Structural and Multidisciplinary Optimization (2022) 65:28
https://doi.org/10.1007/s00158-021-03117-5

RESEARCH PAPER

Discrete adjoint methodology for general multiphysics problems

A modular and efficient algorithmic outline with implementation in an open-source simulation software

Ole Burghardt[1] · Pedro Gomes[2] · Tobias Kattmann[3] · Thomas D. Economon[4] · Nicolas R. Gauger[1] · Rafael Palacios[2]

Received: 26 January 2021 / Revised: 11 October 2021 / Accepted: 13 October 2021 / Published online: 6 January 2022

Abstract
This article presents a methodology whereby adjoint solutions for partitioned multiphysics problems can be computed efficiently, in a way that is completely independent of the underlying physical sub-problems, the associated numerical solution methods, and the number and type of couplings between them. By applying the reverse mode of algorithmic differentiation to each discipline, and by using a specialized recording strategy, diagonal and cross terms can be evaluated individually, thereby allowing different solution methods for the generic coupled problem (for example block-Jacobi or block-Gauss-Seidel). Based on an implementation in the open-source multiphysics simulation and design software SU2, we demonstrate how the same algorithm can be applied for shape sensitivity analysis on a heat exchanger (conjugate heat transfer), a deforming wing (fluid–structure interaction), and a cooled turbine blade where both effects are simultaneously taken into account.

Keywords Discrete adjoints · Multiphysics · Sensitivity analysis · Algorithmic differentiation · Fluid–structure interaction · Conjugate heat transfer

1 Introduction

In engineering sciences, especially those concerned with fluid flow problems, one finds important cases where multiphysics effects play a crucial role in the design of new parts. In the context of partitioned simulation approaches, and from a software perspective, this means that two or more solvers are coupled by exchanging data at common physical boundaries during the simulation. Finding superior designs via numerical optimization is a question of establishing suitable design goals and using software that can facilitate the procedure to fulfill them. For optimization methods relying on discrete adjoint solutions, the couplings between the different physics models must be considered to obtain accurate sensitivities. In the present study on adjoint approaches, we are interested in design processes that are closely linked to finding optimal values for an objective function, denoted by $\tilde{J}$, by varying some set of design parameters (x) with support of the total derivative $\frac{\mathrm{d}}{\mathrm{d}x}\tilde{J}$.

Furthermore, we assume that values of $\tilde{J}$ are computed from x and associated simulation results, generally solutions (u) to PDE problems that model the physical phenomena of interest. These solutions can be said to depend only on the design parameters since they, and all other derived quantities, are uniquely defined by the parameters, i.e., $u = u(x)$.

However, in deriving the adjoint solution method, it is useful to consider the function of interest as a function of the parameters and the solution. We denote this by defining an objective function $J : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}$ and setting

$$\tilde{J}(x) := J(x, u(x))$$

Responsible Editor: Emilio Carlos Nelli Silva

Ole Burghardt
ole.burghardt@scicomp.uni-kl.de

[1] Chair for Scientific Computing, TU Kaiserslautern, 67663 Kaiserslautern, Germany

[2] Department of Aeronautics, Imperial College London, London SW7 2AZ, UK

[3] Bosch Corporate Research—Fluid Dynamics, 71272 Renningen, Germany

[4] SU2 Foundation, San Francisco, CA 94158, USA

accordingly, where m is the number of design parameters and n is the number of components, formally the number of unknowns in the discretized PDE problem. Naive differentiation[1] of $\tilde{J}$ at x by application of the chain rule yields

$$\frac{\mathrm{d}}{\mathrm{d}x}\tilde{J}(\mathrm{x}) = \frac{\partial}{\partial x}J(\mathrm{x},\mathrm{u}) + \frac{\partial J}{\partial u}(\mathrm{x},\mathrm{u})\ \frac{\mathrm{d}}{\mathrm{d}x}u(\mathrm{x}), \tag{1}$$

which is a computationally intractable problem for large m, as the last term in (1) requires differentiation of the converged solution with respect to all design parameters.

Adjoint methods from control theory (Lions 1971) can efficiently capture the implicit dependence of the function on the solution variables without having to evaluate the computationally expensive term $\frac{\mathrm{d}}{\mathrm{d}x}u(\mathrm{x})$. In general, (1) is manipulated using some form of the PDE equations (also called constraints). One way to achieve this will be detailed in Sect. 2.

In the context of computational fluid dynamics, the method was first used by Jameson (1988, 1995) who employed adjoint solutions for certain (potential) flow solution constraints through the notion of a Lagrange multiplier. The method has since evolved in two major directions; in the continuous approach where the adjoint is a solution to a PDE derived from the primal PDE, whereas in the discrete approach, the adjoint is based on the discrete PDE solver. Both have shown to provide accurate gradients even for complex flow problems.

For multiphysics problems, the development of continuous adjoint methods is an attractive area of research, but by definition, any such method must consider the specific physics. In Feppon et al. (2019), a coupled adjoint PDE system is developed—using Hadamard's method—for cases with a combination of conjugate heat transfer (CHT) and fluid–structure interaction (FSI) between a fluid and a solid domain. Although no studies on the accuracy of derived sensitivities are given, numerical examples indicate that even triple physics cases can be properly handled with continuous adjoints. On the discrete adjoint side, such a framework for simultaneous FSI and CHT has recently been developed (Smith et al. 2021) which is especially designed for problems that require the coupling between a flow and a thermoelastic solver, e.g., for design aspects of hypersonic vehicles [see also the subsequent study in Kamali et al. (2020)]. It builds on hand-coded discrete adjoints included in FUN3D (Biedron et al. 2019), derived from the PDE solvers in a residual-based formulation. The authors show accurate gradients obtained from adjoint vectors that are the solution to a 9×9 (or 6×6 in case of pure CHT) block matrix equation where all blocks have been assembled individually to account for the solvers, the necessary transfer routines between them, and FSI mesh deformations. For a pure CHT problem, another approach using a residual-based formulation for the PDE solvers and subsequent derivation of the discrete adjoint equations is presented in Makhija and Beran (2019), it also incorporates density-based topology optimization variables into the problem setting, which in the SU2 framework is only currently supported for FSI problems (Gomes and Palacios 2020).

Indeed, the differentiation of multiphysics solvers for topology optimization is common (Dunning et al. 2015; Lundgaard et al. 2018; Picelli et al. 2020). However, in such applications solvers tend to be of the monolithic type (for example so that locations of the domain can be either fluid or solid), or the discrete adjoint methodology is developed specifically for the primal methods used. The coupling of flow and heat adjoint solvers has also been explored in a preliminary study to this paper (Burghardt et al. 2019), and in an implementation in the open-source software OpenFOAM (Weller et al. 1998), for analyzing temperature sensitivities in a heat sink by using algorithmic differentiation [in short: AD; Griewank and Walther (2008)] of the entire CHT solver (and not just the residual of the discretization) and a checkpointing strategy to reduce memory usage (Towara et al. 2019). In this work, AD is also applied to entire solvers, but the main strategy used to reduce memory usage is preaccumulation (Albring et al. 2016; Sagebaum et al. 2019). These AD-based "whole solver" approaches allow a generic treatment of solvers, for which the source code is available, since they do not require the tailor-made preconditioning strategies used in residual-based discrete adjoint methods.

Although numerous examples of purpose-built discrete adjoint methods could be found in the literature, a flexible approach, adaptable to multiple kinds of partitioned multiphysics problems, has not been developed so far. By which is meant problems that are defined by combining different physical domains, hereafter referred to as *zones*, through an exchange of interface data. Note that this does not preclude different zones from overlapping, moreover the interface need not be a surface (but generally it is). In general, we assume that each zone is governed by a system of PDE's; however, this is not strictly required insofar as a fixed-point iteration can be identified for the zone. All zones have one or more *solvers* associated, for fluid dynamics we are most commonly dealing with Navier-Stokes or RANS equations but one can also think of solid zones where we are concerned with elastic deformations or heat conduction. The solvers within a zone are typically tightly coupled and exchange information over the entire domain. For clarity, a monolithic FSI or CHT solver as used for topology optimization [e.g., as in Lundgaard et al. (2018)] would be classified as one solver and thus occupy one zone. However, when the part of the domain being

[1] We use upright symbols to indicate a particular value of the parameters or solution.

designed is much smaller than the surrounding fluid volume, it could be reasonable to model the problem as two partially overlapping zones as done in Picelli et al. (2020), and then track the interface as it develops. Such strategies are not yet available in the SU2 framework due to its aeronautical background, characterized by high Reynolds number applications for which body-fitted meshes and shape optimization are more common than immersed boundary and topology optimization. With shape optimization using mesh morphing, a high quality surface mesh is easier to maintain which is crucial in many fluid mechanics applications. Nevertheless, the discrete adjoint method described in this paper would be applicable whenever two or more zones (as described above) are involved.

In Sect. 2 the fixed-point concept for multiphysics problems is introduced, and then used to derive the generic multiphysics adjoint approach. Section 3 deals with implementation details of the proposed algorithm in SU2. Finally, numerical examples are presented in Sect. 4, these are also used to analyze the physical meaning of the abstract terms introduced in Sect. 2, to facilitate comparing the proposed generic method with physics-specific methods found in the literature.

2 Multiphysics discrete adjoints

Let us denote the (e.g., PDE) solvers by a function

$$\mathcal{G}(\mathrm{x}, u) : \mathbb{R}^n \to \mathbb{R}^n \tag{2}$$

that maps an intermediate solution u^i to u^{i+1}, where $\mathrm{x} \in \mathbb{R}^m$ denotes an arbitrary but fixed design parameter vector (usually the mesh coordinates unless otherwise specified).

It is assumed that the solution $\mathrm{u} = u(\mathrm{x})$ is given by the fixed point of $\mathcal{G}$, this incurs no loss of generality since common primal solution methods can easily be put in this form, which may not be trivial for the residual-based adjoint formulation [consider for example predictor-corrector methods in fluid dynamics for incompressible flows, e.g., He et al. (2018)]. In a practical sense, u is the first state for which a suitable convergence criteria is met, for example

$$\|\mathcal{G}(\mathrm{x}, \mathrm{u}^i) - \mathrm{u}^i\| < \varepsilon_{\mathrm{tol}}$$

where $\varepsilon_{\mathrm{tol}}$ is a prescribed accuracy. Nevertheless, for all mathematical considerations it is $\mathcal{G}(\mathrm{x}, \mathrm{u}) = \mathrm{u}$. This notation can be extended to multiphysics problems by assuming a partitioned approach is used (with r the total number of zones), and regarding $\mathcal{G}(\mathrm{x}, \mathrm{u}^i)$ as the vector whose components are the iterates in each zone,

$$\begin{pmatrix} \mathrm{u}^{i+1}_{(1)} \\ \vdots \\ \mathrm{u}^{i+1}_{(r)} \end{pmatrix} = \begin{pmatrix} \mathcal{G}_{(1)}(\mathrm{x}, \mathrm{u}^i) \\ \vdots \\ \mathcal{G}_{(r)}(\mathrm{x}, \mathrm{u}^i) \end{pmatrix}. \tag{3}$$

Note that the inputs $\mathrm{u}^i = (\mathrm{u}^i_{(1)}, \dots, \mathrm{u}^i_{(r)})$ of each solver on the right hand side are (possibly) the solution iterates and parameters of all zones, as these dependencies arise from the transfer of information at common physical boundaries and in this framework there is no limitation on how the zones may be connected. $\mathcal{G}_{(j)}$ is then regarded as a fixed-point iterator with respect to the solution variables $\mathrm{u}^i_{(j)}$ and all $\mathrm{u}^i_{(k)}$ with $j \neq k$ as solver parameters, this distinction will later be used to define adjoint cross terms. This view naturally maps to the common block-Gauss-Seidel (BGS) solution approach, but in practice the solution method for the primal coupled problem is not restricted. However, note that inner BGS iterations (within each zone) are omitted but re-introduced later for the adjoint method.

Throughout this article, subscripts will indicate the zone number, whereas superscripts refer to the iteration count unless otherwise specified.

2.1 Generic coupled discrete adjoint algorithm

In the context of discrete adjoints, well-known derivations are based on the *residual operator* (see the Introduction chapter in Giles and Pierce (1997), “Duality formulation for adjoint design” in Giles and Pierce (2000) or “Direct and Adjoint Derivative Equations” in Martins and Ning (2021) for a simple approach leading to (4) and the desired gradient representation (7), or the “Adjoint Approach” chapter in Hinze et al. (2008) for a functional-analytic view on the matter, covering the very same equations but with mathematical interpretations of all terms as dual space pairings) namely,

$$\mathcal{R}(x, u) := \mathcal{G}(x, u) - u$$

and the linear operator one obtains from differentiating w.r.t. u at (x, u), with x and the corresponding solution u—defined by $\mathcal{G}(\mathrm{x}, \mathrm{u}) = \mathrm{u}$ or $\mathcal{R}(\mathrm{x}, \mathrm{u}) = 0$—being fixed;

$$\frac{\partial \mathcal{R}}{\partial u}(\mathrm{x}, \mathrm{u}) = \frac{\partial \mathcal{G}}{\partial u}(\mathrm{x}, \mathrm{u}) - \mathrm{Id}.$$

For a given objective function J, we define the adjoint solution $\lambda \in \mathbb{R}^n$ as the solution to the *adjoint equation*

$$\frac{\partial \mathcal{R}}{\partial u}^{\mathrm{T}}(\mathrm{x}, \mathrm{u})\, \lambda = -\frac{\partial J}{\partial u}^{\mathrm{T}}(\mathrm{x}, \mathrm{u}), \tag{4}$$

where $\frac{\partial \mathcal{G}}{\partial u}^{\mathrm{T}}$ is the adjoint operator of $\frac{\partial \mathcal{G}}{\partial u}$ (the transposed in the discrete setting of this paper).

If it can be obtained, the gradient of the reduced objective function $\tilde{J}(x)$ can be represented without requiring the computationally expensive term $\frac{\mathrm{d}}{\mathrm{dx}}u(\mathrm{x})$. This is achieved by first interpreting the matrix-vector multiplication term in (1) as a dual pairing using the standard inner product of $\mathbb{R}^n$ (which we denote by "$\langle\cdot,\cdot\rangle$", turning $\frac{\partial J}{\partial u}(\mathrm{x},\mathrm{u})$ into $\frac{\partial J}{\partial u}^{\mathrm{T}}(\mathrm{x},\mathrm{u})$) and then replacing $\frac{\partial J}{\partial u}^{\mathrm{T}}(\mathrm{x},\mathrm{u})$ by the left-hand side of (4),

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}x}\tilde{J}(\mathrm{x}) &= \frac{\partial}{\partial x}J(\mathrm{x},\mathrm{u}) + \left\langle \frac{\partial J}{\partial u}^{\mathrm{T}}(\mathrm{x},\mathrm{u})\,,\, \frac{\mathrm{d}}{\mathrm{d}x}u(\mathrm{x})\right\rangle \\ &= \frac{\partial}{\partial x}J(\mathrm{x},\mathrm{u}) - \left\langle \frac{\partial \mathcal{R}}{\partial u}^{\mathrm{T}}(\mathrm{x},\mathrm{u})\,\lambda\,,\, \frac{\mathrm{d}}{\mathrm{d}x}u(\mathrm{x})\right\rangle.\end{aligned} \tag{5}$$

Using the definition of adjoint operators, the last line of (5) can be rewritten as

$$\frac{\mathrm{d}}{\mathrm{d}x}\tilde{J}(\mathrm{x}) = \frac{\partial}{\partial x}J(\mathrm{x},\mathrm{u}) - \left\langle \lambda\,,\, \frac{\partial \mathcal{R}}{\partial u}(\mathrm{x},\mathrm{u})\,\frac{\mathrm{d}}{\mathrm{d}x}u(\mathrm{x})\right\rangle \tag{6}$$

and by definition of $u(x)$, i.e., $\mathcal{R}(x,u(x))=0$ for all x and therefore

$$-\frac{\partial \mathcal{R}}{\partial u}(\mathrm{x},\mathrm{u})\,\frac{\mathrm{d}}{\mathrm{d}x}u(\mathrm{x}) = \frac{\partial \mathcal{R}}{\partial x}(\mathrm{x},\mathrm{u}),$$

we obtain (noting that $\frac{\partial}{\partial x}\mathcal{R} = \frac{\partial}{\partial x}\mathcal{G}$)

$$\frac{\mathrm{d}}{\mathrm{d}x}\tilde{J}(\mathrm{x}) = \frac{\partial}{\partial x}J(\mathrm{x},\mathrm{u}) + \lambda^{\mathrm{T}}\frac{\partial}{\partial x}\mathcal{G}(\mathrm{x},\mathrm{u}). \tag{7}$$

The adjoint equation (4) translates itself into a fixed-point iteration for λ,

$$\lambda \overset{!}{=} \frac{\partial J}{\partial u}^{\mathrm{T}}(\mathrm{x},\mathrm{u}) + \frac{\partial \mathcal{G}}{\partial u}^{\mathrm{T}}(\mathrm{x},\mathrm{u})\,\lambda, \tag{8}$$

with convergence properties similar to the primal problem (Albring et al. 2016).

For clarity, when using adjoint solutions defined in terms of the discretization residual $\hat{\mathcal{R}}$ of the primal problem directly ($\hat{\mathcal{R}} \neq \mathcal{R}$ since the program $\mathcal{G}$ typically contains $\mathcal{R}$ and the solution method), instead of the fixed-point approach above, the adjoint equations become

$$\frac{\partial \hat{\mathcal{R}}}{\partial u}^{\mathrm{T}}(\mathrm{x},\mathrm{u})\,\hat{\lambda} = -\frac{\partial J}{\partial u}^{\mathrm{T}}(\mathrm{x},\mathrm{u}), \tag{9}$$

which is a linear system for a residual-based adjoint solution $\hat{\lambda}$, whose value is different than what results from (8). Due to the poor conditioning of the residual Jacobian for flow problems, these systems require some form of preconditioning (Maute et al. 2000; Kenway et al. 2019), usually an approximate Jacobian found in the solution of the primal problem is used. Here too we consider the residual approach to be less generic as the preconditioner is problem-specific. In the fixed-point approach the preconditioner is part of the operator ($\mathcal{G}$), and thus the primal solver can be treated almost entirely like a black box. The disadvantage is the reduced flexibility in fine tuning the solution method of the adjoint equations, for example using a flexible Krylov solver to accelerate or stabilize the convergence, as done in Maute et al. (2000); Kenway et al. (2014). Notwithstanding, strongly coupled partitioned solution methods are still applicable, e.g., Gomes and Palacios (2020), at the expense of stricter requirements on the information the primal solver must expose, for example which degrees of freedom participate in interface exchanges. Furthermore, quasi-Newton methods can also be used to accelerate or stabilize the convergence of (8). In SU2 we use a least-squares approximation of the inverse Jacobian of (8), based on the IQN-ILS method (Degroote et al. 2009).

For the evaluation of each new iterate $\lambda^{(i+1)}$ of (8), the matrix-vector product $(\partial\mathcal{G}/\partial u)^{\mathrm{T}}(\mathrm{x},\mathrm{u})\,\lambda^{(i)}$ on the right hand side is evaluated through the reverse mode of AD. In a multizone context, the adjoint solution consists of the adjoint solutions of all zones, that is,

$$\lambda = (\lambda_{(1)},\ldots,\lambda_{(r)}),$$

and the fixed-point iteration update in (8) becomes a simultaneous update for all zones $k = 1\ldots r$,

$$\lambda_{(k)}^{(i+1)} = \frac{\partial J}{\partial u_{(k)}}^{\mathrm{T}}(\mathrm{x},\mathrm{u}) + \frac{\partial \mathcal{G}}{\partial u_{(k)}}^{\mathrm{T}}(\mathrm{x},\mathrm{u})\,\lambda^{(i)}. \tag{10}$$

As mentioned before for primal solution algorithms, more intricate update schemes can be proposed. Most importantly, BGS-type updates where multiple updates are carried out for one zone, keeping the adjoints of others fixed. With i being the outer iteration count, l the inner iteration count and l^* the number of repeated inner updates, this gives an alternative to (10),

$$\begin{cases} \lambda_{(k)}^{(l+1,i)} = \dfrac{\partial J}{\partial u_{(k)}}^{\mathrm{T}}(\mathrm{x},\mathrm{u}) + \dfrac{\partial \mathcal{G}_{(k)}}{\partial u_{(k)}}^{\mathrm{T}}(\mathrm{x},\mathrm{u})\,\lambda_{(k)}^{(l,i)} \\ \qquad\qquad + \displaystyle\sum_{j\neq k}\frac{\partial \mathcal{G}_{(j)}}{\partial u_{(k)}}^{\mathrm{T}}(\mathrm{x},\mathrm{u})\,\lambda_{(j)}^{(i)} \\ \lambda_{(k)}^{(i+1)} = \lambda_{(k)}^{(l^*,i)}. \end{cases} \tag{11}$$

that can lead to a more stable procedure for $l^* > 1$ and to lower solution times, especially for problems with deforming grids (note that for $\lambda^{(0,i)} := \lambda^{(i)}$ and $l^* = 1$, (10) is recovered). Also for stability, it is sometimes beneficial to relax the update of the cross terms ($j \neq k$ summation). In Sect. 3.2 we detail how the cross terms are managed to allow this relaxation using minimal storage and computation.

Regarding convergence of the method, the upper row in (11) that maps $\lambda_{(k)}^{(l,i)}$ to $\lambda_{(k)}^{(l,i+1)}$ is linear in terms of $\lambda_{(k)}$ by definition. Denoting this mapping by $\mathcal{N}_{(k)}$, we see that

$$\left\|\frac{\partial \mathcal{N}_{(k)}}{\partial \lambda_{(k)}}(\lambda_{(k)}^{(l,i)})\right\| = \left\|\frac{\partial \mathcal{G}_{(k)}}{\partial u_{(k)}}(\mathrm{x},\mathrm{u})\right\| \tag{12}$$

which means the inner iterations in (11) converge if the maximal eigenvalue of $\partial\mathcal{G}_{(k)}/\partial u_{(k)}$ is smaller than one. Although no assumption was made that could guarantee this condition, since $\partial\mathcal{G}_{(k)}/\partial u_{(k)}$ is used during the primal simulation giving the solution u (usually by implicit time stepping), there is evidence that the eigen spectrum is non-problematic. However, should any convergence problems arise, for example due to limitations of the primal method or due to inherent unsteadiness (caused e.g. by flow separation), there are techniques to stabilize the inner fixed-point iterations, such as the recursive projection method by Shroff and Keller (1993) which applies a Newton update to a small number of eigenvectors with large eigenvalues. Many studies show that such an approach works well for tangent and adjoint computations [e.g., Campobasso and Giles (2004); Albring et al. (2017)]. Similarly, this applies to the full iteration (11), where we cannot assume that perturbations caused by the coupling terms are small enough that the spectrum of $\mathcal{N}$ does not become unstable (in the sense that it leads to large eigenvalues in the coupled system). Again, should (outer) convergence problems occur it is possible to use stronger coupling methods than BGS as done in Gomes and Palacios (2020). Clearly, in both cases (inner and outer) these stabilization techniques do not interfere significantly with the main goal of the proposed method, which is to be generic and applicable to any combination of zones, solvers, interface interpolation methods, etc.

In the more particular case of FSI discrete adjoint methods, we furthermore note that some authors take a 2-field (fluid and structure) approach to the problem [e.g., Kenway et al. (2014)], while others consider the mesh deformation as a third field [e.g., Sanchez et al. (2018)]. The zone concept is better suited to the 2-field approach, while the 3-field approach is more natural when establishing the coupled adjoint equations manually, as it allows isolating key terms of the block Jacobians in equations (8) and (9). This isolation is important for efficiency, and in the proposed algorithm it is achieved automatically by identifying key abstract phases of the primal solution algorithm, for example the transfer of data across interfaces. This is detailed in Sect. 3.

2.2 Design parameter sensitivities

In the previous section the parameters were assumed to be the mesh coordinates of all zones (but any other differentiable parameter of the solvers could have been given as example). For shape optimization it is not practical nor adequate to use all nodal coordinates as design variables. The number of shape design variables is reduced via the common strategy (in aerodynamic shape optimization) of using a surface parameterization followed by volume deformation [e.g., Economon et al. (2016)]. That is, a small set of design variables (e.g., free-form-deformation, FFD, control points) is used to define or modify the design surface nodes. The perturbation of these nodes is then used to deform the inner nodes of the meshes, a process akin to FSI mesh deformation, which in SU2 makes use of the same method, namely the solid elasticity analogy. Let this two-step process be represented by the explicit relation

$$(x_{(1)}, \dots, x_{(r)}) = M(a). \tag{13}$$

Due to the variety of methods for surface parameterization and volume deformation, it is natural to consider this operation separately from the primal solution. Consequently, its differentiation is also separate from the adjoint solution. Since the grid coordinates are an explicit function of the shape parameters a, derivatives with respect to these at point a are given by

$$\frac{\mathrm{d}}{\mathrm{d}a}\tilde{J}(x(\mathrm{a})) = \frac{\mathrm{d}}{\mathrm{d}x}\tilde{J}(x(\mathrm{a}))\ \frac{\mathrm{d}}{\mathrm{d}a}M(\mathrm{a}). \tag{14}$$

Although other alternatives can be proposed, this product fits the reverse mode of AD, and its cost is independent of the number of shape parameters. Finally, this modular approach can be extended to other types of optimization, for example, filtering operations for density-based topology optimization are a form of parameterization that fits (13).

3 Algorithmic differentiation of a multiphysics solver

The central part of the method described in the previous section is the evaluation of Jacobian transposed matrix-vector products of the form

$$\frac{\partial\mathcal{G}_{(j)}}{\partial u_{(k)}}^{\mathrm{T}}(\mathrm{x},\mathrm{u})\ \lambda_{(j)} \tag{15}$$

for a given pair of zones (j, k) and an adjoint vector $\lambda_{(j)}$. This section details how that is achieved with low storage and computational cost using the reverse mode of AD.

3.1 Storing and evaluating derivative information via AD

Using operator overloading AD, products of the form in (15) can be evaluated by first letting $\mathcal{G}$ compute a solution update starting from a previous iterate for all zones [as in (3)] while recording (or taping) the operations performed along the execution (or directly their derivatives) to an external data structure, that can then be used to evaluate the overall derivatives by applying the chain rule in reverse.

More precisely, each intermediate value that these operations generate is assigned an *index* of a global adjoint vector, incremented each operation, during the recording. The external data structure (called the *tape*) holds all information to compute partial derivatives of an intermediate value with respect to the inputs of the respective operation—which could be an elementary one or, for example, the solution of an entire linear system whose derivatives are handled symbolically in SU2.

In a reverse accumulation process (Griewank and Walther 2008), values defined by the term (15) are computed by initializing derivative values at indices of outputs of $\mathcal{G}$ with values of λ, accumulating all derivative contributions of intermediate values that were needed to compute $\mathcal{G}(u)$ from u and finally extracting derivatives at indices of u.

To draw a clear connection to the vocabulary of AD, let us write $y = \mathcal{G}(u)$ and define $\dot{y} := (\partial\mathcal{G}/\partial u)\dot{u}$ with $\dot{u}$ being an input derivative of u. We then have

$$\langle \dot{y}, \bar{y} \rangle = \left\langle \frac{\partial \mathcal{G}}{\partial u}\dot{u}, \bar{y} \right\rangle = \left\langle \dot{u}, \frac{\partial \mathcal{G}}{\partial u}^{\mathrm{T}} \bar{y} \right\rangle = \langle \dot{u}, \bar{u} \rangle, \tag{16}$$

with $\bar{u} := (\partial\mathcal{G}/\partial u)^{\mathrm{T}}\bar{y}$ and "$\langle\cdot,\cdot\rangle$" the standard inner product. Setting $\bar{y} = \lambda$ and $\dot{u} = 1$, we obtain (15). Note that, as pointed out already, rewriting $\langle \dot{y}, \lambda \rangle$ as $\bar{u}$ is crucial since computing the Jacobian first is unfeasible, whereas the reverse mode needs much less memory. There are many AD tools available that implement (16), but as in Albring et al. (2016), CoDiPack (Sagebaum et al. 2019) was chosen for its performance and convenience to use within $C++$ codes.

In order to proceed from the black-box evaluation in (8) to the modular evaluation needed to implement (15) (recall we deal with pairs of zones), it is important to access derivatives through indices that refer to the solution variable at its input or output state, as the transfer of data may assign them an incremented index (e.g., by overwriting data at boundaries that are an interface). In our implementation, we use an additional data structure storing both indices alongside each primal solution variable.

For cost effectiveness of the adjoint computation besides manual supply of derivatives for some (large) operations, we note that small parts of the recording are immediately evaluated, and a Jacobian matrix is *preaccumulated* [i.e. stored; Sagebaum et al. (2019)] instead to reduce the memory footprint.

Another efficiency aspect that has to be considered is the transfer of solution data across zones, and subsequent mesh deformation for some classes of problems. By using solution variable indices and tagging sections of the tape, cross derivatives like in (15) for $j \neq k$ can be computed once per outer iteration independently of the diagonal terms computed on every inner iteration. The tape layout and high level operations that allow this are described next.

3.2 Adjoint evaluation algorithm, subroutines and tape layout

For our implementation of (11) we build the adjoint solution update (Algorithm 1) upon the following subroutines:

- `ComputeAdjointsObjFunc`: First initializes the adjoint value of J with `1.0` and then evaluates all derivatives $\partial J/\partial u$ (x, u) for the variables of all zones.
- `ComputeAdjoints`: Taking the zone index j as argument, it initializes the adjoint values at the indices of $\mathrm{u}_{(j)}^{(i+1)}$ (that were saved during recording) with values of $\lambda_{(j)}$ (we leave out superscripts for simplicity, as inner and outer iterations are present). Then it evaluates the product $(\partial\mathcal{G}_{(j)}/\partial u)^{\mathrm{T}}(\mathrm{x},\mathrm{u})\ \lambda_{(j)}$ for all variables in one zone (inner iteration) or all zones (outer iteration with cross terms).
- `Iterate`: Takes the zone index k as argument and extracts the current derivative values (which can be $(\partial\mathcal{G}_{(j)}/\partial u_{(k)})^{\mathrm{T}}(\mathrm{x},\mathrm{u})\ \lambda_{(j)}$ or $\partial J/\partial u_{(k)}(\mathrm{x},\mathrm{u})$) from the tape indices for $\mathrm{u}_{(k)}^{(i)}$ into the adjoint solution vector in zone k.
- `GetExternal`: Besides the adjoint solution vector, every zone needs a separate vector, referred to as `External`, which holds the sum of cross terms $j \neq k$ and the objective function gradient. Mathematically this can be seen as the right hand side of the adjoint equations for each zone.
- `UpdateCrossTerm`: Takes zone indices j (source) and k (target) as inputs and updates the `External` vector in zone k with derivatives with respect to zone j (i.e., $(\partial\mathcal{G}_{(j)}/\partial u_{(k)})^{\mathrm{T}}(\mathrm{x},\mathrm{u})\ \lambda_{(j)}$). As the adjoints of zone j are updated, we update its cross contributions to zones $k \neq j$, so that the update of subsequent zones uses the most current value of their `External` vectors. Furthermore, we may want to apply a relaxation factor to these cross contributions. To do so, the (relaxed) difference between current and previous cross terms is added to the `External` of zone k, while simultaneously updating the previous cross term. This strategy requires a table of previous cross terms to be kept (a block sparse matrix with vector blocks).

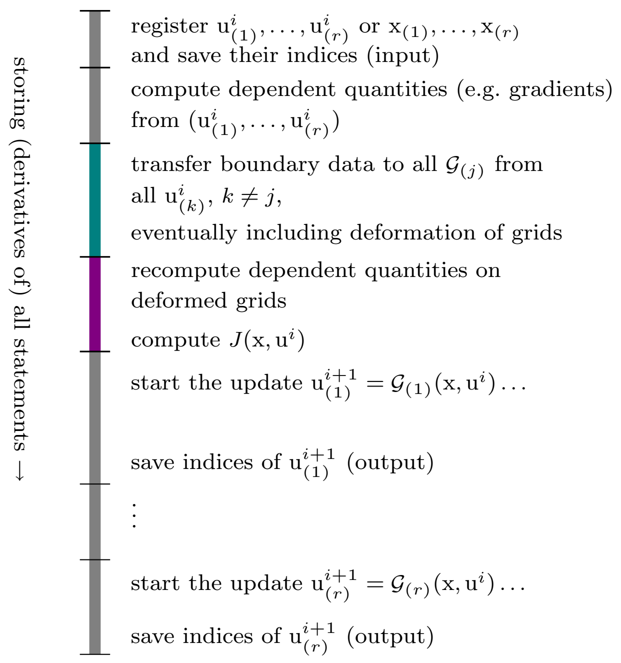
storing (derivatives of) all statements →
register $u^i_{(1)}, \dots, u^i_{(r)}$ or $x_{(1)}, \dots, x_{(r)}$ and save their indices (input)
compute dependent quantities (e.g. gradients) from $(u^i_{(1)}, \dots, u^i_{(r)})$
transfer boundary data to all $\mathcal{G}_{(j)}$ from all $u^i_{(k)}$, $k \neq j$, eventually including deformation of grids
recompute dependent quantities on deformed grids
compute $J(x, u^i)$
start the update $u^{i+1}_{(1)} = \mathcal{G}_{(1)}(x, u^i) \dots$
save indices of $u^{i+1}_{(1)}$ (output)
⋮
start the update $u^{i+1}_{(r)} = \mathcal{G}_{(r)}(x, u^i) \dots$
save indices of $u^{i+1}_{(r)}$ (output)

Fig. 1 Proposed tape layout for a multiphysics solver

```
ComputeAdjointsObjFunc();
for $k \leftarrow 1$ to $r$ do
    GetExternal($k$) = Iterate($k$);
end
for $i \leftarrow 1$ to OuterIter do
    for $j \leftarrow 1$ to $r$ do
        for $l \leftarrow 0$ to InnerIter do
            $\lambda[j] = \lambda[j]$ + GetExternal($j$);
            ComputeAdjoints($j$, $\lambda[j]$);
            $\lambda[j]$ = Iterate($j$);
        end
        for $k \leftarrow 1$ to $r$ do
            if $k \neq j$ then
                $\lambda[k]$ = Iterate($k$);
                UpdateCrossTerm($j$, $k$, $\lambda[k]$);
            end
        end
    end
end
```

Algorithm 1: Multiphysics discrete adjoint algorithm.

For a given zone pair (j, k), the `ComputeAdjoints` routine does not need to evaluate the entire tape (which includes all solvers), only the necessary sections are evaluated for computational efficiency (mainly $\mathcal{G}_{(j)}$ and possibly the data transfer). To allow this, key positions are saved during the recording of the primal computations, so that the tape evaluation can be restricted to the desired sections.

The layout of the tape is shown in Fig. 1. The saved positions for the corresponding sections, separated by hyphens, are: `Start`, `Registered`, `Dependencies`, `Transfer` and `Objective_Function`.

A further position, `It_Ready`, is saved to indicate the start of the recording of solvers of the first zone.

By these separations, the transfer of data section (colored blue) is automatically included when the section in purple, corresponding to the objective function, is evaluated (which is only the case when the adjoint of the objective function is relevant, i.e., when its Jacobian is needed). Moreover, using the `Transfer` position, the transfer of data section can be optionally included when evaluating w.r.t. zones, as needed in Algorithm 2.

The recording process makes use of software abstractions that exist for the primal solvers, for example running one iteration or computing the possible objective functions. One action that warrants further discussion is the computation of dependent quantities (second item in Fig. 1) as it is specific of the adjoint solvers. This *dependency* abstraction serves to capture the cyclic nature of iterative methods, for example, eddy viscosity is needed at the start of a RANS iteration, this quantity is a post-processing result of the turbulence solver, which in turn requires some pre-processing quantities of the flow solver, namely primitive variables and their gradients. Similarly, to trace the dependencies of a fluid iteration to the grid coordinates, one needs to evaluate the geometric properties of the grid (areas, volumes, etc.). In summary, the *dependency* abstractions are relatively simple (but crucial) orchestrations of pre- and post-processing operations.

A helper function `AD_ComputeAdjoints` taking as arguments the entry and exit points for evaluation (represented by horizontal hyphens in Fig. 1) provides an interface to the AD tool to evaluate the tape by sections.

The `ComputeAdjoints` routine is implemented in terms of this helper, e.g., to compute $(\partial\mathcal{G}_{(j)}/\partial u)^{\mathrm{T}}(x, u)\, \lambda_{(j)}$, four calls are needed to account for all sections, including data transfer (see Algorithm 2).

```
InitializeAdjoint($j$, $\lambda[j]$);
AD_ComputeAdjoints((j-1)*2+1+It_Ready,
  (j-1)*2+It_Ready);
AD_ComputeAdjoints(Transfer, Dependencies);
AD_ComputeAdjoints(Dependencies, Start);
```

Algorithm 2: Tape evaluation process.

As mentioned, for efficiency the transfer section of the tape is only evaluated in the last inner iteration (see Algorithm 1) as it may contain an expensive mesh deformation operation. Furthermore, during the recording of the deformation operation, we introduce the simplification that the stiffness matrix is constant (i.e., not a function of the solution variables or parameters) as this nearly halves the total memory usage of a typical FSI problem. This approximation is consistent with what is done to project the volumetric sensitivities onto the design surface modified by the shape parameterization approach. For similar memory usage concerns, and general differentiability of the methods involved, we also assume the interpolation operators used to transfer fluid forces and structural displacements between domains to be constant.

4 Applications

This section showcases the proposed method on three multiphysics examples, one for CHT, one for FSI, and a final combined FSI-CHT problem. In all problems $\mathcal{G}_{(1)}$ is a flow solver. In the first case, the second zone is a rigid material that conducts heat and transfers it into the fluid ($\mathcal{G}_{(2)}$ is a heat solver), whereas in the second case we have an elastic material that deforms under fluid loads ($\mathcal{G}_{(2)}$ is an elasticity solver).

In the proposed generic approach, both are two-zone problems for which the adjoint fixed-point iteration is

$$\begin{pmatrix} \lambda_{(1)}^{(i+1)} \\ \lambda_{(2)}^{(i+1)} \end{pmatrix} = \begin{pmatrix} \frac{\partial J}{\partial u_{(1)}} \\ \frac{\partial J}{\partial u_{(2)}} \end{pmatrix} + \begin{pmatrix} \frac{\partial \mathcal{G}_{(1)}}{\partial u_{(1)}} & \frac{\partial \mathcal{G}_{(1)}}{\partial u_{(2)}} \\ \frac{\partial \mathcal{G}_{(2)}}{\partial u_{(1)}} & \frac{\partial \mathcal{G}_{(2)}}{\partial u_{(2)}} \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} \lambda_{(1)}^{(i)} \\ \lambda_{(2)}^{(i)} \end{pmatrix}.$$

By the design of Algorithm 1, the matrix-vector product on the right hand side is not done in a single operation, rather it is split into several steps.

That is, to obtain the updated adjoint solution $\lambda_{(1)}^{(i+1)}$ for zone 1, $\lambda_{(1)}$ is updated by evaluating the diagonal term $(\partial \mathcal{G}_{(1)}/\partial u_{(1)})^{\mathrm{T}}\ \lambda_{(1)}^{(l)}$ for *InnerIter* times [cf. (11)] while adding to it the external contribution from the last outer iteration, $(\partial \mathcal{G}_{(2)}/\partial u_{(1)})^{\mathrm{T}}\ \lambda_{(2)}^{(i)}$, in each step.

Before updating $\lambda_{(2)}$ in similar fashion, the cross term $(\partial \mathcal{G}_{(1)}/\partial u_{(2)})^{\mathrm{T}}\ \lambda_{(1)}^{(i+1)}$ is evaluated with the new value of $\lambda_{(1)}$.

Note the physical interpretation of the cross terms for CHT problems:

- $\partial \mathcal{G}_{(1)}/\partial u_{(2)}$ constitutes the flow solver's dependence on the temperature of the solid, this is due to the temperature at the interface which determines the heat flux;
- $\partial \mathcal{G}_{(2)}/\partial u_{(1)}$ represents the heat solver's dependence on the heat fluxes at the interface, which are determined by the flow solver.

Whereas for an FSI problem:

- $\partial \mathcal{G}_{(1)}/\partial u_{(2)}$ is the flow solver's dependence on the fluid mesh displacements with respect to the initial (undeformed) grid coordinates, which in turn are defined by the structural displacements at the interface. That is, the displacements are an explicit function of the initial coordinates and of the fluid-structure interface displacements;
- $\partial \mathcal{G}_{(2)}/\partial u_{(1)}$ constitutes the elasticity solver's dependence on the fluid forces at the interface, which are due to pressure and shear stresses.

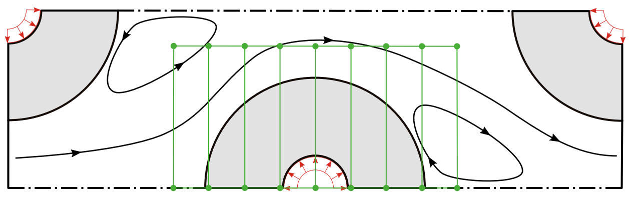
Fig. 2 Simulation setup schematic. Heatflux boundary conditions indicated with red arrows and symmetry planes with a dash-dotted line. The FFD box in green encloses the half pin

As already pointed out, all kinds of coupling terms (regardless of the kind of interface) are immediately computed whenever the transfer tape section is part of the adjoint evaluation process (as in Algorithm 2). In terms of the second example, it will directly evaluate the flow solver output w.r.t. communicated structural displacements as the mesh adaption is contained. Generally speaking, no additional operations are necessary for FSI interfaces which will be in particular useful for the combination test case.

4.1 Conjugate heat transfer test case

This test case is for steady state, turbulent, incompressible flow through a 2D array of heated cylinders. A typical pin array for e.g., power electronics cooling can consist of tens up to a couple of thousand individual pins. Instead of a simulation on the entire pin array a common simplification is to only use a characteristic unit cell, see Fig. 2. Through mirroring at symmetry boundaries and translating at periodic boundaries one recovers an approximation to the initial full array. The presented simulation setup includes several simplifications which have to be validated using a full size pin array. It is, however, within reason to presume that general design rules can be derived from such a reduced, fast to evaluate, unit cell approach.

The geometry of the simulation domain is fully characterized by three quantities: distance between pin midpoints 6.44 mm (three neighboring pins form an equilateral triangle), the inner pin radius 0.6 mm and outer pin radius 2 mm. Inlet and outlet in the fluid domain are prescribed as periodic boundaries (Patankar et al. 1977). Symmetry boundaries on top and bottom are indicated in Fig. 2 and handling of the interfaces between fluid and solid is described in literature (Burghardt et al. 2019). On the inner pin arc in the solids a heatflux of 5e5 W/m^2 is prescribed and all other non-interface walls are adiabatic.

The pressure drop over the domain is set to 208 Pa and the inlet bulk temperature to 338 K. Other constant fluid

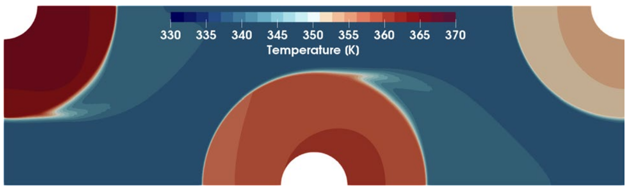

Fig. 3 Temperature contour lines

properties are density $\rho = 1045\ \mathrm{kg/m^3}$, specific heat $c_p = 3540$ J/(kg K), laminar viscosity $\mu_{dyn} = 1.385e-3$ Pa s, laminar and turbulent Prandtl numbers $\mathrm{Pr}_{lam} = 11.7$ and $\mathrm{Pr}_{turb} = 0.9$. The fluids material properties represent Glysantin in a 50/50 mixture with water, a commonly used liquid for automotive cooling circuits. The constant material properties in the solid represent aluminum: density $\rho = 2719\,\mathrm{kg/m^3}$, specific heat $c_p = 871$ J/(kg K) and thermal conductivity $\lambda = 200$ W/(m K). Convective fluxes are discretized using a Flux-Difference-Splitting (FDS) with second order accuracy achieved by a MUSCL approach (Economon 2020). No limiters are used for variable reconstruction and gradients are computed via the Green-Gauss theorem. The Menter SST turbulence model without wall function is used.

The computational mesh is fully structured, e.g., only quadrilaterals are used, and consists of 8477 elements in the fluid and 6747 elements in the solid. A $y^+ < 1$ is obtained everywhere but the mesh is too coarse in order to ensure a mesh independent solution. The purpose here is solely to verify the gradient accuracy of the discrete adjoint method. Nonetheless, temperature contour lines are given in Fig. 3 that show a thin temperature boundary layer on the middle pin and a noticeable heat convection into the fluid body at the separation point at approximately 1 o'clock. Note that the quarter pins are not periodically connected in this simulation which leads to a considerable temperature difference compared to the examined half pin.

As geometry parametrization a free-form-deformation (FFD) box encloses the half pin in the center of the simulation domain, see Fig. 2, and consists of 18 control points. For the gradient validation only the vertical movement of the middle, upper control point is considered. Since a constant heatflux is prescribed, the temperature of the pin is determined by the cooling performance of the pin shape, i.e., the geometry of the fluid-solid interface. Therefore the average temperature on the inner arc of the center pin is set as an objective function and the interface as the region to be designed. Other boundaries are fixed to preserve symmetry. Note that for simplicity the gradient only considers changes to the center pin and not equivalent changes to the quarter pins.

The sensitivity computed via discrete adjoint for the specified FFD point (− 741.3575) is validated using finite

Table 1 Validation of the average temperature gradient obtained from adjoints (− 741.3575) against finite differences (FD) with deformation paramater $a = a_0 + 10^p$ for the middle upper FFD point in Fig. 2

Magnitude p	Finite difference	Relative error (%)
− 4	− 705.6977	4.81007
− 5	− 738.1632	0.43087
− 6	− 740.5946	0.10291
− 7	− 740.8425	0.06947

differences as shown in Table 1. The optimal step size has to be determined for each design variable individually such that choosing a single step size for all design variables would lead to higher deviation for selected FFD points. Note that further lowering the finite difference step size in Table 1 results in a higher difference due to increased numerical errors.

The CHT adjoint testcase has a maximum memory consumption of 2 GB which is approximately 2 times the memory used by the primal evaluation. The same ratio is observed in the compute time which is about 1000 CPU seconds for the adjoint evaluation and was run on 14 cores (Intel Xeon Gold 6132 CPU). A compute time comparison highly depends on convergence criteria and is only given to show the comparable compute cost between primal and adjoint evaluation.

4.2 Fluid–structure interaction test case

To verify the shape sensitivities for FSI problems we consider the simulation of a flexible wing at low Mach number (0.6) and 4 degrees angle-of-attack (AoA). The wing geometry (Fig. 4) is generated by *lofting* two symmetric 4-digit NACA profiles, a 0.25 m chord 9% thickness profile at the root, and a 0.175 m chord 7.2% thickness profile at the tip. The wing span is 1 m, with the 0.25c line swept back 5 degrees, and a linear twist distribution of -3 degrees to prevent static divergence. The primal simulation methodology has been described in the literature (Gomes and Palacios 2020), thus here we describe only the choice of numerical schemes. We approximate the wing structure as a neo-Hookean hyper-elastic solid with elasticity modulus of 7.5 GPa and Poisson's ratio of 0.35, with these properties the vertical displacement of the wing tip is approximately 20% of chord (see Fig. 5). On the fluid side air at standard temperature and pressure is considered. Both fluid and structural grids are composed mostly of hexahedra, the former having 832 000 nodes and the latter 174 000 nodes, respectively. We note that the fluid grid is not sufficiently fine for detailed flow field analysis, as $y^+ \approx 5$ and we use Menter's SST turbulence model without wall functions (which at the time of writing were not fully implemented in SU2). However, it

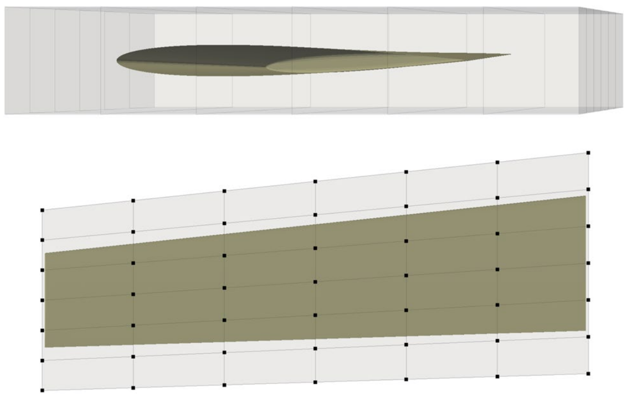

Fig. 4 Wing geometry and free-form-deformation box, viewed from the wing tip and the top

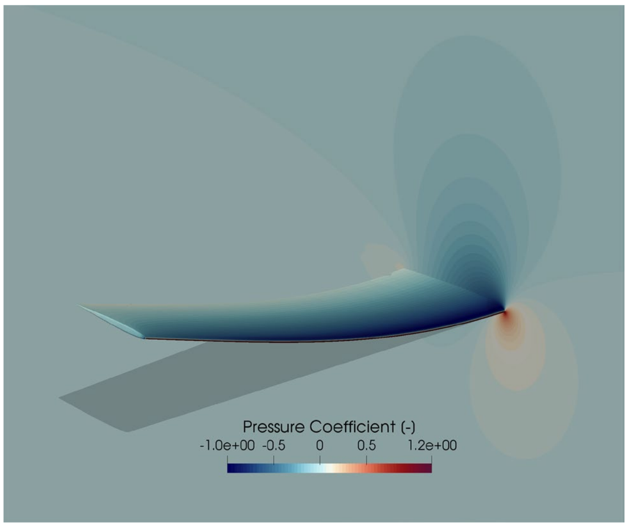

Fig. 5 Deformed configuration of the wing and pressure coefficient contours

is adequate for verification of discrete adjoint sensitivities (whose accuracy is not mesh dependant). Convective fluxes are computed with a second order Roe scheme, where the gradients are computed via the Green-Gauss theorem, and the MUSCL reconstruction of flow variables is limited with Venkatakrishnan and Wang's limiter.

The free-form-deformation (FFD) box shown in Fig. 4 has 98 control points, rather than verifying derivatives for each point (and each Cartesian coordinate), we define stretch (acting on chord) and translation variables for each spanwise section of points. We will refer to these 14 variables as S_{0-6} and T_{0-6} , respectively (starting the numbering at the root) and normalize them taking as reference value 0.05 m (20% of root chord). The gradients of drag coefficient (C_d) and elastic energy (or compliance) are verified, we chose these functionals as the first is fluid oriented, whereas the second is mostly structural. The values of the functionals are also

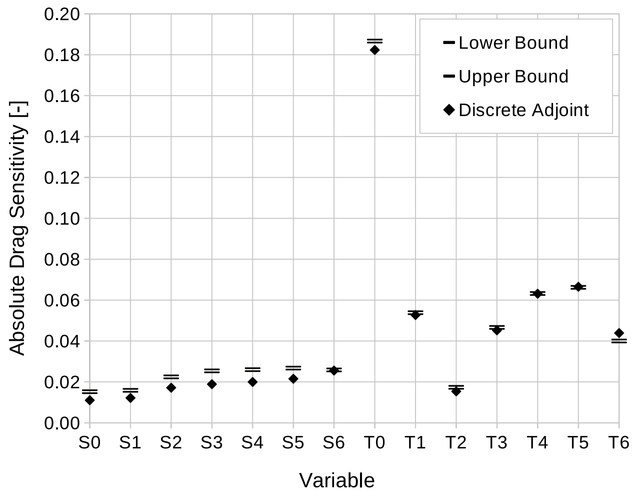

Fig. 6 Absolute value of drag sensitivities, from the adjoint solver, and finite difference error bounds

normalized (by 0.01 and 75 J, respectively) which leaves both gradients dimensionless and of comparable magnitudes.

A suitable finite difference step size was determined by conducting a convergence study, starting with a step size of 0.05 and halving it each time we obtained second order central approximations to the gradients. At step sizes below 0.0125 the approximations start to diverge due to the typical limitations of finite differences, as the variation of the functionals approaches the accuracy to within which the primal problem can be converged ($\approx 10^{-6}$ change over FSI iterations). Therefore, gradients obtained with step size of 0.025 were considered. The error associated with the approximation, used to compute lower and upper confidence bounds, was estimated by taking the average plus three standard deviations of the differences between second and fourth order central approximations, across all variables (S_{0-6}, T_{0-6}). With fourth order values being computed using also the function values obtained at ± 0.05. The results for drag are plotted in Fig. 6, together with the sensitivity values computed via the discrete adjoint approach. Fig. 7 shows the results for the elastic energy functional.

The agreement between finite difference approximations and adjoint sensitivities of drag is better for the translation variables (T_{0-6}), to some extent this was expected due to the larger magnitude of those derivatives, and the way those variables are constructed from the FFD control points (a translation variable moves all the points in a section by an equal amount, whereas a stretch variable moves points away from the centerline of the box proportionally to their distance to it). The overall agreement is also better for the elastic energy functional, this is also expected as the elastic energy depends strongly on pressure and structural deformation (i.e., almost directly on the solution variables), whereas drag at subsonic

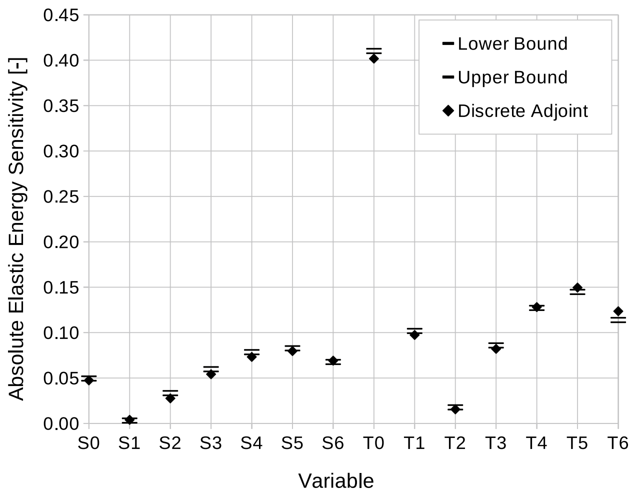

Fig. 7 Absolute value of elastic energy sensitivities, from the adjoint solver, and finite difference error bounds

speed depends mostly on viscous effects (and therefore on gradients of the solution variables).

Since due to the typical limitations of finite differences, it was not possible to obtain derivative approximations with a small enough step size (the one used is 2.5% of the plausible range for use in optimization) we cannot attribute the discrepancies only to the memory-saving approximations introduced in Subsect. 3.2.

A typical challenge of implementing an algorithmic differentiation-based discrete adjoint solver in a non-intrusive manner, i.e., not requiring extensive modifications to the architecture of the code, and/or of the implementation of the algorithms, is maintaining a low memory footprint. Running the adjoint FSI problem used for verification requires 113 GB of memory, or approximately 4 times the memory required to run the primal simulation. We note, however, that while multi-grid acceleration was used for the primal solution, it was not for the adjoint solution, doing so roughly doubles the memory footprint of the fluid problem, which would increase the adjoint-to-primal memory ratio to ≈ 6. Notwithstanding not using multi-grid, the adjoint solution takes approximately the same time as the primal, about 15 minutes running on 48 cores (4 Intel Xeon E5-2650v4 CPU). This memory footprint ratio is reasonable considering that operator overloading AD is used throughout the code [compared to Albring et al. (2016); Towara et al. (2019); Kenway et al. (2019)]. In an absolute sense it may appear large for the total problem size of one million fluid and structural nodes. However, for practical purposes, it is more relevant to judge the memory footprint against the available memory in the number of compute nodes that needs to be enlisted to obtain optimization results in reasonable time frames. The current framework also performs well under this metric.

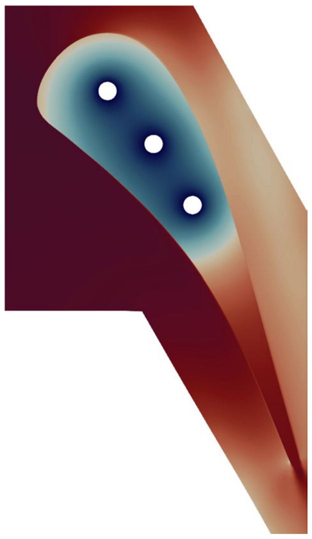

Fig. 8 Cooled turbine blade in high-pressure flow

4.3 Turbine stage stator blade (combination case of CHT and FSI)

This final test case highlights the modularity of the approach presented in this paper. As it not only allows for handling multiphysics setups where the number and type of physical domains is unknown, but also for arbitrary interfaces between them. In this example a single zone acts as a heat conducting body with a CHT interface to a surrounding fluid zone, simultaneously, the same physical interface also receives solid displacements by additionally being marked as an FSI interface. This results in a deformation of the heat and fluid zones according to a third elastically deforming zone, which occupies the same physical space as the heat conducting zone. In this example the same numerical grid is used twice, but it would also be possible to use different grids to individually resolve temperature and stress gradients.

Similarly to the previous examples, the sensitivity analysis framework is applied to this turbine stator blade with three internal cooling channels (Fig. 8) in high pressure compressible flow that causes deformations altering the pressure difference acting on the blade. Keeping track of the heat transfer, e.g., to control the temperature of the blade, requires the deformation to be considered (Fig. 9).

Effectively three different kinds of physics are coupled by exchanging structural displacements, fluid forces, and heat quantities at the physical interface. Note that no thermoelastic solver was used for this study, i.e., we assume no dependency of the structural solver on the temperature distribution.

The cooling channels are set to a fixed temperature of 400 K, the compressible air flow at Mach 0.2 enters at

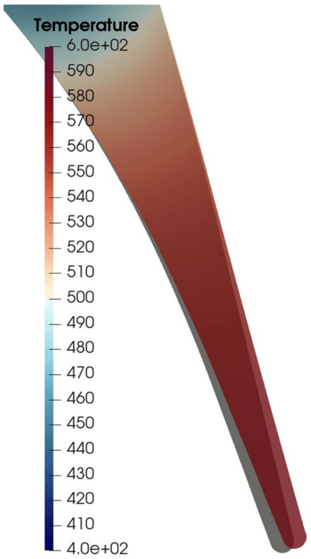

Fig. 9 Deformation of the blade through flow tractions

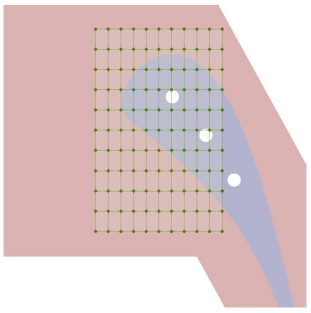

Fig. 10 FFD box, each vertex being a shape parameter

1.3 MPa and 600 K and drops to 0.9 MPa at the outlet. The material parameters are set to the ones of steel, except for a significantly reduced elasticity modulus of 4GPa in order to increase the deformation, and thus increase the importance of coupling effects (in a 3D setting significant deformation would occur also with larger elasticity moduli due to bending and torsional loads over the span).

A free-form-deformation box (Fig. 10) is created around the blade in order to validate the adjoint solutions containing all cross contributions created by the dependencies between the primal solvers. A single FFD parameter (a) of the box (control point at position (6,9) in the green lattice) is varied (resulting in the deformation shown in Fig. 11) and the change in the heat flux (given by the surface integral around the blade perimeter based on the flow solution) is monitored.

The heat flux derivative $\frac{\mathrm{d}}{\mathrm{d}a}\tilde{J}(x(a_0))$ [cf. (14)] is calculated by summing up contributions obtained from the adjoint solutions (7) of the flow (-276185.18), heat conduction

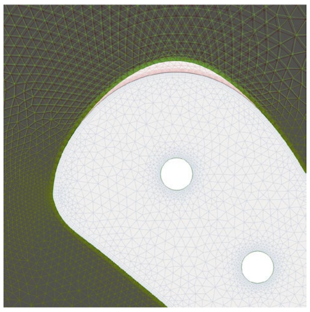

Fig. 11 Shape modification through a single parameter

Table 2 Validation of the heat flux gradient obtained from adjoints against finite differences (FD) with deformation paramater $a = a_0 + 10^p$

Magnitude p	Finite difference	Relative error (%)
− 4	12168.1	0.3074
− 5	12142.2	0.0989
− 6	12132.1	0.0159
− 7	12130.0	0.0013

(287154.54), and the elastic deformation (1160.81), giving a total of $\frac{\mathrm{d}}{\mathrm{d}a}\tilde{J}(x(a_0)) = 12130.17$. Regarding these values, one notes that the elastic deformation has a significant effect on the shape sensitivity of the heat flux. This result was verified using finite differences as shown in Table 2.

Error estimates based on the setup used for this test case become less reliable for perturbations below 1.0E − 7 due to numerical errors. As the relative importance of the coupling effects, illustrated by the individual contributions to the total derivative, is much larger than the relative error between finite differences and the adjoint method, higher order finite difference approximation were considered unnecessary.

5 Conclusions

This paper has presented a methodology on how to set up and implement an algorithm to compute discrete adjoints for multiphysics problems using algorithmic differentiation, thereby providing accurate gradients for optimization purposes.

By design, no a priori knowledge about the physical setup of the primal problem is required, which allows the current implementation in SU2 to be applied to conjugate heat transfer, fluid–structure interaction, or combinations of both, without problem-specific code having to be written for each coupled adjoint problem.

To the authors' knowledge, no such unifying approach has been pursued so far within the optimization community. Besides all apparent advantages of using a common code base (such as ease of maintenance and compatibility), it also opens the way to include further kinds of physics with minimal effort, to ultimately tackle arbitrarily complex multiphysics optimization problems.

Acknowledgements The authors would like to gratefully acknowledge the support of the Bayerische Forschungsstiftung (Grant No. BFS AZ-1232-16), as well as the computational resources provided by the RHRK high-performance computing center via the 'Elwetritsch' high-performance cluster at the TU Kaiserslautern.

Funding Open Access funding enabled and organized by Projekt DEAL.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Replication of results The source code developed for this work required to run the adjoint and sensitivity computations as shown in Sect. 4 is available at https://github.com/su2code/SU2.

References

Albring TA, Sagebaum M, Gauger NR (2016) Efficient aerodynamic design using the discrete adjoint method in SU2. In: 17th AIAA/ISSMO multidisciplinary analysis and optimization conference

Albring TA, Dick T, Gauger NR (2017) Assessment of the recursive projection method for the stabilization of discrete adjoint solvers. In: 18th AIAA/ISSMO multidisciplinary analysis and optimization conference

Biedron RT, Carlson JR, Derlaga JM, Gnoffo PA, Hammond DP, Jones WT, Kleb B, Lee-Rausch EM, Nielsen EJ, Park MA, Rumsey CL, Thomas JL, Thompson KB, Wood WA (2019) Fun3d manual: 13.6. In: NASA TM 2019-220416

Burghardt O, Gauger NR, Economon TD (2019) Coupled adjoints for conjugate heat transfer in variable density incompressible flows. In: AIAA Aviation 2019 Forum

Campobasso MS, Giles MB (2004) Stabilization of a linear flow solver for turbomachinery aeroelasticity using recursive projection method. AIAA J 42(9):1765–1774

Degroote J, Bathe K-J, Vierendeels J (2009) Performance of a new partitioned procedure versus a monolithic procedure in fluid–structure interaction. Comput Struct 87(11):793–801

Dunning PD, Stanford BK, Kim HA (2015) Coupled aerostructural topology optimization using a level set method for 3D aircraft wings. Struct Multidisc Optim 51(5):1113–1132

Economon TD (2020) Simulation and adjoint-based design for variable density incompressible flows with heat transfer. AIAA J 58(2):757–769

Economon TD, Palacios F, Copeland SR, Lukaczyk TW, Alonso JJ (2016) SU2: an open-source suite for multiphysics simulation and design. AIAA J 54(3):828–846

Feppon F, Allaire G, Bordeu F, Cortial J, Dapogny C (2019) Shape optimization of a coupled thermal fluid-structure problem in a level set mesh evolution framework. SeMA J 76(3):413–458

Giles MB, Pierce NA (1997) Adjoint equations in cfd-duality, boundary conditions and solution behaviour. In: 13th computational fluid dynamics conference

Giles MB, Pierce NA (2000) An introduction to the adjoint approach to design. Flow, Turbul Combust 65(3):393–415

Gomes P, Palacios R (2020) Aerodynamic-driven topology optimization of compliant airfoils. Struct Multidisc Optim 62:2117–2130

Griewank A, Walther A (2008) Evaluating derivatives: principles and techniques of algorithmic differentiation, 2nd edn. Society for industrial and applied mathematics, Philadelphia

He P, Mader CA, Martins JRRA, Maki KJ (2018) An aerodynamic design optimization framework using a discrete adjoint approach with OpenFOAM. Comput Fluids 168:285–303

Hinze M, Pinnau R, Ulbrich M, Ulbrich S (2008) Optimization with PDE constraints. Springer, Berlin

Jameson A (1995) Optimum aerodynamic design using cfd and control theory. In: 12th computational fluid dynamics conference

Jameson A (1988) Aerodynamic design via control theory. J Sci Comput 3(3):233–260

Kamali S, Mavriplis DJ, Anderson EM (2020) Sensitivity analysis for aero-thermo-elastic problems using the discrete adjoint approach. In: AIAA Aviation 2020 Forum

Kenway GKW, Kennedy GJ, Martins JRRA (2014) Scalable parallel approach for high-fidelity steady-state aeroelastic analysis and adjoint derivative computations. AIAA J 52(5):935–951

Kenway GKW, Mader CA, He P, Martins JRRA (2019) Effective adjoint approaches for computational fluid dynamics. Prog Aerosp Sci 110:100542

Lions JL (1971) Optimal control of systems governed by partial differential equations. Springer, Berlin

Lundgaard C, Alexandersen J, Zhou M, Andreasen CS, Sigmund O (2018) Revisiting density-based topology optimization for fluid–structure-interaction problems. Struct Multidisc Optim 58(3):969–995

Makhija DS, Beran PS (2019) Concurrent shape and topology optimization for steady conjugate heat transfer. Struct Multidisc Optim 59(3):919–940

Markus T, Johannes L, Uwe N (2019) Discrete adjoint approaches for CHT applications in OpenFOAM. Advances in evolutionary and deterministic methods for design, optimization and control in engineering and sciences. Springer, Cham, pp 163–178

Martins JRRA, Ning A (2021) Engineering design optimization. Cambridge University Press, Cambridge

Maute K, Nikbay M, Farhat C (2000) Analytically based sensitivity analysis and optimization of nonlinear aeroelastic systems. In: 8th symposium on multidisciplinary analysis and optimization, California, USA. pp 1–10

Patankar SV, Liu CH, Sparrow EM (1977) Fully developed flow and heat transfer in ducts having streamwise-periodic variations of cross-sectional area. J Heat Transfer 99(2):180–186

Picelli R, Ranjbarzadeh S, Sivapuram R, Gioria RS, Silva ECN (2020) Topology optimization of binary structures under design-dependent fluid–structure interaction loads. Struct Multidisc Optim 62(4):2101–2116

Sagebaum M, Albring T, Gauger NR (2019) High-performance derivative computations using CoDiPack. ACM Trans Math Softw 45(4):1–26

Sanchez R, Albring T, Palacios R, Gauger NR, Economon T, Alonso J (2018) Coupled adjoint-based sensitivities in large-displacement fluid–structure interaction using algorithmic differentiation. Int J Num Methods Eng 113(7):1081–1107

Shroff Gautam M, Keller Herbert B (1993) Stabilization of unstable procedures: the recursive projection method. SIAM J Num Anal 30(4):1099–1120

Smith LJ, Halim LJ, Kennedy G, Smith MJ (2021) A high-fidelity coupling framework for aerothermoelastic analysis and adjoint-based gradient evaluation. In: AIAA Scitech 2021 Forum

Weller HG, Tabor G, Jasak H, Fureby C (1998) A tensorial approach to computational continuum mechanics using object-oriented techniques. Comput Phys 12(6):620–631