerey. “APEBench: A Benchmark for Autoregressive Neural Emulators of PDEs”. In: *Advances in Neural Information Processing Systems (NeurIPS)* 38 (2024).
- [39] Y. Kuramoto. *Chemical Oscillations, Waves, and Turbulence*. Berlin: Springer-Verlag, 1984.
- [40] M. Landreman, B. Medasani, F. Wechsung, A. Giuliani, R. Jorge, and C. Zhu. “SIMSOPT: A flexible framework for stellarator optimization”. In: *Journal of Open Source Software* 6.65 (2021), p. 3525.
- [41] D. Lecoanet and R. R. Kerswell. “Connection between nonlinear energy optimization and instantons”. In: *Phys. Rev. E* 97 (1 Jan. 2018), p. 012212.
- [42] D. Lecoanet, G. M. Vasil, K. J. Burns, B. P. Brown, and J. S. Oishi. “Tensor calculus in spherical coordinates using Jacobi polynomials. Part-II: implementation and examples”. In: *Journal of Computational Physics: X* 3 (2019), p. 100012.
- [43] I. A. Loe, H. Nakao, Y. Jimbo, and K. Kotani. “Phase-reduction for synchronization of oscillating flow by perturbation on surrounding structure”. In: *Journal of Fluid Mechanics* 911 (2021), R2.
- [44] A. Logg, G. Wells, and K.-A. Mardal. *Automated Solution of Differential Equations by the Finite Element Method: The FEniCS Book*. Vol. 84. Apr. 2011.
- [45] A. C. Lorenc. “Analysis methods for numerical weather prediction”. In: *Quarterly Journal of the Royal Meteorological Society* 112.474 (1986), pp. 1177–1194.
- [46] P. Luchini and A. Bottaro. “Adjoint Equations in Stability Analysis”. In: *Annual Review of Fluid Mechanics* 46. Volume 46, 2014 (2014), pp. 493–517.
- [47] Y. Ma, V. Dixit, M. J. Innes, X. Guo, and C. Rackauckas. “A Comparison of Automatic Differentiation and Continuous Sensitivity Analysis for Derivatives of Differential Equation Solutions”. In: *2021 IEEE High Performance Extreme Computing Conference (HPEC)*. 2021, pp. 1–9.
- [48] J. R. Maddison. “Step-based checkpointing with high-level algorithmic differentiation”. In: *Journal of Computational Science* 82 (2024), p. 102405.
- [49] J. R. Maddison, D. N. Goldberg, and B. D. Goddard. “Automated Calculation of Higher Order Partial Differential Equation Constrained Derivative Information”. In: *SIAM Journal on Scientific Computing* 41.5 (2019), pp. C417–C445.
- [50] L. Magri. “Adjoint Methods as Design Tools in Thermoacoustics”. In: *Applied Mechanics Reviews* 71.2 (Mar. 2019).
- [51] P. M. Mannix, C. S. Skene, D. Auroux, and F. Marcotte. “A robust, discrete-gradient descent procedure for optimisation with time-dependent PDE and norm constraints”. In: *The SMAI Journal of computational mathematics* 10 (2024), pp. 1–28.
- [52] B. J. McKeon, J. Li, W. Jiang, J. F. Morrison, and A. J. Smits. “Further observations on the mean velocity distribution in fully developed pipe flow”. In: *Journal of Fluid Mechanics* 501 (2004), pp. 135–147.
- [53] B. J. McKeon and A. S. Sharma. “A critical-layer framework for turbulent pipe flow”. In: *Journal of Fluid Mechanics* 658 (2010), pp. 336–382.
- [54] G. A. Mensah, A. Orchini, and J. P. Moeck. “Perturbation theory of nonlinear, non-self-adjoint eigenvalue problems: Simple eigenvalues”. In: *Journal of Sound and Vibration* 473 (2020), p. 115200.
- [55] S. K. Mitusch, S. W. Funke, and J. S. Dokken. “dolfin-adjoint 2018.1: automated adjoints for FEniCS and Firedrake”. In: *Journal of Open Source Software* 4.38 (2019), p. 1292.
- [56] J. Nagumo, S. Arimoto, and S. Yoshizawa. “An Active Pulse Transmission Line Simulating Nerve Axon”. In: *Proceedings of the IRE* 50.10 (1962), pp. 2061–2070.
- [57] A. G. Nair, K. Taira, B. W. Brunton, and S. L. Brunton. “Phase-based control of periodic flows”. In: *Journal of Fluid Mechanics* 927 (2021), A30.
- [58] H. Nakao. “Phase reduction approach to synchronisation of nonlinear oscillators”. In: *Contemporary Physics* 57.2 (2016), pp. 188–214.
- [59] H. Nakao, K. Arai, and Y. Kawamura. “Noise-Induced Synchronization and Clustering in Ensembles of Uncoupled Limit-Cycle Oscillators”. In: *Phys. Rev. Lett.* 98 (18 May 2007), p. 184101.
- [60] L. O’Connor et al. “Iterative methods for Navier-Stokes inverse problems”. In: *Phys. Rev. E* 109 (4 Apr. 2024), p. 045108.
- [61] S. Olver and A. Townsend. “A fast and well-conditioned spectral method”. In: *siam REVIEW* 55.3 (2013), pp. 462–489.
- [62] S. Olver, A. Townsend, and G. Vasil. “A sparse spectral method on triangles”. In: *SIAM Journal on Scientific Computing* 41.6 (2019), A3728–A3756.
- [63] S. A. Orszag. “Accurate solution of the Orr–Sommerfeld stability equation”. In: *Journal of Fluid Mechanics* 50.4 (1971), pp. 689–703.

- [64] F. Palacios et al. “Stanford University Unstructured (SU²): An open-source integrated computational environment for multi-physics simulation and design”. In: *51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*. 2013.
- [65] E. J. Paul, I. G. Abel, M. Landreman, and W. Dorland. “An adjoint method for neoclassical stellarator optimization”. In: *Journal of Plasma Physics* 85.5 (2019), p. 795850501.
- [66] R.-E. Plessix. “A review of the adjoint-state method for computing the gradient of a functional with geophysical applications”. In: *Geophysical Journal International* 167.2 (Nov. 2006), pp. 495–503.
- [67] G. Pringle, D. Jones, S. Goswami, S. Narayanan, and D. Goldberg. *Providing the ARCHER community with adjoint modelling tools for high-performance oceanographic and cryospheric computation*. Tech. rep. Oct. 2016.
- [68] M. Proctor. “Energy requirement for a working dynamo”. In: *Geophysical & Astrophysical Fluid Dynamics* 109.6 (2015), pp. 611–614.
- [69] C. Rackauckas et al. *Universal Differential Equations for Scientific Machine Learning*. 2021. arXiv: 2001.04385.
- [70] H. Ranocha, M. Schlottke-Lakemper, A. R. Winters, E. Faulhaber, J. Chan, and G. J. Gassner. “Adaptive numerical simulations with Trixi.jl: A case study of Julia for scientific computing”. In: *Proceedings of the JuliaCon Conferences 1.1* (2022), p. 77.
- [71] L. V. Rolandi, J. H. M. Ribeiro, C.-A. Yeh, and K. Taira. “An invitation to resolvent analysis”. In: *Theoretical and Computational Fluid Dynamics* 38.5 (2024), pp. 603–639.
- [72] P. J. Schmid and D. S. Henningson. *Stability and Transition in Shear Flows*. Vol. 142. Applied Mathematical Sciences. New York, NY: Springer-Verlag, Jan. 2001.
- [73] P. J. Schmid. “Nonmodal Stability Theory”. In: *Annual Review of Fluid Mechanics* 39. Volume 39, 2007 (2007), pp. 129–162.
- [74] C. S. Skene, F. Marcotte, and S. M. Tobias. *On nonlinear transitions, minimal seeds and exact solutions for the geodynamo*. 2024. arXiv: 2411.05499.
- [75] C. S. Skene and K. Taira. “Phase-reduction analysis of periodic thermoacoustic oscillations in a Rijke tube”. In: *Journal of Fluid Mechanics* 933 (2022), A35.
- [76] P. Stumm and A. Walther. “MultiStage Approaches for Optimal Offline Checkpointing”. In: *SIAM Journal on Scientific Computing* 31.3 (2009), pp. 1946–1967.
- [77] K. Taira and H. Nakao. “Phase-response analysis of synchronization for periodic flows”. In: *Journal of Fluid Mechanics* 846 (2018), R2.
- [78] K. Taira et al. “Modal Analysis of Fluid Flows: An Overview”. In: *AIAA Journal* 55.12 (2017), pp. 4013–4041.
- [79] K. Taira et al. “Modal Analysis of Fluid Flows: Applications and Outlook”. In: *AIAA Journal* 58.3 (2020), pp. 998–1022.
- [80] W. C. Thacker. “The role of the Hessian matrix in fitting models to measurements”. In: *Journal of Geophysical Research: Oceans* 94.C5 (1989), pp. 6177–6196.
- [81] S. M. Tobias. “The turbulent dynamo”. In: *Journal of Fluid Mechanics* 912 (2021), P1.
- [82] J. Townsend, N. Koep, and S. Weichwald. “Pymanopt: A Python Toolbox for Optimization on Manifolds using Automatic Differentiation”. In: *Journal of Machine Learning Research* 17.137 (2016), pp. 1–5.
- [83] L. N. Trefethen, A. E. Trefethen, S. C. Reddy, and T. A. Driscoll. “Hydrodynamic Stability Without Eigenvalues”. In: *Science* 261.5121 (1993), pp. 578–584.
- [84] G. M. Vasil, K. J. Burns, D. Lecoanet, S. Olver, B. P. Brown, and J. S. Oishi. “Tensor calculus in polar coordinates using Jacobi polynomials”. In: *Journal of Computational Physics* 325 (2016), pp. 53–73.
- [85] G. M. Vasil, D. Lecoanet, K. J. Burns, J. S. Oishi, and B. P. Brown. “Tensor calculus in spherical coordinates using Jacobi polynomials. Part-I: mathematical analysis and derivations”. In: *Journal of Computational Physics: X* 3 (2019), p. 100013.
- [86] G. M. Vasil et al. “The solar dynamo begins near the surface”. In: *Nature* 629.8013 (2024), pp. 769–772.
- [87] G. M. Vasil, K. J. Burns, D. Lecoanet, S. Olver, B. P. Brown, and J. S. Oishi. “Tensor calculus in polar coordinates using Jacobi polynomials”. In: *Journal of Computational Physics* 325 (2016), pp. 53–73.
- [88] G. M. Vasil, D. Lecoanet, K. J. Burns, J. S. Oishi, and B. P. Brown. “Tensor calculus in spherical coordinates using Jacobi polynomials. Part-I: Mathematical analysis and derivations”. In: *Journal of Computational Physics: X* 3 (2019), p. 100013.
- [89] D. Wang and S. J. Ruuth. “Variable step-size implicit-explicit linear multistep methods for time-dependent partial differential equations”. In: *Journal of Computational Mathematics* 26.6 (2008), pp. 838–855.
- [90] N. P. Wedi, M. Hamrud, and G. Mozdzynski. “A fast spherical harmonics transform for global NWP and climate models”. In: *Monthly Weather Review* 141.10 (2013), pp. 3450–3461.
- [91] A. T. Winfree. *The Geometry of Biological Time*. Vol. 12. Interdisciplinary Applied Mathematics. New York, NY: Springer-Verlag, 2001.
- [92] P. Yeung, K. Ravikumar, S. Nichols, and R. Uma-Vaideswaran. “GPU-enabled extreme-scale turbulence simulations: Fourier pseudo-spectral algorithms at the exascale using OpenMP offloading”. In: *Computer Physics Communications* 306 (2025), p. 109364.
- [93] H. Zhang, E. M. Constantinescu, and B. F. Smith. “PETSc TSAdjoint: A Discrete Adjoint ODE Solver for First-Order and Second-Order Sensitivity Analysis”. In: *SIAM Journal on Scientific Computing* 44.1 (2022), pp. C1–C24.