53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference
20th AI
23 - 26 April 2012, Honolulu, Hawaii

AIAA 2012-1589

Review and Unification of Methods for Computing Derivatives of Multidisciplinary Systems

Joaquim R. R. A. Martins * and John T. Hwang †
University of Michigan, Ann Arbor, Michigan, 48109, United States

This paper presents a comprehensive review of all the options available for computing derivatives of multidisciplinary systems in a unified mathematical framework. The basic building blocks for computing derivatives are first introduced: finite differencing, the complex-step method and symbolic differentiation. A generalized chain rule is derived from which it is possible to derive both algorithmic differentiation and analytic methods. This generalized chain rule is shown to have two forms — a forward form and a reverse form — which correspond to the forward and reverse modes in algorithmic differentiation and the direct and adjoint approaches in analytic methods. Finally, the theory is extended to methods for computing derivatives of multidisciplinary systems, and several new insights are provided.

Nomenclature

n		Number of variables in a given context
n_f		Number of output variables
n_x		Number of input variables
n_y		Number of state variables
$\boldsymbol{e}_i$	$[0, \ldots, 1, \ldots, 0]^T$	i^{th} Cartesian basis vector
$\boldsymbol{f}$	$[f_1, \ldots, f_{n_f}]^T$	Vector of output variables
$\boldsymbol{r}$	$[r_1, \ldots, r_{n_y}]^T$	Vector of residuals
$\boldsymbol{r}_i$	$[r_{1,i}, \ldots, r_{n_y,i}]^T$	Vector of residuals belonging to the i^{th} discipline
$\boldsymbol{v}$	$[v_1, \ldots, v_n]^T$	Vector of variables in an given context
$\boldsymbol{x}$	$[x_1, \ldots, x_{n_x}]^T$	Vector of input variables
$\boldsymbol{y}$	$[y_1, \ldots, y_{n_y}]^T$	Vector of state variables
$\boldsymbol{y}_i$	$[y_{1,i}, \ldots, y_{n_y,i}]^T$	Vector of state variables belonging to the i^{th} discipline
$\boldsymbol{F}$	$[F_1, \ldots, F_{n_f}]^T$	Vector of output functions
$\boldsymbol{R}$	$[R_1, \ldots, R_{n_y}]^T$	Vector of residual functions
$\boldsymbol{R}_i$	$[R_{1,i}, \ldots, R_{n_y,i}]^T$	Vector of residual functions belonging to the i^{th} discipline
$\boldsymbol{Y}_i$	$[Y_{1,i}, \ldots, Y_{n_y,i}]^T$	Vector of intermediate functions belonging to the i^{th} discipline
$\boldsymbol{V}$	$[V_1, \ldots, V_n]^T$	Vector of functions in an given context
$\Delta\boldsymbol{v}$	$[\Delta v_1, \ldots, \Delta v_n]^T$	Vector of perturbations of $\boldsymbol{v}$ about the linearization point
$\dfrac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}}$	$\left[\dfrac{\mathrm{d}f_i}{\mathrm{d}x_j}\right]_{n_f \times n_x}$	Jacobian of total derivatives of $\boldsymbol{f}$ with respect to $\boldsymbol{x}$
$\dfrac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}}$	$\left[\dfrac{\partial F_i}{\partial x_j}\right]_{n_f \times n_x}$	Jacobian of partial derivatives of $\boldsymbol{F}$ with respect to $\boldsymbol{x}$
$\boldsymbol{D}_{\boldsymbol{v}}$	$\left[\dfrac{\mathrm{d}v_i}{\mathrm{d}v_j}\right]_{n \times n}$	Lower triangular Jacobian of total derivatives
$\boldsymbol{D}_{\boldsymbol{V}}$	$\left[\dfrac{\partial V_i}{\partial v_j}\right]_{n \times n}$	Lower triangular Jacobian of partial derivatives

*Associate Professor, Department of Aerospace Engineering, AIAA Senior Member
†Ph.D. Candidate, Department of Aerospace Engineering, AIAA Student Member

I. Introduction

The computation of derivatives is part of the broader field of sensitivity analysis, which is the study of how the outputs of a model change in response to changes in the inputs of the same model, and plays a key role in gradient-based optimization, uncertainty quantification, error analysis, model development, and computational model-assisted decision making. There are various types of sensitivities that can be defined. One common classification distinguishes between local and global sensitivity analysis [1]. Global sensitivity analysis aims to quantify the response with respect to inputs over a wide range of values, and it is better suited for models that have large uncertainties. Local sensitivity analysis aims to quantify the response for a fixed set of inputs, and is typically used in physics based models where the uncertainties tend to be lower. In the present review, we focus on the computation of local sensitivities in the form of first-order total derivatives, where the model is a numerical algorithm. The computational models are assumed to be deterministic. Although stochastic models require approaches that are beyond the scope of this paper, some of these approaches can benefit from the deterministic techniques presented herein.

Derivatives play a central role in many numerical algorithms. In many cases, such as in Newton-based methods, the computational effort of an algorithm depends heavily on the run time and memory requirements of the computation of the derivatives. Examples of such algorithms include Newton–Krylov methods applied to the solution of the Euler equations [2], coupled aerostructural equations [3, 4, 5], and quasi-Newton methods used to solve optimization problems [6, 7]. Other applications of derivatives include gradient-enhanced surrogate models [8], structural topology optimization [9, 10, 11, 12], aerostructural optimization [13, 14] and aircraft stability [15, 16].

The accuracy of the derivative computation affects the convergence behavior of the solver used in the algorithm. For instance, accurate derivatives are important in gradient-based optimization to ensure robust and efficient convergence, especially for problems with large numbers of constraints. The precision of the gradients limits that of the optimum solution, and inaccurate gradients can cause the optimizer to halt or to take a less direct route to the optimum that involves more iterations.

In this review, for generality, we consider the numerical models to be algorithms that solve a set of governing equations to find the state of a system. The computational effort involved in these numerical models, or simulations, is assumed to be significant. Examples of such simulations include computational fluid dynamics (CFD) and structural finite-element solvers. We also extend our review to consider multiple coupled simulations, which appear in multidisciplinary design optimization (MDO) problems.

The simplest method for computing derivatives is the use of an appropriate finite-difference formula, such as a forward finite-difference, where each input of interest is perturbed and the output reevaluated to determine its new value. The derivative is then estimated by taking the difference in the output relative to the unperturbed one and dividing by the value of the perturbation. Although finite differences are not known for being particularly accurate or computationally efficient, they are extremely easy to implement and therefore widely used.

In addition to inaccuracies inherent in finite-differences, computing sensitivities with respect to a large number of inputs using these methods is prohibitively expensive when the computational cost of the simulations is significant. Most applications require more accuracy and efficiency than is afforded by this approach, motivating the pursuit of the more advanced methods that we describe in this paper.

The overarching goal of this paper is to review the available methods for the sensitivity analysis of coupled systems and to advance the understanding of these methods in a unified mathematical framework. Some of this material has been the subject of excellent reviews and textbooks, but they have either been limited to a single discipline [17, 18, 19, 20] or limited in scope [21, 22, 23]. Spurred by the recent advances in this area, we decided to write this review and connect some methods that are usually not explained together, leading to new insights and a broader view of the subject. In addition to a deeper understanding of the topic, we aim to help researchers and practitioners decide which method is suitable for their particular needs.

We start this review by defining the nomenclature and the context of the theory. Then we progress through the theory of sensitivity analysis for single systems and connect the various methods under a unified mathematical framework. Finally, we extend this theory to the sensitivity analysis of coupled systems, and present some recent advances. The historical literature is cited as the theory is presented.

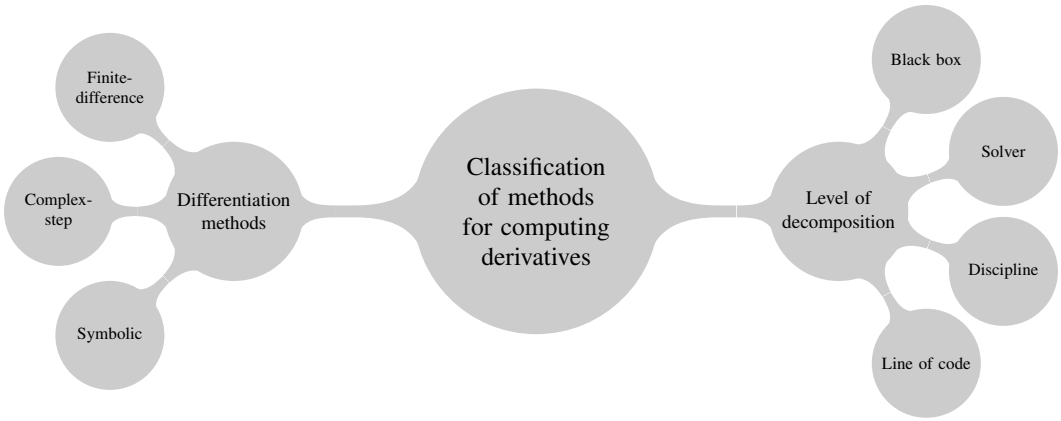


Figure 1: Classification of methods for derivative computations: the differentation methods are the building blocks for other methods, each of which considers a different level of decomposition.

II. Differentiation of a Function

Throughout this paper we assume that we want ultimately to compute the derivatives of a vector-valued function $\boldsymbol{f}$ with respect to a vector of independent variables $\boldsymbol{x}$, i.e., we want the Jacobian,

$$\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}} = \begin{bmatrix} \frac{\mathrm{d}f_1}{\mathrm{d}x_1} & \cdots & \frac{\mathrm{d}f_1}{\mathrm{d}x_{n_x}} \\ \vdots & \ddots & \vdots \\ \frac{\mathrm{d}f_{n_f}}{\mathrm{d}x_1} & \cdots & \frac{\mathrm{d}f_{n_f}}{\mathrm{d}x_{n_x}} \end{bmatrix} \tag{1}$$

which is an $n_f \times n_x$ matrix.

A. Finite Differences

Finite-difference formulas are derived from combining Taylor series expansions. Using the right combinations of these expansions, it is possible to obtain finite-difference formulas that estimate an arbitrary order derivative with any required order truncation error. The simplest finite-difference formula can be directly derived from one Taylor series expansion, yielding

$$\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}x_j} = \frac{\boldsymbol{f}(\boldsymbol{x} + \boldsymbol{e}_j h) - \boldsymbol{f}(\boldsymbol{x})}{h} + \mathcal{O}(h) \tag{2}$$

which is directly related to the definition of derivative. Note that in general there are multiple functions of interest, and thus $\boldsymbol{f}$ can be a vector that includes all the outputs of a given component. The application of this formula requires the evaluation of a component at the reference point $\boldsymbol{x}$, and one perturbed point $\boldsymbol{x} + \boldsymbol{e}_j h$, and yields one column of the Jacobian (1). Each additional column requires an additional evaluation of the component. Hence, the cost of computing the complete Jacobian is proportional to the number of input variables of interest, n_x.

Finite-difference methods are widely used to compute derivatives due to their simplicity and the fact that they can be implemented even when a given component is a black box. Most gradient-based optimization algorithms perform finite-differences by default when the user does not provide the required gradients.

When it comes to accuracy, we can see from the forward-difference formula (2) that the truncation error is proportional to the magnitude of the perturbation, h. Thus it is desirable to decrease h as much as possible. The problem with decreasing h is that the perturbed value of the functions of interest will approach the reference values. When using finite-precision arithmetic, this leads to *subtractive cancellation*: a loss of significant digits in the subtraction operation. In the extreme case, when h is small enough, all digits of the perturbed functions will match the reference values, yielding zero for the derivatives. Given the opposite trends exhibited by the subtractive cancellation error and truncation error, for each $\boldsymbol{x}$ there is a best h that minimizes the overall error.

Due to their flexibility, finite-difference formulas can always be used to compute derivatives, at any level of nesting. They can be used to compute derivatives of a single function, composite functions, iterative functions or any system with multiply nested components.

B. Complex Step

The complex-step derivative approximation, strangely enough, computes derivatives of real functions using complex variables. This method originated with the work of Lyness and Moler [24] and Lyness [25]. They developed several methods that made use of complex variables, including a reliable method for calculating the n^{th} derivative of an analytic function. However, only later was this theory rediscovered by Squire and Trapp [26], who derived a simple formula for estimating the first derivative.

The complex-step derivative approximation, like finite-difference formulas, can also be derived using a Taylor series expansion. Rather than using a real step h, we now use a pure imaginary step, ih. If $\boldsymbol{f}$ is a real function in real variables and it is also analytic, we can expand it in a Taylor series about a real point $\boldsymbol{x}$ as follows,

$$\boldsymbol{f}(\boldsymbol{x} + ih\boldsymbol{e}_j) = \boldsymbol{f}(\boldsymbol{x}) + ih\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}x_j} - \frac{h^2}{2}\frac{\mathrm{d}^2\boldsymbol{f}}{\mathrm{d}x_j^2} - \frac{ih^3}{6}\frac{\mathrm{d}^3\boldsymbol{f}}{\mathrm{d}x_j^3} + \ldots \tag{3}$$

Taking the imaginary parts of both sides of this equation and dividing it by h yields

$$\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}x_j} = \frac{\operatorname{Im}\left[\boldsymbol{f}(\boldsymbol{x} + ih\boldsymbol{e}_j)\right]}{h} + \mathcal{O}(h^2) \tag{4}$$

Hence the approximations is a $\mathcal{O}(h^2)$ estimate of the derivative. Like a finite-difference formula, each additional evaluation results in a column of the Jacobian (1), and the cost of computing the required derivatives is proportional to the number of design variables, n_x.

Because there is no subtraction operation in the complex-step derivative approximation (4), the only source of numerical error is the truncation error, which is $\mathcal{O}(h^2)$. By decreasing h to a small enough value, the truncation error can be made to be of the same order as the numerical precision of the evaluation of $\boldsymbol{f}$.

The first application of this approach to an iterative solver is due to Anderson et al. [27], who used it to compute derivatives of a Navier–Stokes solver, and later multidisciplinary systems [28]. Martins et al. [29] showed that the complex-step method is generally applicable to any algorithm and described the detailed procedure for its implementation. They also present an alternative way of deriving and understanding the complex step, and connect this to algorithmic differentiation.

The complex-step method requires access to the source code of the given component, and thus cannot be applied to black box components without additional effort. To implement the complex-step method, the source code of the component must be modified so that all real variables and computations are replaced with complex ones. In addition, some intrinsic functions need to be replaced, depending on the programming language. Martins et al. [29] provide a script that facilitates the implementation of the complex-step method to Fortran codes, as well as details for implementation in Matlab, C/C++ and Python.

Figure 2 illustrates the difference between the complex-step and finite-difference formulas. When using the complex-step method, the differencing quotient is evaluated using the imaginary parts of the function values and step size, and the quantity $\boldsymbol{f}(x_j)$ has no imaginary component to subtract.

Figure 2: Derivative approximations $\mathrm{d}f/\mathrm{d}x$ using a forward step in the real (left) and complex (right) axes. Here, f and x are scalars. In the complex-step method, there is no subtraction operation involved because the value of the initial point, $f(x)$, has no imaginary part.

The complex-step approach is now widely used, with applications ranging from the verification of high-fidelity aerostructural derivatives [30, 14] to development of immunology models [31]. In one case, the complex-step was

implemented in an aircraft design optimization framework that involves multiple languages (Python, C, C++ and Fortran) [32], demonstrating the flexibility of this approach.

C. Symbolic Differentiation

Symbolic differentiation is only possible for explicit functions, and can either be done by hand or by appropriate software. For a sequence of composite functions, it is possible to use the chain rule, but symbolic differentiation becomes impossible for general algorithms.

III. Computation of Derivatives in a System

The methods of differentiation presented in the previous section are limited in scope to computing derivatives of a single function or a more complex system without regard to its internal dependencies and structure. These methods are the building blocks of more sophisticated methods that are the focus of this section.

A. Variables, Components and System

To make sure the methods are explained for the most general case and show how the various methods can be derived under a single theoretical framework, it is important to characterize the computational model that is involved and precisely define the relevant terms. In the most general sense, a computational model takes a series of numerical inputs and produces outputs. As previously mentioned, the computational model is assumed to be deterministic. The computational model is ultimately implemented as a computer program. Depending on the type of method, we might take the computational model view or the computer program view. In either case, we sometimes refer to the model or program a *system*, since it is an interconnected series of computations.

It is particularly important to realize the nested nature of the system The most fundamental building blocks of this system are the unary and binary operations. These operations can be combined to obtain more elaborate explicit functions, which are typically expressed in one line of computer code. A more complex computation can be performed by evaluating a sequence of explicit functions V_i, where $i = 1, \ldots, n$. In its simplest form, each function in this sequence depends only on the inputs and the functions that have been computed earlier in the sequence. Thus we can represent such a computation as,

$$v_i = V_i(v_1, v_2, \ldots, v_{i-1}). \tag{5}$$

Here we adopt the convention that the lower case represents the *value* of a variable, and the upper case represents the *function* that computes the value. This is a distinction that will be particularly useful in developing the theory presented herein.

In the more general case, a given function might require values that have not been previously computed, i.e.,

$$v_i = V_i(v_1, v_2, \ldots, v_i, \ldots, v_n). \tag{6}$$

The solution of such systems require numerical methods that can be programmed by using loops. Numerical methods range from simple fixed-point iterations to sophisticated Newton-type algorithms. Note that loops are also used to repeat one or more computations over a computational grid.

One concept that will be used later is that it is always possible to represent any given computation without loops and dependencies — as written in Equation (5) — if we *unroll* all the loops, and represent all values a variable might take in the iteration as a separate variable that is never overwritten.

In the context of this paper, it is useful to generalize any computation that produces an output vector of variables $\boldsymbol{v}_{\text{out}} \subset \boldsymbol{v}$ for given an arbitrary set of input variables $\boldsymbol{v}_{\text{in}} \subset \boldsymbol{v}$. We write this computation as

$$\boldsymbol{v}_{\text{out}} = \boldsymbol{V}(\boldsymbol{v}_{\text{in}}), \tag{7}$$

and call it a *component*. What constitutes a component is somewhat arbitrary, but components are usually defined and organized in a way that helps us understand the overall system. The boundaries of a given component are usually determined by a number of practical factors, such as technical discipline, programming language, data dependencies or developer team.

A given component is in part characterized by the input variables it requires, and by the variables that it outputs. In the process of computing the output variables, a component might also set a number of other variables in $\boldsymbol{v}$ that are neither inputs or output, which we call *intermediate variables*.

When a component is just a series of explicit functions, we can consider the component itself to be an explicit composite function. In cases where the computation of the outputs requires iteration, it is helpful to denote the computation as a vector of *residual equations*,

$$\boldsymbol{r} = \boldsymbol{R}(\boldsymbol{v}) = 0 \tag{8}$$

where the algorithm changes certain components of $\boldsymbol{v}$ until all the residuals converge to zero (or in practice, to within a small specified tolerance). The subset of $\boldsymbol{v}$ that is iterated to achieve the solution of these equations are called the *state variables*.

To relate these concepts to the usual conventions in sensitivity analysis, we now separate the subsets in $\boldsymbol{v}$ into independent variables $\boldsymbol{x}$, state variables $\boldsymbol{y}$ and quantities of interest, $\boldsymbol{f}$. Note that these do not necessary correspond exactly to the component inputs, intermediate variables and outputs, respectively. Using this notation, we can write the residual equations as,

$$\boldsymbol{r} = \boldsymbol{R}(\boldsymbol{x}, \boldsymbol{y}(\boldsymbol{x})) = 0 \tag{9}$$

where $\boldsymbol{y}(\boldsymbol{x})$ denotes the fact that $\boldsymbol{y}$ depends *implicitly* on $\boldsymbol{x}$ through the solution of the residual equations (9). It is the solution of these equations that completely determines $\boldsymbol{y}$ for a given $\boldsymbol{x}$. The functions of interest (usually included in the set of component outputs) also have the same type of variable dependence in the general case, i.e.,

$$\boldsymbol{f} = \boldsymbol{F}(\boldsymbol{x}, \boldsymbol{y}(\boldsymbol{x})). \tag{10}$$

When we compute the values $\boldsymbol{f}$, we assume that the state variables $\boldsymbol{y}$ have already been determined by the solution of the residual equations (9). The dependencies involved in the computation of the functions of interest are represented in Figure 3. For the purposes of this paper, we are ultimately interested in the total derivatives of quantities $\boldsymbol{f}$ with respect to $\boldsymbol{x}$.

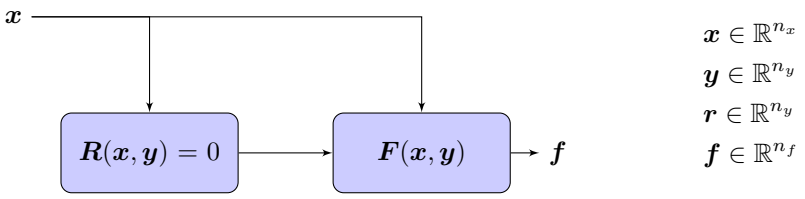


Figure 3: Definition of the variables involved at the solver level, showing the dependency of the quantity of interest on the design variables, both directly and through the residual equations that determine the system states

B. A Unified Framework

In this section, we present the mathematical framework that unifies the methods for computing total derivatives. The methods differ in the extent to which they decompose a system, but they all come from a basic principle: a generalized chain rule.

To arrive at this form of chain rule, we start from the sequence of variables $(v_1, \ldots, v_n)$, whose values are functions of earlier variables, $v_i = V_i(v_1, \ldots, v_{i-1})$. For brevity, $V_i(v_1, \ldots, v_{i-1})$ is written as $v_i(\cdot)$. We define a partial derivative, $\partial V_i / \partial v_j$, of a function V_i with respect to a variable v_j as

$$\frac{\partial V_i}{\partial v_j} = \frac{V_i(v_1, \ldots, v_{j-1}, v_j + h, v_{j+1}, \ldots, v_{i-1}) - V_i(\cdot)}{h}. \tag{11}$$

The total variation Δv_k, due to a perturbation Δv_j can be computed by using the sum of partial derivatives,

$$\Delta v_k = \sum_{l=j}^{k-1} \frac{\partial V_k}{\partial v_l} \Delta v_l \tag{12}$$

where all intermediate Δv's between j and k are computed and used. The total derivative is defined as,

$$\frac{\mathrm{d}v_i}{\mathrm{d}v_j} = \frac{\Delta v_i}{\Delta v_j}, \tag{13}$$

Using the two equations above, we can derive the following equation:

$$\boxed{\frac{\mathrm{d}v_i}{\mathrm{d}v_j} = \delta_{ij} + \sum_{k=j}^{i-1} \frac{\partial V_i}{\partial v_k} \frac{\mathrm{d}v_k}{\mathrm{d}v_j}}, \tag{14}$$

which expresses a total derivative in terms of the other total derivatives and the Jacobian of partial derivatives. Equation (14) is represents the chain rule for a system whose variables are $\boldsymbol{v}$.

To get a better understanding of the structure of the chain rule (14), and the options for performing the computation it represents, we now write it in matrix form. We can write the partial derivatives of the elementary functions V_i with respect to v_i as the square $n \times n$ Jacobian matrix,

$$\boldsymbol{D_V} = \frac{\partial V_i}{\partial v_j} = \begin{bmatrix} 0 & \cdots & & \\ \frac{\partial V_2}{\partial v_1} & 0 & \cdots & \\ \frac{\partial V_3}{\partial v_1} & \frac{\partial V_3}{\partial v_2} & 0 & \cdots \\ \vdots & \vdots & \ddots & \ddots \\ \frac{\partial V_n}{\partial v_1} & \frac{\partial V_n}{\partial v_2} & \cdots & \frac{\partial V_n}{\partial v_{n-1}} \quad 0 \end{bmatrix}, \tag{15}$$

where $\boldsymbol{D}$ is a differential operator. The total derivatives of the variables v_i form another Jacobian matrix of the same size that has a unit diagonal,

$$\boldsymbol{D_v} = \frac{\mathrm{d}v_i}{\mathrm{d}v_j} = \begin{bmatrix} 1 & 0 & \cdots & & \\ \frac{\mathrm{d}v_2}{\mathrm{d}v_1} & 1 & 0 & \cdots & \\ \frac{\mathrm{d}v_3}{\mathrm{d}v_1} & \frac{\mathrm{d}v_3}{\partial v_2} & 1 & 0 & \cdots \\ \vdots & \vdots & \ddots & \ddots & \\ \frac{\mathrm{d}v_n}{\mathrm{d}v_1} & \frac{\mathrm{d}v_n}{\mathrm{d}v_2} & \cdots & \frac{\mathrm{d}v_n}{\mathrm{d}v_{n-1}} \quad 1 \end{bmatrix}. \tag{16}$$

Both of these matrices are lower triangular matrices, due to our assumption that we have unrolled all the loops.

Using this notation, the chain rule (14) can be writen as

$$\boldsymbol{D_v} = \boldsymbol{I} + \boldsymbol{D_V D_v}. \tag{17}$$

Rearranging this, we obtain,

$$\left(\boldsymbol{I} - \boldsymbol{D_V}\right) \boldsymbol{D_v} = \boldsymbol{I}. \tag{18}$$

where all these matrices are square, with size $n \times n$. The matrix $\left(\boldsymbol{I} - \boldsymbol{D_V}\right)$ can be formed by finding the partial derivatives, and then we can solve for the total derivatives $\boldsymbol{D_v}$. Since $\left(\boldsymbol{I} - \boldsymbol{D_V}\right)$ and $\boldsymbol{D_v}$ are inverses of each other, we can further rearrange it to obtain the transposed system:

$$\left(\boldsymbol{I} - \boldsymbol{D_V}\right)^T \boldsymbol{D_v^T} = \boldsymbol{I}. \tag{19}$$

This leads to the following symmetric relationship:

$$\boxed{\left(\boldsymbol{I} - \boldsymbol{D_V}\right) \boldsymbol{D_v} = \quad \boldsymbol{I} \quad = \left(\boldsymbol{I} - \boldsymbol{D_V}\right)^T \boldsymbol{D_v^T}} \tag{20}$$

We call the left and right hand sides of this equation the forward and reverse chain rule equations, respectively. As we will see throughout this paper: *All methods for derivative computation can be derived from one of the forms of this chain rule* (20) *by changing what we mean by "variables"*, which can be seen as a level of decomposition. The various levels of decomposition were shown in Figure 1 and summarized later, in Table 1.

C. Algorithmic Differentiation

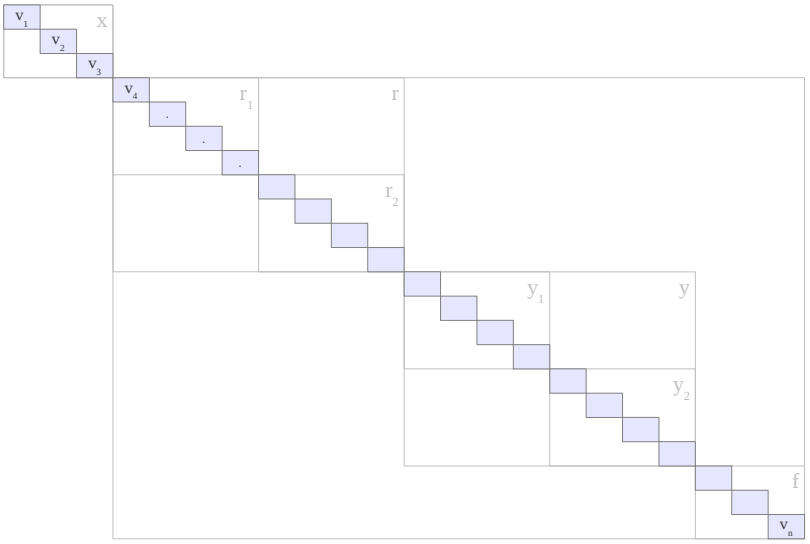
Algorithmic differentiation (AD) — also known as computational differentiation or automatic differentiation — is a well known method based on the systematic application of the differentiation chain rule to computer programs [33, 34]. Although this approach is as accurate as an analytic method, it is potentially much easier to implement since the implementation can be done automatically. To explain AD, we start by describing the basic theory and how it relates to the chain rule identity (20) introduced in the previous section. We then explain how the method is implemented in practice, and show an example.

From the AD perspective, the variables $\boldsymbol{v}$ in the chain rule (20) are all the variables assigned in the computer program, and AD applies the chain rule for every single line in the program. The computer program thus can be considered a sequence of explicit functions V_i, where $i = 1, \ldots, n$. In its simplest form, each function in this sequence depends only on the inputs and the functions that have been computed earlier in the sequence, as expressed in the functional dependence (5).

Again, for this assumption to hold, we assume that all the loops in the program are *unrolled*, and therefore no variables are overwritten and each variable only depends on earlier variables in the sequence. Later, when we explain how AD is implemented, it will become clear that this assumption is not restrictive, as programs iterate the chain rule (and thus the total derivatives) together with the program variables, converging to the correct total derivatives.

In the AD perspective, the independent variables $\boldsymbol{x}$ and the quantities of interest $\boldsymbol{f}$ are assumed to be in the vector of variables $\boldsymbol{v}$. Typically, the design variables are among the v's with lower indices, and the quantities of interest are among the last quantities. Thus, to make clear the connection to the other derivative computation methods, we group these variables as follows,

$$\boldsymbol{v} = [\underbrace{v_1, \ldots, v_{n_x}}_{\boldsymbol{x}}, \ldots, v_j, \ldots, v_i, \ldots, \underbrace{v_{(n-n_f)}, \ldots, v_n}_{\boldsymbol{f}}]^T. \tag{21}$$



$$\boldsymbol{v} = [\underbrace{v_1, \ldots, v_{n_x}}_{\boldsymbol{x}}, \ldots, v_j, \ldots, v_i, \ldots, \underbrace{v_{(n-n_f)}, \ldots, v_n}_{\boldsymbol{f}}]^T$$

Figure 4: Decomposition level for algorithmic differentiation: the variables $\boldsymbol{v}$ are all the variables assigned in the computer program.

The chain rule (14) introduced in the previous section was

$$\frac{\mathrm{d}v_i}{\mathrm{d}v_j} = \delta_{ij} + \sum_{k=j}^{i-1} \frac{\partial V_i}{\partial v_k} \frac{\mathrm{d}v_k}{\mathrm{d}v_j}, \tag{22}$$

where the V represent explicit functions, each defined by a single line in the computer program. The partial derivatives, $\partial V_i / \partial v_k$ can be automatically differentiated symbolically by applying another chain rule within the function defined by the respective line.

The chain rule (22) can be solved in two ways. In the *forward mode*, we choose one v_j and keep j fixed. Then we work our way forward in the index $i = 1, 2, \ldots, n$ until we get the desired total derivative. In the *reverse mode*, on the other hand, we fix v_i (the quantity we want to differentiate) and work our way backward in the index $j = n, n-1, \ldots, 1$ all the way to the independent variables. We now describe these two modes in more detail, and compare the computational costs associated with each of them.

1. Forward Mode

To get a better understanding of the structure of the chain rule (14), and the options for performing that computation, we now write it in the matrix form (18):

$$(\boldsymbol{I} - \boldsymbol{D_V})\,\boldsymbol{D_v} = \boldsymbol{I} \quad \Rightarrow$$

$$\begin{bmatrix} 1 & 0 & \cdots & & \\ -\frac{\partial V_2}{\partial v_1} & 1 & 0 & \cdots & \\ -\frac{\partial V_3}{\partial v_1} & -\frac{\partial V_3}{\partial v_2} & 1 & 0 & \cdots \\ \vdots & \vdots & \ddots & \ddots & \\ -\frac{\partial V_n}{\partial v_1} & -\frac{\partial V_n}{\partial v_2} & \cdots & -\frac{\partial V_n}{\partial v_{n-1}} & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & \cdots & & \\ \frac{\mathrm{d}v_2}{\mathrm{d}v_1} & 1 & 0 & \cdots & \\ \frac{\mathrm{d}v_3}{\mathrm{d}v_1} & \frac{\mathrm{d}v_3}{\mathrm{d}v_2} & 1 & 0 & \cdots \\ \vdots & \vdots & \ddots & \ddots & \\ \frac{\mathrm{d}v_n}{\mathrm{d}v_1} & \frac{\mathrm{d}v_n}{\mathrm{d}v_2} & \cdots & \frac{\mathrm{d}v_n}{\mathrm{d}v_{n-1}} & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & & \\ 0 & 1 & 0 & \cdots & \\ 0 & 0 & 1 & 0 & \cdots \\ \vdots & \vdots & \vdots & \ddots & \ddots \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{23}$$

The terms that we ultimately want to compute are the total derivatives of quantities of interest with respect to the design variables, corresponding to a block in the $\boldsymbol{D_v}$ matrix in the lower left. Using the definition expressed in Equation (1), this block is

$$\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}} = \begin{bmatrix} \frac{\mathrm{d}f_1}{\mathrm{d}x_1} & \cdots & \frac{\mathrm{d}f_1}{\mathrm{d}x_{n_x}} \\ \vdots & \ddots & \vdots \\ \frac{\mathrm{d}f_{n_f}}{\mathrm{d}x_1} & \cdots & \frac{\mathrm{d}f_{n_f}}{\mathrm{d}x_{n_x}} \end{bmatrix} = \begin{bmatrix} \frac{\mathrm{d}v_{(n-n_f)}}{\mathrm{d}v_1} & \cdots & \frac{\mathrm{d}v_{(n-n_f)}}{\mathrm{d}v_{n_x}} \\ \vdots & \ddots & \vdots \\ \frac{\mathrm{d}v_n}{\mathrm{d}v_1} & \cdots & \frac{\mathrm{d}v_n}{\mathrm{d}v_{n_x}} \end{bmatrix}, \tag{24}$$

which is an $n_f \times n_x$ matrix.

The forward mode is equivalent to solving the linear system (24) for one column of $\boldsymbol{D_v}$. Since $(\boldsymbol{I} - \boldsymbol{D_V})$ is a lower triangular matrix, this solution can be accomplished by forward substitution. In the process, We end up computing the derivative of the chosen quantity with respect to all the other variables. The cost of this procedure is similar to the cost of the procedure that computes the v's, and as we will see in Section 3, the forward AD operations are interspersed with the operations that compute the v's in the original computer code.

2. Reverse Mode

The matrix representation for the reverse mode of algorithmic differentiation is given by Equation (19), which expands to,

$$(\boldsymbol{I} - \boldsymbol{D_V})^T\,\boldsymbol{D_v^T} = \boldsymbol{I} \quad \Rightarrow$$

$$\begin{bmatrix} 1 & -\frac{\partial V_2}{\partial v_1} & -\frac{\partial V_3}{\partial v_1} & \cdots & -\frac{\partial V_n}{\partial v_1} \\ 0 & 1 & -\frac{\partial V_3}{\partial v_2} & \cdots & -\frac{\partial V_n}{\partial v_2} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & 1 & -\frac{\partial V_n}{\partial v_{n-1}} \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & \frac{\mathrm{d}v_2}{\mathrm{d}v_1} & \frac{\mathrm{d}v_3}{\mathrm{d}v_1} & \cdots & \frac{\mathrm{d}v_n}{\mathrm{d}v_1} \\ 0 & 1 & \frac{\mathrm{d}v_3}{\mathrm{d}v_2} & \cdots & \frac{\mathrm{d}v_n}{\mathrm{d}v_2} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & 1 & \frac{\mathrm{d}v_n}{\mathrm{d}v_{n-1}} \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & & \\ 0 & 1 & 0 & \cdots & \\ 0 & 0 & 1 & 0 & \cdots \\ \vdots & \vdots & \vdots & \ddots & \ddots \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{25}$$

The block matrix we want to compute is in the upper right section of $\boldsymbol{D_v^T}$ and now its size is $n_x \times n_f$. As with the forward mode, we need to solve this linear system one column at the time, but now each column yields the derivatives of the chosen quantity with respect to all the other variables. Because the matrix $(\boldsymbol{I} - \boldsymbol{DV})^T$ is upper triangular, the system can be solved using back substitution.

3. Implementation and Tools

For readers that are not familiar with AD, we have worked through an example in Appendix A, in which the chain rule (14) is applied both in forward and reverse modes, and the chain rule identity (20) is evaluated numerically for the same example.

The implementation of AD that intersperses lines of code that computes the derivatives with the original code is called the *source code transformation* approach and is exemplified in the code listed in Figure 12. There is another alternative to implementing AD: *operator overloading* [35, 33]. When using this approach, the original code does not change, but the variable types and the operations are redefined. When using operator overloading, each real number v is replaced by a type that includes not only the original real number, but the corresponding derivative as well, i.e., $\bar{v} = (v,\, \mathrm{d}v)$. Then, all operations are redefined such that, in addition to the result of the original operations, they yield the derivative of that operation as well [33].

One significant connection to make is that the complex-step method is equivalent to the forward mode of AD with an operator overloading implementation, as explained by Martins et al. [29].

There are AD tools available for a most programming languages, including Fortran, C/C++[36, 37, 38], and Matlab. They have been extensively developed and provide the user with great functionality, including the calculation of higher-order derivatives and reverse mode options. ADIFOR [39], TAF [40], TAMC [41] and Tapenade [42, 43] are some of the tools available for Fortran that use source transformation. The necessary changes to the source code are made automatically. The operator overloading approach is used in the following tools: AD01 [35], ADOL-F [44], IMAS [45] and OPTIMA90. Although it is in theory possible to have a script make the necessary changes in the source code automatically, none of these tools have this facility and the changes must be done manually.

D. Analytic Methods

Analytic methods are the most accurate and efficient methods available for computing derivatives. However, analytic methods are much more involved that the other methods, since they require detailed knowledge of the computational model and a long implementation time. Analytic methods are applicable when we have a quantity of interest $\boldsymbol{f}$ that depends implicitly on the independent variables of interest $\boldsymbol{x}$, as previously described in Equation (10), which we repeat here for convenience:

$$\boldsymbol{f} = \boldsymbol{F}(\boldsymbol{x}, \boldsymbol{y}(\boldsymbol{x})). \tag{26}$$

The implicit relationship between the state variables $\boldsymbol{y}$ and the independent variables is defined by the solution of a set of residual equations, which we also repeat here:

$$\boldsymbol{r} = \boldsymbol{R}(\boldsymbol{x}, \boldsymbol{y}(\boldsymbol{x})) = 0. \tag{27}$$

By writing the computational model in this form, we have assume a *discrete* analytic approach. This is in contrast to the *continuous* approach, in which the equations are not discretized until later. We will not discuss the continuous approach in this paper, but ample literature can be found on the subject [46, 47, 48, 49], including discussions comparing the two approaches [50, 51].

In this section we derive the two forms of the analytic method — the direct and the adjoint — in two ways. The first derivation follows the derivation that is typically presented in the literature, while the second derivation is based on the chain rule identity (20), and is a new perspective that connects it to algorithmic differentiation.

1. *Traditional Derivation*

As a first step toward obtaining the derivatives that we ultimately want to compute, we use the chain rule to write the total derivative Jacobian of $\boldsymbol{f}$ as

$$\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}} = \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}} + \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}} \frac{\mathrm{d}\boldsymbol{y}}{\mathrm{d}\boldsymbol{x}}, \tag{28}$$

where the result is an $n_f \times n_x$ matrix. As previously mentioned it important to distinguish the total and partial derivatives and define the context. The partial derivatives represent the variation of $\boldsymbol{f} = \boldsymbol{F}(\boldsymbol{x})$ with respect to changes in $\boldsymbol{x}$ for a fixed $\boldsymbol{y}$, while the total derivative $\mathrm{d}\boldsymbol{f}/\mathrm{d}\boldsymbol{x}$ takes into account the change in $\boldsymbol{y}$ that is required to keep the residual equations (27) equal to zero. As we have seen, this distinction depends on the context, i.e., what is considered a total or partial derivative depends on the level that is being considered in the nested system of components.

We should also mention that the partial derivatives can be computed using the methods that we have described earlier (finite differences and complex step), as well as the method that we describe in the next section (algorithmic differentiation).

Since the governing equations must always be satisfied, the total derivative of the residuals (27) with respect to the design variables must also be zero. Thus, using the chain rule we obtain,

$$\frac{\mathrm{d}\boldsymbol{r}}{\mathrm{d}\boldsymbol{x}} = \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}} + \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{y}} \frac{\mathrm{d}\boldsymbol{y}}{\mathrm{d}\boldsymbol{x}} = 0. \tag{29}$$

The computation of the total derivative matrix $\mathrm{d}\boldsymbol{y}/\mathrm{d}\boldsymbol{x}$ in Equations (28) and (29) has a much higher computational cost than any of the partial derivatives, since it requires the solution of the residual equations. The partial derivatives can be computed by differentiating the function $\boldsymbol{F}$ with respect to $\boldsymbol{x}$ while keeping $\boldsymbol{y}$ constant.

The linearized residual equations (29) provide the means for computing the total sensitivity matrix $\mathrm{d}\boldsymbol{y}/\mathrm{d}\boldsymbol{x}$, by rewriting those equations as

$$\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{y}} \frac{\mathrm{d}\boldsymbol{y}}{\mathrm{d}\boldsymbol{x}} = -\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}. \tag{30}$$

Substituting this result into the total derivative equation (28), we obtain

$$\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}} = \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}} - \underbrace{\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}} \overbrace{\left[\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{y}}\right]^{-1}}^{-\frac{\mathrm{d}\boldsymbol{y}}{\mathrm{d}\boldsymbol{x}}}}_{\boldsymbol{\psi}} \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}. \tag{31}$$

The inverse of the square Jacobian matrix $\partial \boldsymbol{R}/\partial \boldsymbol{y}$ is not necessarily explicitly calculated.However, we use the inverse to denote the fact that this matrix needs to be solved as a linear system with some right-hand-side vector.

Equation (31) shows that there are two ways of obtaining the total derivative matrix $\mathrm{d}\boldsymbol{y}/\,\mathrm{d}\boldsymbol{x}$, depending on which right-hand side is chosen for the solution of the linear system.

2. *Direct Method*

The direct method consists in solving the linear system with $-\partial \boldsymbol{R}/\partial \boldsymbol{x}$ as the right-hand side vector, which results in the linear system (30). This linear system needs to be solved for n_x right-hand sides to get the full Jacobian matrix $\mathrm{d}\boldsymbol{y}/\,\mathrm{d}\boldsymbol{x}$. Then, we can use $\mathrm{d}\boldsymbol{y}/\,\mathrm{d}\boldsymbol{x}$ in Equation (28) to obtain the derivatives of interest, $\mathrm{d}\boldsymbol{f}/\,\mathrm{d}\boldsymbol{x}$.

Since the cost of computing derivatives with the direct method is proportional to the number of design variables, n_x, it does not have much of a computational cost advantage relative to finite differencing. In a case where the computational model is a nonlinear system, then the direct method can be advantageous. Both methods require the solution of a system with the same size n_x times, but the direct method solves a linear system, while the finite-difference method solves the original nonlinear one.

3. *Adjoint Method*

Returning to the total sensitivity equation (31), we observe that there is an alternative option for computing the total derivatives: The linear system involving the square Jacobian matrix $\partial \boldsymbol{R}/\partial \boldsymbol{y}$ can be solved with $\partial \boldsymbol{f}/\partial \boldsymbol{y}$ as the right-hand side. This results in the following linear system, which we call the *adjoint equations*,

$$\left[\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{y}}\right]^T \boldsymbol{\psi} = -\left[\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}}\right]^T, \tag{32}$$

where we will call $\boldsymbol{\psi}$ the *adjoint matrix* (of size $n_y \times n_f$). Although this is usually expressed as a vector, we obtain a matrix due to our generalization for the case where $\boldsymbol{f}$ is a vector. The solution of this linear system needs to be solved for each column of $[\partial \boldsymbol{F}/\partial \boldsymbol{y}]^T$, and thus the computational cost is proportional to the number of quantities of interest, n_f. The adjoint vector can then be substituted into Equation (31) to find the total sensitivity,

$$\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}} = \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}} + \boldsymbol{\psi}^T \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}} \tag{33}$$

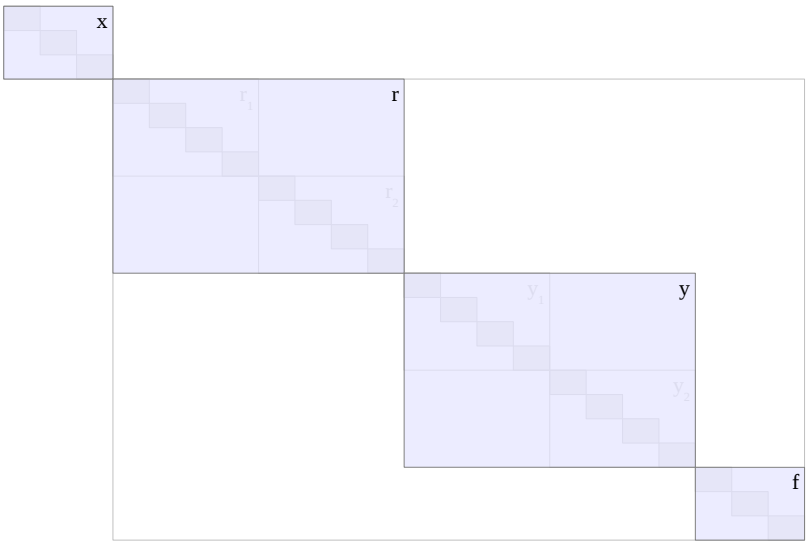
Thus, the cost of computing the total derivative matrix using the adjoint method is independent of the number of design variables, n_x, and instead proportional to the number of quantities of interest, $\boldsymbol{f}$.

Note that the partial derivatives shown in these equations need to be computed using some other method. They can be differentiated symbolically, computed by finite differences, the complex-step method or even AD. The use of AD for these partials has been shown to be particularly effective in the development of analytic methods for PDE solvers [52].

4. *Derivation from Chain Rule*

The first step in this derivation is to clearly define the level of decomposition, i.e., how the variables $\boldsymbol{v}$ are defined. Since the analytic methods apply to coupled systems of equations, the assumption that the Jacobians are lower triangular matrices does no longer apply. Therefore, we first linearize the residuals (27) so that it is possible to write explicit equations for the state variables $\boldsymbol{y}$. We linearize about the converged point $[\boldsymbol{x}_0, \boldsymbol{r}_0, \boldsymbol{y}_0, \boldsymbol{f}_0]^T$, and divide $\boldsymbol{v}$ into

$$\boldsymbol{v}_1 = \Delta \boldsymbol{x}, \quad \boldsymbol{v}_2 = \Delta \boldsymbol{r}, \quad \boldsymbol{v}_3 = \Delta \boldsymbol{y}, \quad \boldsymbol{v}_4 = \Delta \boldsymbol{f}. \tag{34}$$



$$\boldsymbol{v} = [\underbrace{v_1, \ldots, v_{n_x}}_{\boldsymbol{\Delta x}}, \underbrace{v_{(n_x+1)}, \ldots, v_{(n_x+n_y)}}_{\boldsymbol{\Delta r}}, \underbrace{v_{(n_x+n_y+1)}, \ldots, v_{(n_x+2n_y)}}_{\boldsymbol{\Delta y}}, \underbrace{v_{(n-n_f)}, \ldots, t_n}_{\boldsymbol{\Delta f}}]^T.$$

Figure 5: Decomposition level for analytic methods

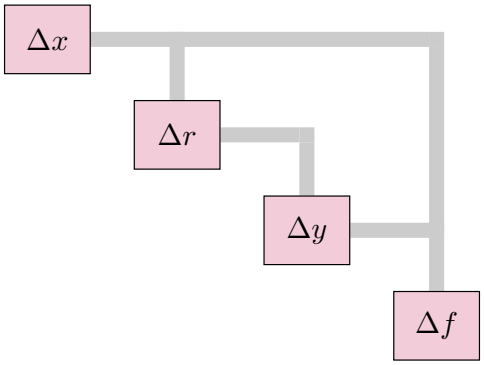


Figure 6: Dependence of the variations in the design variables, residuals, states and quantities of interest for the linearized system

So instead of defining them as every single variable assignment in the computer program, we defined them as variations in the design variables, residuals, state variables and quantities of interest. This decomposition is represented in Figure 5. The dependence of these variations about the converged states is illustrated in Figure 6.

Since $\boldsymbol{x}$ are the only independent variables, we have an initial perturbation $\Delta\boldsymbol{x}$ that leads to a response $\Delta\boldsymbol{r}$. However, we require that $\boldsymbol{r} = 0$ be satisfied when we take a total derivative, and therefore,

$$\boldsymbol{R} = 0 \quad \Rightarrow \quad \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}\Delta\boldsymbol{x} + \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{y}}\Delta\boldsymbol{y} = 0 \tag{35}$$

The solution vector $\Delta\boldsymbol{y}$ from this linear system is used in conjunction with the original perturbation vector $\Delta\boldsymbol{x}$ to

compute the total change in $\Delta \boldsymbol{f}$, i.e.,

$$\boldsymbol{v}_1 = \Delta \boldsymbol{x} \tag{36}$$

$$\boldsymbol{v}_2 = \Delta \boldsymbol{r} = \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}} \Delta \boldsymbol{x} \tag{37}$$

$$\boldsymbol{v}_3 = \Delta \boldsymbol{y} = \left[\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{y}} \right]^{-1} (-\Delta \boldsymbol{r}) \tag{38}$$

$$\boldsymbol{v}_4 = \Delta \boldsymbol{f} = \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}} \Delta \boldsymbol{x} + \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}} \Delta \boldsymbol{y} \tag{39}$$

$$\tag{40}$$

At this point, all variables are functions of only previous variables, so we can apply the forward and reverse chain rule equations (20) to the linearized system with the definition (34). The result is the set of block matrix equations show in Figure 7. The forward chain rule (18) yields the left column, which is the direct method. The right column represents the adjoint method, which is obtained from the reverse chain rule (19).

IV. Derivatives for Multidisciplinary Systems

We now extend the analytic methods derived in the previous section to multidisciplinary systems, where each discipline is seen as one component. We start with the equations in the last row of Figure 7 (also repeated in the first row of Figure 10), which represent the direct and adjoint methods, respectively, for a given system. In this form, the Jacobian matrices for the direct and adjoint methods are block lower and upper triangular, respectively, but not fully lower or upper triangular because we eliminate the inverse of the Jacobian $\partial \boldsymbol{R}/\partial \boldsymbol{y}$ at the expense of creating a linear system that must now be solved. This inverse Jacobian is necessary to obtain an explicit definition for $\boldsymbol{y}$, but we eliminate it because the solution of a linear system is cheaper than matrix inversion of the same size.

The direct and adjoint methods for multidisciplinary systems can be derived by partitioning the various variables by disciplines, as follows,

$$\boldsymbol{R} = [\boldsymbol{R}_1 \boldsymbol{R}_2]^T \quad \boldsymbol{y} = [\boldsymbol{y}_1 \boldsymbol{y}_2]^T \tag{41}$$

where we have assumed two different disciplines. All the design variables are included in $\boldsymbol{x}$. Then, we use these vectors in the equations in the direct and adjoint equations shown in the first row of Figure 10 to obtain the second row in Figure 10. These are the coupled versions of the direct and adjoint methods, respectively. The coupled direct method was first developed by Bloebaum and Sobieski [53, 22, 54]. The coupled adjoint was originally developed by Martins et al. [30].

Figure 8 illustrates the level of decomposition that this involves, using earlier notation. In Figure 10, we can see that this decomposition turns the Jacobian $\partial \boldsymbol{R}/\partial \boldsymbol{y}$ into a matrix with distinct blocks corresponding to the different disciplines. Figure 9(a) shows a graphical view of the two-discipline system.

Figure 9(b) and the third row in Figure 10 show another alternative for obtaining the total derivatives of multidisciplinary systems that was first developed by Sobieski [22] for the direct method, and by Martins et al. [30] for the adjoint method. The advantage of this approach is that we do not need to know the residuals of a given disciplinary solvers, but instead can use the *coupling variables*. To derive the direct and adjoint versions of this approach within our mathematical framework, we define the artificial residual functions

$$\boldsymbol{R}_i = \boldsymbol{Y}_i - \boldsymbol{y}_i, \tag{42}$$

where the $\boldsymbol{y}_i$ vector contains the intermediate variables of the i$^{\text{th}}$ discipline, and $\boldsymbol{Y}_i$ is the vector of functions that explicitly define these intermediate variables. This leads to the third row of equations in Figure 10, which we call the *functional* approach. This contrasts with the *residual* approach that we used previously.

The $\boldsymbol{y}_i$ vector is treated as a vector of state variables in the terminology of systems with residuals. In general, the functions in the vector $\boldsymbol{Y}_i$ can depend on all other intermediate variables in the i$^{\text{th}}$ discipline as well as any other disciplines, signifying that this development allows for coupling to be present among the intermediate variables.

To further generalize the computation of derivatives for multidisciplinary systems, consider a system with two disciplines: one with residuals and one without, as shown in Figure 9(c). In a practical setting, this could correspond to a problem with one discipline that has residuals that are nonlinear in its state variables and another discipline where all intermediate variables are explicitly defined. Because of the high cost of the first discipline, it would be

$$\begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ -\dfrac{\partial \boldsymbol{V}_2}{\partial \boldsymbol{v}_1} & \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} \\ -\dfrac{\partial \boldsymbol{V}_3}{\partial \boldsymbol{v}_1} & -\dfrac{\partial \boldsymbol{V}_3}{\partial \boldsymbol{v}_2} & \boldsymbol{I} & \boldsymbol{0} \\ -\dfrac{\partial \boldsymbol{V}_4}{\partial \boldsymbol{v}_1} & -\dfrac{\partial \boldsymbol{V}_4}{\partial \boldsymbol{v}_2} & -\dfrac{\partial \boldsymbol{V}_4}{\partial \boldsymbol{v}_3} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{I} \\ \dfrac{\mathrm{d}\boldsymbol{v}_2}{\mathrm{d}\boldsymbol{v}_1} \\ \dfrac{\mathrm{d}\boldsymbol{v}_3}{\mathrm{d}\boldsymbol{v}_1} \\ \dfrac{\mathrm{d}\boldsymbol{v}_4}{\mathrm{d}\boldsymbol{v}_1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{0} \end{bmatrix}$$

(a) Forward chain rule

$$\begin{bmatrix} \boldsymbol{I} & -\left[\dfrac{\partial \boldsymbol{V}_2}{\partial \boldsymbol{v}_1}\right]^T & -\left[\dfrac{\partial \boldsymbol{V}_3}{\partial \boldsymbol{v}_1}\right]^T & -\left[\dfrac{\partial \boldsymbol{V}_4}{\partial \boldsymbol{v}_1}\right]^T \\ \boldsymbol{0} & \boldsymbol{I} & -\left[\dfrac{\partial \boldsymbol{V}_3}{\partial \boldsymbol{v}_2}\right]^T & -\left[\dfrac{\partial \boldsymbol{V}_4}{\partial \boldsymbol{v}_2}\right]^T \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I} & -\left[\dfrac{\partial \boldsymbol{V}_4}{\partial \boldsymbol{v}_3}\right]^T \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \dfrac{\mathrm{d}\boldsymbol{v}_4}{\mathrm{d}\boldsymbol{v}_1} \\ \dfrac{\mathrm{d}\boldsymbol{v}_4}{\mathrm{d}\boldsymbol{v}_2} \\ \dfrac{\mathrm{d}\boldsymbol{v}_4}{\mathrm{d}\boldsymbol{v}_3} \\ \boldsymbol{I} \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{I} \end{bmatrix}$$

(b) Reverse chain rule

$$\begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ -\dfrac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}} & \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \left[\dfrac{\partial \boldsymbol{R}}{\partial \boldsymbol{y}}\right]^{-1} & \boldsymbol{I} & \boldsymbol{0} \\ -\dfrac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}} & \boldsymbol{0} & -\dfrac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{I} \\ \dfrac{\mathrm{d}\boldsymbol{r}}{\mathrm{d}\boldsymbol{x}} \\ \dfrac{\mathrm{d}\boldsymbol{y}}{\mathrm{d}\boldsymbol{x}} \\ \dfrac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{0} \end{bmatrix}$$

(c) Forward chain rule (simplified)

$$\begin{bmatrix} \boldsymbol{I} & -\left[\dfrac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}\right]^T & \boldsymbol{0} & -\left[\dfrac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}}\right]^T \\ \boldsymbol{0} & \boldsymbol{I} & \left[\dfrac{\partial \boldsymbol{R}}{\partial \boldsymbol{y}}\right]^{-T} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I} & -\left[\dfrac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}}\right]^T \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \dfrac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}} \\ \dfrac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{r}} \\ \dfrac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{y}} \\ \boldsymbol{I} \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{I} \end{bmatrix}$$

(d) Reverse chain rule (simplified)

$$\begin{aligned} \frac{\mathrm{d}\boldsymbol{r}}{\mathrm{d}\boldsymbol{x}} &= \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}} \\ \frac{\mathrm{d}\boldsymbol{y}}{\mathrm{d}\boldsymbol{x}} &= -\left[\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{y}}\right]^{-1} \frac{\mathrm{d}\boldsymbol{r}}{\mathrm{d}\boldsymbol{x}} \\ \frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}} &= \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}} + \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}} \frac{\mathrm{d}\boldsymbol{y}}{\mathrm{d}\boldsymbol{x}} \end{aligned}$$

(e) Forward substituted (direct method)

$$\begin{aligned} \frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{y}} &= \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}} \\ \frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{r}} &= -\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{y}} \left[\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{y}}\right]^{-1} \\ \frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}} &= \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}} + \frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{r}} \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}} \end{aligned}$$

(f) Back substituted (adjoint method)

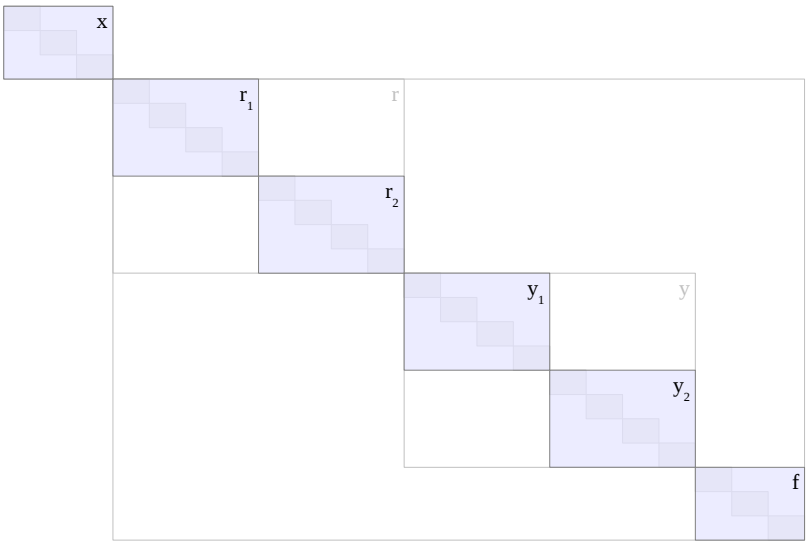
$$\begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} \\ -\dfrac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}} & -\dfrac{\partial \boldsymbol{R}}{\partial \boldsymbol{y}} & \boldsymbol{0} \\ -\dfrac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}} & -\dfrac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{I} \\ \dfrac{\mathrm{d}\boldsymbol{y}}{\mathrm{d}\boldsymbol{x}} \\ \dfrac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{0} \\ \boldsymbol{0} \end{bmatrix}$$

(g) Direct method with the inverse Jacobian eliminated

$$\begin{bmatrix} \boldsymbol{I} & -\left[\dfrac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}\right]^T & -\left[\dfrac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}}\right]^T \\ \boldsymbol{0} & -\left[\dfrac{\partial \boldsymbol{R}}{\partial \boldsymbol{y}}\right]^T & -\left[\dfrac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}}\right]^T \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \dfrac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}} \\ \dfrac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{r}} \\ \boldsymbol{I} \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{I} \end{bmatrix}$$

(h) Adjoint method with the inverse Jacobian eliminated

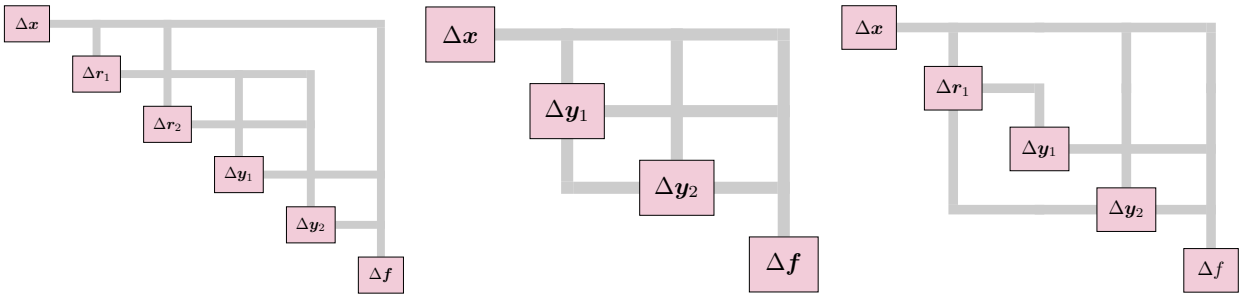
Figure 7: Derivation of the direct (left) and adjoint (right) methods from the forward and reverse chain rule, respectively. The top row shows the chain rule in block form with the four variable vectors; in the second row we replace those vectors with the variables we defined, and the third row shows the equations after the solution of the block matrix, which correspond to the traditional direct and adjoint equations. In the last row, the direct and adjoint methods are presented with the inverse of the $\partial \boldsymbol{R}/\partial \boldsymbol{y}$ matrix eliminated, at the cost of the overall matrix no longer being lower/upper triangular.



$$\boldsymbol{v} = [\underbrace{v_1, \ldots, v_{n_x}}_{\boldsymbol{x}}, \underbrace{\ldots, v_{(n_x+n_{y1})}}_{\boldsymbol{r_1}}, \underbrace{\ldots, v_{(n_x+n_{y1}+n_{y2})}}_{\boldsymbol{r_2}}, \underbrace{\ldots, v_{(n_x+2n_{y1}+n_{y2})}}_{\boldsymbol{y_1}}, \underbrace{\ldots, v_{(n_x+2n_{y1}+2n_{y2})}}_{\boldsymbol{y_2}}, \underbrace{v_{(n-n_f)}, \ldots, t_n}_{\boldsymbol{f}}]^T.$$

Figure 8: Decomposition for the disciplinary level

valuable to be able to use the direct or adjoint method even with the second discipline added. Equations (g) and (h) in Figure 10 show that this is possible with a hybrid formulation that combines elements from the residual and functional approaches. Equations (g) and (h) can be generalized to any number of disciplines with a combination of residuals (using the residual approach) and explicit intermediate variables (using the functional approach).



(a) Residual (b) Functional (c) Hybrid

Figure 9: The different approaches for handling coupled multidisciplinary systems

V. Conclusions

In this paper we achieved what we had set out to accomplish: we derived all known methods for the computation of derivatives for general multiply-nested systems under a single mathematical framework. One of the keys to achieving this unification was the realization that we must view the components of the system at various levels: the elementary function level, the line of code level, the composite function level, the numerical solver level, and the discipline level.

We first presented finite differences, the complex-step method and symbolic differentiation, and discussed their

$$\begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} \\ -\dfrac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}} & -\dfrac{\partial \boldsymbol{R}}{\partial \boldsymbol{y}} & \boldsymbol{0} \\ -\dfrac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}} & -\dfrac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{I} \\ \dfrac{\mathrm{d}\boldsymbol{y}}{\mathrm{d}\boldsymbol{x}} \\ \dfrac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{0} \\ \boldsymbol{0} \end{bmatrix}$$

(a) Direct method

$$\begin{bmatrix} \boldsymbol{I} & -\left[\dfrac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}\right]^T & -\left[\dfrac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}}\right]^T \\ \boldsymbol{0} & -\left[\dfrac{\partial \boldsymbol{R}}{\partial \boldsymbol{y}}\right]^T & -\left[\dfrac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}}\right]^T \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \dfrac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}} \\ \dfrac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{r}} \\ \boldsymbol{I} \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{I} \end{bmatrix}$$

(b) Adjoint method

$$\begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ -\dfrac{\partial \boldsymbol{R}_1}{\partial \boldsymbol{x}} & -\dfrac{\partial \boldsymbol{R}_1}{\partial \boldsymbol{y}_1} & -\dfrac{\partial \boldsymbol{R}_1}{\partial \boldsymbol{y}_2} & \boldsymbol{0} \\ -\dfrac{\partial \boldsymbol{R}_2}{\partial \boldsymbol{x}} & -\dfrac{\partial \boldsymbol{R}_2}{\partial \boldsymbol{y}_1} & -\dfrac{\partial \boldsymbol{R}_2}{\partial \boldsymbol{y}_2} & \boldsymbol{0} \\ -\dfrac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}} & -\dfrac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}_1} & -\dfrac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}_2} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{I} \\ \dfrac{\mathrm{d}\boldsymbol{y}_1}{\mathrm{d}\boldsymbol{x}} \\ \dfrac{\mathrm{d}\boldsymbol{y}_2}{\mathrm{d}\boldsymbol{x}} \\ \dfrac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{0} \end{bmatrix}$$

(c) Coupled direct — residual approach

$$\begin{bmatrix} \boldsymbol{I} & -\left[\dfrac{\partial \boldsymbol{R}_1}{\partial \boldsymbol{x}}\right]^T & -\left[\dfrac{\partial \boldsymbol{R}_2}{\partial \boldsymbol{x}}\right]^T & -\left[\dfrac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}}\right]^T \\ \boldsymbol{0} & -\left[\dfrac{\partial \boldsymbol{R}_1}{\partial \boldsymbol{y}_1}\right]^T & -\left[\dfrac{\partial \boldsymbol{R}_2}{\partial \boldsymbol{y}_1}\right]^T & -\left[\dfrac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}_1}\right]^T \\ \boldsymbol{0} & -\left[\dfrac{\partial \boldsymbol{R}_1}{\partial \boldsymbol{y}_2}\right]^T & -\left[\dfrac{\partial \boldsymbol{R}_2}{\partial \boldsymbol{y}_2}\right]^T & -\left[\dfrac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}_2}\right]^T \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \dfrac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}} \\ \dfrac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{r}_1} \\ \dfrac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{r}_2} \\ \boldsymbol{I} \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{I} \end{bmatrix}$$

(d) Coupled adjoint — residual approach

$$\begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ -\dfrac{\partial \boldsymbol{Y}_1}{\partial \boldsymbol{x}} & \boldsymbol{I} & -\dfrac{\partial \boldsymbol{Y}_1}{\partial \boldsymbol{y}_2} & \boldsymbol{0} \\ -\dfrac{\partial \boldsymbol{Y}_2}{\partial \boldsymbol{x}} & -\dfrac{\partial \boldsymbol{Y}_2}{\partial \boldsymbol{y}_1} & \boldsymbol{I} & \boldsymbol{0} \\ -\dfrac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}} & -\dfrac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}_1} & -\dfrac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}_2} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{I} \\ \dfrac{\mathrm{d}\boldsymbol{y}_1}{\mathrm{d}\boldsymbol{x}} \\ \dfrac{\mathrm{d}\boldsymbol{y}_2}{\mathrm{d}\boldsymbol{x}} \\ \dfrac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{0} \end{bmatrix}$$

(e) Coupled direct — functional approach

$$\begin{bmatrix} \boldsymbol{I} & -\left[\dfrac{\partial \boldsymbol{Y}_1}{\partial \boldsymbol{x}}\right]^T & -\left[\dfrac{\partial \boldsymbol{Y}_2}{\partial \boldsymbol{x}}\right]^T & -\left[\dfrac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}}\right]^T \\ \boldsymbol{0} & \boldsymbol{I} & -\left[\dfrac{\partial \boldsymbol{Y}_2}{\partial \boldsymbol{y}_1}\right]^T & -\left[\dfrac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}_1}\right]^T \\ \boldsymbol{0} & -\left[\dfrac{\partial \boldsymbol{Y}_1}{\partial \boldsymbol{y}_2}\right]^T & \boldsymbol{I} & -\left[\dfrac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}_2}\right]^T \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \dfrac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}} \\ \dfrac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{y}_1} \\ \dfrac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{y}_2} \\ \boldsymbol{I} \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{I} \end{bmatrix}$$

(f) Coupled adjoint — functional approach

$$\begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ -\dfrac{\partial \boldsymbol{R}_1}{\partial \boldsymbol{x}} & -\dfrac{\partial \boldsymbol{R}_1}{\partial \boldsymbol{y}_1} & -\dfrac{\partial \boldsymbol{R}_1}{\partial \boldsymbol{y}_2} & \boldsymbol{0} \\ -\dfrac{\partial \boldsymbol{Y}_2}{\partial \boldsymbol{x}} & -\dfrac{\partial \boldsymbol{Y}_2}{\partial \boldsymbol{y}_1} & \boldsymbol{I} & \boldsymbol{0} \\ -\dfrac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}} & -\dfrac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}_1} & -\dfrac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}_2} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{I} \\ \dfrac{\mathrm{d}\boldsymbol{y}_1}{\mathrm{d}\boldsymbol{x}} \\ \dfrac{\mathrm{d}\boldsymbol{y}_2}{\mathrm{d}\boldsymbol{x}} \\ \dfrac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{0} \end{bmatrix}$$

(g) Hybrid direct

$$\begin{bmatrix} \boldsymbol{I} & -\left[\dfrac{\partial \boldsymbol{R}_1}{\partial \boldsymbol{x}}\right]^T & -\left[\dfrac{\partial \boldsymbol{Y}_2}{\partial \boldsymbol{x}}\right]^T & -\left[\dfrac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}}\right]^T \\ \boldsymbol{0} & -\left[\dfrac{\partial \boldsymbol{R}_1}{\partial \boldsymbol{y}_1}\right]^T & -\left[\dfrac{\partial \boldsymbol{Y}_2}{\partial \boldsymbol{y}_1}\right]^T & -\left[\dfrac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}_1}\right]^T \\ \boldsymbol{0} & -\left[\dfrac{\partial \boldsymbol{R}_1}{\partial \boldsymbol{y}_2}\right]^T & \boldsymbol{I} & -\left[\dfrac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}_2}\right]^T \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \dfrac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}} \\ \dfrac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{r}_1} \\ \dfrac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{y}_2} \\ \boldsymbol{I} \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{I} \end{bmatrix}$$

(h) Hybrid adjoint

Figure 10: Derivation of the coupled direct (left column) and adjoint (right column) methods. The first row shows the original single system equations; in the second row we divide the residuals and state variables into two blocks; in the third row we present the functional approach to the coupled sensitivity equations; and the last row shows how the residual and functional approaches can be combined.

	Monolithic	Analytic	Multidisciplinary analytic	AD
Level of decomposition	Black box	Solver	Discipline	Line of code
Differentiation method	FD/CS	Any	Any	Symbolic
Linear solution	Trivial	Numerical	Numerical (block)	Forward-substitution Back-substitution

Table 1: Classification of the methods for computing derivatives with respect to the level of decomposition, differentiation method, and strategy for solving the linear system.

relative advantages and disadvantages. These methods can be used at any level and are the basic building blocks for more sophisticated methods.

The second key to unifying the various derivative computation methods was the derivation of the chain rule identity (20), which shows the elegant symmetry and connections between the various methods and their respective modes. Algorithmic differentiation can be derived from this chain rule by considering the variables and respective functions to be the lines in a given computer program.

To derive the analytic methods, we linearized the system and defined the variables to be perturbations about the converged solution. Using this definition, we retrieved the well-known direct and adjoint equations from the forward chain rule (18) and reverse chain rule (19), respectively.

Since we derived these from the same equations, we showed the connection between the forward mode of algorithmic differentiation and the direct method, as well as the connection between the adjoint method and the reverse mode of algorithmic differentiation.

Finally, the analytic methods were generalized for the case of multidisciplinary systems, where multiple solvers are coupled. Two different approaches — the residual approach and the functional approach — were shown to be possible for both the coupled direct and couple adjoint methods, resulting in four possible combinations. In addition, we showed that it is possible to combine the residual and functional approaches to create a hybrid approach. This flexibility is valuable, since it is not always possible to use one or the other, due to limitations of the disciplinary solvers.

In summary, each of the methods for computing derivatives shares a common origin, but they differ in three aspects. Table 1 classifies each of these methods in terms of the level of decomposition at which the generalized chain rule is applied, the differentiation method used to assemble the Jacobian of partial derivatives, and the strategy for solving the linear system that results.

Acknowledgments

We would like to thank Graeme J. Kennedy for his valuable insights.

References

[1] Saltelli, A., Ratto, M., Andres, T., Campolongo, F., Cariboni, J., Gatelli, D., Saisana, M., and Tarantola, S., *Global Sensitivity Analysis: The Primer*, John Wiley & Sons Ltd., 2008.

[2] Hicken, J. and Zingg, D., "A Parallel Newton–Krylov Solver for the Euler Equations Discretized Using Simultaneous Approximation Terms," *AIAA Journal*, Vol. 46, No. 11, 2008.

[3] Biros, G. and Ghattas, O., "Parallel Largrange–Newton–Krylov–Schur methods for PDE-constrained optimization. Part I: The Krylov–Schur solver," *SIAM Journal on Scientific Computing*, Vol. 27, No. 2, 2005, pp. 687–713.

[4] Biros, G. and Ghattas, O., "Parallel Largrange–Newton–Krylov–Schur methods for PDE-constrained optimization. Part II: The Largrange–Newton solver and its application to optimal control of steady viscous flows," *SIAM Journal on Scientific Computing*, Vol. 27, No. 2, 2005, pp. 687–713.

[5] Kennedy, G. J. and Martins, J. R. R. A., "Parallel Solution Methods for Aerostructural Analysis and Design Optimization," *Proceedings of the 13th AIAA/ISSMO Multidisciplinary Analysis Optimization Conference*, Forth Worth, TX, September 2010, AIAA 2010-9308.

[6] Dennis, J. and Moreé, J. J., "Quasi-Newton Methods, Motivation and Theory," *SIAM Review*, Vol. 19, No. 1, 1977, pp. 46–89.

[7] Gill, P. E., Murray, W., and Saunders, M. A., "SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization," *SIAM Review*, Vol. 47, No. 1, 2005, pp. 99–131. doi:10.1137/S0036144504446096.

[8] Chung, H. S. and Alonso, J. J., "Using gradients to construct response surface models for high-dimensional design optimization problems," *39th AIAA Aerospace Sciences Meeting*, Reno, NV, January 2001, AIAA-2001-0922.

[9] James, K., Hansen, J. S., and Martins, J. R. R. A., "Structural topology optimization for multiple load cases using a dynamic aggregation technique," *Engineering Optimization*, Vol. 41, No. 12, December 2009, pp. 1103–1118. doi:10.1080/03052150902926827.

[10] Sigmund, O., "On the usefulness of non-gradient approaches in topology optimization," *Structural and Multidisciplinary Optimization*, Vol. 43, 2011, pp. 589—596. doi:10.1007/s00158-011-0638-7.

[11] Lee, E., James, K. A., and Martins, J. R. R. A., "Stress-Constrained Topology Optimization with Design-Dependent Loading," *Structural and Multidisciplinary Optimization*, 2012. doi:10.1007/s00158-012-0780-x, (In press).

[12] Lee, E. and Martins, J. R. R. A., "Structural Topology Optimization with Design-Dependent Pressure Loads," *Computer Methods in Applied Mechanics and Engineering*, 2012. doi:10.1016/j.cma.2012.04.007, (In press).

[13] Martins, J. R. R. A., Alonso, J. J., and Reuther, J. J., "High-Fidelity Aerostructural Design Optimization of a Supersonic Business Jet," *Journal of Aircraft*, Vol. 41, No. 3, 2004, pp. 523–530. doi:10.2514/1.11478.

[14] Kenway, G. K. W., Kennedy, G. J., and Martins, J. R. R. A., "A Scalable Parallel Approach for High-Fidelity Aerostructural Analysis and Optimization," *53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, Honolulu, HI, April 2012.

[15] Mader, C. A. and Martins, J. R. R. A., "An Automatic Differentiation Discrete Adjoint Approach for Time-Spectral Computational Fluid Dynamics," *AIAA Journal*, 2012, (In press).

[16] Mader, C. A. and Martins, J. R. R. A., "Computation of Aircraft Stability Derivatives Using an Automatic Differentiation Adjoint Approach," *AIAA Journal*, Vol. 49, No. 12, 2011, pp. 2737–2750. doi:10.2514/1.55678.

[17] Barthelemy, J.-F. and Sobieszczanski-Sobieski, J., "Optimum Sensitivity Derivatives of Objective Functions in Nonlinear Programming," *AIAA Journal*, Vol. 21, 1982, pp. 913–915.

[18] Adelman, H. M. and Haftka, R. T., "Sensitivity Analysis of Discrete Structural Systems," *AIAA Journal*, Vol. 24, No. 5, 1986, pp. 823–832. doi:10.2514/3.48671.

[19] van Keulen, F., Haftka, R., and Kim, N., "Review of options for structural design sensitivity analysis. Part 1: Linear systems," *Computer Methods in Applied Mechanics and Engineering*, Vol. 194, 2005, pp. 3213–3243.

[20] Haug, E. J., Choi, K. K., and Komkov, V., *Design Sensitivity Analysis of Structural Systems*, Vol. 177 of *Mathematics in Science and Engineering*, Academic Press, 1986.

[21] Sobieszczanski-Sobieski, J., "Sensitivity Analysis and Multidisciplinary Optimization for Aircraft Design: Recent Advances and Results," *Journal of Aircraft*, Vol. 27, No. 12, December 1990, pp. 993–1001. doi:10.2514/3.45973.

[22] Sobieszczanski-Sobieski, J., "Sensitivity of Complex, Internally Coupled Systems," *AIAA Journal*, Vol. 28, No. 1, 1990, pp. 153–160.

[23] Sobieszczanski-Sobieski, J., "Higher Order Sensitivity Analysis of Complex, Coupled Systems," *AIAA Journal*, Vol. 28, No. 4, 1990, pp. 756–758, Technical Note.

[24] Lyness, J. N., "Numerical algorithms based on the theory of complex variable," *Proceedings — ACM National Meeting*, Thompson Book Co., Washington DC, 1967, pp. 125–133.

[25] Lyness, J. N. and Moler, C. B., "Numerical Differentiation of Analytic Functions," *SIAM Journal on Numerical Analysis*, Vol. 4, No. 2, 1967, pp. 202–210.

[26] Squire, W. and Trapp, G., "Using Complex Variables to Estimate Derivatives of Real Functions," *SIAM Review*, Vol. 40, No. 1, 1998, pp. 110–112.

[27] Anderson, W. K., Newman, J. C., Whitfield, D. L., and Nielsen, E. J., "Sensitivity analysis for the Navier–Stokes equations on unstructured meshes using complex variables," *AIAA Paper* 99-3294, 1999.

[28] Newman III, J. C., Whitfield, D. L., and Anderson, W. K., "Step-Size Independent Approach for Multidisciplinary Sensitivity Analysis," *Journal of Aircraft*, Vol. 40, No. 3, 2003, pp. 566–573.

[29] Martins, J. R. R. A., Sturdza, P., and Alonso, J. J., "The Complex-Step Derivative Approximation," *ACM Transactions on Mathematical Software*, Vol. 29, No. 3, 2003, pp. 245–262. doi:10.1145/838250.838251.

[30] Martins, J. R. R. A., Alonso, J. J., and Reuther, J. J., "A Coupled-Adjoint Sensitivity Analysis Method for High-Fidelity Aero-Structural Design," *Optimization and Engineering*, Vol. 6, No. 1, March 2005, pp. 33–62. doi:10.1023/B:OPTE.0000048536.47956.62.

[31] Luzyanina, T. and Bocharov, G., "Critical Issues in the Numerical Treatment of the Parameter Estimation Problems in Immunology," *Journal of Computational Mathematics*, Vol. 30, No. 1, Jan. 2012, pp. 59–79.

[32] Sturdza, P., "An Aerodynamic Design Method for Supersonic Natural Laminar Flow Aircraft," PhD thesis 153–159, Stanford University, Stanford, California, 2004.

[33] Griewank, A., *Evaluating Derivatives*, SIAM, Philadelphia, 2000.

[34] Naumann, U., *The Art of Differentiating Computer Programs — An Introduction to Algorithmic Differentiation*, SIAM, 2011.

[35] Pryce, J. D. and Reid, J. K., "AD01, a Fortran 90 Code for Automatic Differentiation," Report RAL-TR-1998-057, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire, OX11 OQX, U.K., 1998.

[36] Griewank, A., Juedes, D., and Utke, J., "Algorithm 755: ADOL-C: a package for the automatic differentiation of algorithms written in C/C++," *ACM Transactions on Mathematical Software*, Vol. 22, No. 2, 1996, pp. 131–167.

[37] Bendtsen, C. and Stauning, O., "FADBAD, a flexible C++ package for automatic differentiation — using the forward and backward methods," Tech. Rep. IMM-REP-1996-17, Technical University of Denmark, DK-2800 Lyngby, Denmark, 1996.

[38] Bischof, C. H., Roh, L., and Mauer-Oats, A. J., "ADIC: an extensible automatic differentiation tool for ANSI-C," *Software — Practice and Experience*, Vol. 27, No. 12, 1997, pp. 1427–1456.

[39] Carle, A. and Fagan, M., "ADIFOR 3.0 Overview," Tech. Rep. CAAM-TR-00-02, Rice University, 2000.

[40] Giering, R. and Kaminski, T., "Applying TAF to Generate Efficient Derivative Code of Fortran 77-95 Programs," *Proceedings of GAMM 2002, Augsburg, Germany*, 2002.

[41] Gockenbach, M. S., "Understanding Code Generated by TAMC," IAAA Paper TR00-29, Department of Computational and Applied Mathematics, Rice University, Texas, USA, 2000.

[42] Hascoët, L. and Pascual, V., "TAPENADE 2.1 User's Guide," Technical report 300, INRIA, 2004.

[43] Pascual, V. and Hascoët, L., "Extension of TAPENADE Towards Fortran 95," *Automatic Differentiation: Applications, Theory, and Tools*, edited by H. M. Bücker, G. Corliss, P. Hovland, U. Naumann, and B. Norris, Lecture Notes in Computational Science and Engineering, Springer, 2005.

[44] Shiriaev, D., "ADOL–F Automatic Differentiation of Fortran Codes," *Computational Differentiation: Techniques, Applications, and Tools*, edited by M. Berz, C. H. Bischof, G. F. Corliss, and A. Griewank, SIAM, Philadelphia, Penn., 1996, pp. 375–384.

[45] Rhodin, A., "IMAS — Integrated Modeling and Analysis System for the solution of optimal control problems," *Computer Physics Communications*, , No. 107, 1997, pp. 21–38.

[46] Jameson, A., "Aerodynamic Design via Control Theory," *Journal of Scientific Computing*, Vol. 3, No. 3, sep 1988, pp. 233–260.

[47] Jameson, A., Martinelli, L., and Pierce, N. A., "Optimum Aerodynamic Design Using the Navier–Stokes Equations," *Theoretical and Computational Fluid Dynamics*, Vol. 10, 1998, pp. 213–237.

[48] Anderson, W. K. and Venkatakrishnan, V., "Aerodynamic design optimization on unstructured grids with a continuous adjoint formulation," *Computers and Fluids*, Vol. 28, No. 4, 1999, pp. 443–480.

[49] Giles, M. B. and Pierce, N. A., "An Introduction to the Adjoint Approach to Design," *Flow, Turbulence and Combustion*, Vol. 65, 2000, pp. 393–415.

[50] Nadarajah, S. and Jameson, A., "A Comparison of the Continuous and Discrete Adjoint Approach to Automatic Aerodynamic Optimization," *Proceedings of the 38th AIAA Aerospace Sciences Meeting and Exhibit*, Reno, NV, 2000, AIAA 2000-0667.

[51] Dwight, R. P. and Brezillion, J., "Effect of Approximations of the Discrete Adjoint on Gradient-Based Optimization," *AIAA Journal*, Vol. 44, No. 12, 2006, pp. 3022–3031.

[52] Mader, C. A., Martins, J. R. R. A., Alonso, J. J., and van der Weide, E., "ADjoint: An Approach for the Rapid Development of Discrete Adjoint Solvers," *AIAA Journal*, Vol. 46, No. 4, April 2008, pp. 863–873. doi:10.2514/1.29123.

[53] Bloebaum, C., "Global Sensitivity Analysis in Control-Augmented Structural Synthesis," *Proceedings of the 27th AIAA Aerospace Sciences Meeting*, Reno, NV, January 1989, AlAA 1989-0844.

[54] Bloebaum, C., Hajela, P., and Sobieszczanski-Sobieski, J., "Non-hierarchic system decomposition in structural optimization," *Proceedings of the 3rd USAF/NASA Symposium on Recent Advances in Multidisciplinary Analysis and Optimization*, San Francisco, CA, 1990.

[55] Hascoët, L., "TAPENADE: A tool for Automatic Differentiation of programs," *Proceedings of* 4^{th} *European Congress on Computational Methods, ECCOMAS'2004, Jyvaskyla, Finland*, 2004.

[56] Cusdin, P. and Müller, J.-D., "On the Performance of Discrete Adjoint CFD Codes using Automatic Differentiation," *International Journal of Numerical Methods in Fluids*, Vol. 47, No. 6-7, 2005, pp. 939–945.

[57] Fagan, M. and Carle, A., "Reducing Reverse-Mode Memory Requirements by Using Profile-Driven Checkpointing," *Future Generation Comp. Syst.*, Vol. 21, No. 8, 2005, pp. 1380–1390.

[58] Giering, R., Kaminski, T., and Slawig, T., "Generating Efficient Derivative Code with TAF: Adjoint and Tangent Linear Euler Flow Around an Airfoil." *Future Generation Comp. Syst.*, Vol. 21, No. 8, 2005, pp. 1345–1355.

[59] Heimbach, P., Hill, C., and Giering, R., "An Efficient Exact Adjoint of the Parallel MIT General Circulation Model, Generated via Automatic Differentiation." *Future Generation Comp. Syst.*, Vol. 21, No. 8, 2005, pp. 1356–1371.

Appendix A: Algorithmic Differentiation Example

To explain the practical implementation of AD, we use an example. Suppose we want to compute the derivatives for the following vector function,

$$\begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} (x_1 x_2 + \sin x_1)\left(3x_2^2 + 6\right) \\ x_1 x_2 + x_2^2 \end{bmatrix}, \tag{43}$$

where the independent variables are $\boldsymbol{x}^T = [x_1, x_2]$ and the quantities of interest are $\boldsymbol{f}^T = [f_1, f_2]$. This problem is simple enough that we can use symbolic differentiation to find the Jacobian, which is the 2×2 matrix,

$$\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}} = \begin{bmatrix} \dfrac{\mathrm{d}f_1}{\mathrm{d}x_1} & \dfrac{\mathrm{d}f_1}{\mathrm{d}x_2} \\ \dfrac{\mathrm{d}f_2}{\mathrm{d}x_1} & \dfrac{\mathrm{d}f_2}{\mathrm{d}x_2} \end{bmatrix} = \begin{bmatrix} (x_2 + \cos x_1)\left(3x_2^2 + 6\right) & x_1\left(3x_2^2 + 6\right) + 6x_2\left(x_1 x_2 + \sin x_1\right) \\ x_2 & x_1 + 2x_2 \end{bmatrix}. \tag{44}$$

However, the point of this example is to show how the chain rule can be systematically applied in an automated fashion. To illustrate this more clearly, we assume that the function above is computed using a computer program that performs the following series of unary and binary operation:

$$\begin{aligned}
&v_1 = x_1 && v_2 = x_2 \\
&v_3 = V_3(v_1) = \sin v_1 && v_4 = V_4(v_1, v_2) = v_1 v_2 \\
&v_5 = V_5(v_2) = v_2^2 && v_6 = 3 \\
&v_7 = V_7(v_3, v_4) = v_3 + v_4 && v_8 = V_8(v_5, v_6) = v_5 v_6 \\
&v_9 = 6 && v_{10} = V_{10}(v_8, v_9) = v_8 + v_9 \\
&v_{11} = V_{11}(v_7, v_{10}) = v_7 v_{10} \quad (= f_1) && v_{12} = V_{12}(v_4, v_5) = v_4 + v_5 \quad (= f_2)
\end{aligned} \tag{45}$$

Thus in this case, $n = 12$. The Fortran source code corresponding to these computations is shown in Figure 11, Appendix B.

To use the chain rule (22) in forward mode to compute $\mathrm{d}f_1/\mathrm{d}x_1$ (which is the same as $\mathrm{d}v_{11}/\mathrm{d}v_1$), we set $j = 1$, and then vary $i = 1, 2, \ldots, 11$. Note that in the sum in the chain rule, we only include the k's for which $\partial V_i/\partial v_k \neq 0$. Since each operation in this case has at most two terms, the sums in the chain rule have at most two terms. The sequence given by the procedure is as follows:

$$\begin{aligned}
&\frac{\mathrm{d}v_1}{\mathrm{d}v_1} = 1 \\
&\frac{\mathrm{d}v_2}{\mathrm{d}v_1} = 0 \\
&\frac{\mathrm{d}v_3}{\mathrm{d}v_1} = \frac{\partial V_3}{\partial v_1}\frac{\mathrm{d}v_1}{\mathrm{d}v_1} = \cos v_1 \times 1 = \cos v_1 \\
&\frac{\mathrm{d}v_4}{\mathrm{d}v_1} = \frac{\partial V_4}{\partial v_1}\frac{\mathrm{d}v_1}{\mathrm{d}v_1} + \frac{\partial V_4}{\partial v_2}\frac{\mathrm{d}v_2}{\mathrm{d}v_1} = v_2 \times 1 + v_1 \times 0 = v_2 \\
&\frac{\mathrm{d}v_5}{\mathrm{d}v_1} = \frac{\partial V_5}{\partial v_2}\frac{\mathrm{d}v_2}{\mathrm{d}v_1} = 2v_2 \times 0 = 0 \\
&\frac{\mathrm{d}v_6}{\mathrm{d}v_1} = 0 \\
&\frac{\mathrm{d}v_7}{\mathrm{d}v_1} = \frac{\partial V_7}{\partial v_3}\frac{\mathrm{d}v_3}{\mathrm{d}v_1} + \frac{\partial V_7}{\partial v_4}\frac{\mathrm{d}v_4}{\mathrm{d}v_1} = 1 \times \cos v_1 + 1 \times v_2 = \cos v_1 + v_2 \\
&\frac{\mathrm{d}v_8}{\mathrm{d}v_1} = \frac{\partial V_8}{\partial v_5}\frac{\mathrm{d}v_5}{\mathrm{d}v_1} + \frac{\partial V_8}{\partial v_6}\frac{\mathrm{d}v_6}{\mathrm{d}v_1} = v_6 \times 0 + v_5 \times 0 = 0 \\
&\frac{\mathrm{d}v_9}{\mathrm{d}v_1} = 0 \\
&\frac{\mathrm{d}v_{10}}{\mathrm{d}v_1} = \frac{\partial V_{10}}{\partial v_8}\frac{\mathrm{d}v_8}{\mathrm{d}v_1} + \frac{\partial V_{10}}{\partial v_9}\frac{\mathrm{d}v_9}{\mathrm{d}v_1} = 1 \times 0 + 1 \times 0 = 0 \\
&\frac{\mathrm{d}v_{11}}{\mathrm{d}v_1} = \frac{\partial V_{11}}{\partial v_7}\frac{\mathrm{d}v_7}{\mathrm{d}v_1} + \frac{\partial V_{11}}{\partial v_{10}}\frac{\mathrm{d}v_{10}}{\mathrm{d}v_1} = v_{10}\left(\cos v_1 + v_2\right) + v_7 \times 0 = \left(3v_2^2 + 6\right)\left(\cos v_1 + v_2\right)
\end{aligned}$$

The above operation can be interspersed with the original code above. Figure 12 shows the code that results from running the algorithmic differentiation tool Tapenade [55] through the original code (Figure 11).

For the reverse mode, we use the chain rule in reverse, i.e., we set $i = 11$ and loop backwards such that $j = 11, 10, \ldots, 1$. The

resulting sequence of computations is:

$$
\begin{aligned}
\frac{\mathrm{d}v_{11}}{\mathrm{d}v_{11}} &= 1 \\
\frac{\mathrm{d}v_{11}}{\mathrm{d}v_{10}} &= \frac{\mathrm{d}v_{11}}{\mathrm{d}v_{11}}\frac{\partial V_{11}}{\partial v_{10}} = 1 \times v_7 = v_7 \\
\frac{\mathrm{d}v_{11}}{\mathrm{d}v_{9}} &= \frac{\mathrm{d}v_{11}}{\mathrm{d}v_{10}}\frac{\partial V_{10}}{\partial v_{9}} = v_7 \times 1 = v_7 \\
\frac{\mathrm{d}v_{11}}{\mathrm{d}v_{8}} &= \frac{\mathrm{d}v_{11}}{\mathrm{d}v_{10}}\frac{\partial V_{10}}{\partial v_{8}} = v_7 \times 1 = v_7 \\
\frac{\mathrm{d}v_{11}}{\mathrm{d}v_{7}} &= \frac{\mathrm{d}v_{11}}{\mathrm{d}v_{11}}\frac{\partial V_{11}}{\partial v_{7}} = 1 \times v_{10} = v_{10} \\
\frac{\mathrm{d}v_{11}}{\mathrm{d}v_{6}} &= \frac{\mathrm{d}v_{11}}{\mathrm{d}v_{8}}\frac{\partial V_{8}}{\partial v_{6}} = v_7 \times v_5 = v_7 v_5 \\
\frac{\mathrm{d}v_{11}}{\mathrm{d}v_{5}} &= \frac{\mathrm{d}v_{11}}{\mathrm{d}v_{8}}\frac{\partial V_{8}}{\partial v_{5}} = v_7 \times v_6 = v_7 v_6 \\
\frac{\mathrm{d}v_{11}}{\mathrm{d}v_{4}} &= \frac{\mathrm{d}v_{11}}{\mathrm{d}v_{7}}\frac{\partial V_{7}}{\partial v_{4}} = v_{10} \times 1 = v_{10} \\
\frac{\mathrm{d}v_{11}}{\mathrm{d}v_{3}} &= \frac{\mathrm{d}v_{11}}{\mathrm{d}v_{7}}\frac{\partial V_{7}}{\partial v_{3}} = v_{10} \times 1 = v_{10} \\
\frac{\mathrm{d}v_{11}}{\mathrm{d}v_{2}} &= \frac{\mathrm{d}v_{11}}{\mathrm{d}v_{4}}\frac{\partial V_{4}}{\partial v_{2}} + \frac{\mathrm{d}v_{11}}{\mathrm{d}v_{5}}\frac{\partial V_{5}}{\partial v_{2}} = v_{10} \times v_1 + v_7 v_6 \times 2v_2 = \left(3v_2^2 + 6\right) v_1 + 6v_2 \left(\sin v_1 + v_1 v_2\right) \\
\frac{\mathrm{d}v_{11}}{\mathrm{d}v_{1}} &= \frac{\mathrm{d}v_{11}}{\mathrm{d}v_{3}}\frac{\partial V_{3}}{\partial v_{1}} + \frac{\mathrm{d}v_{11}}{\mathrm{d}v_{4}}\frac{\partial V_{4}}{\partial v_{1}} = v_{10} \times \cos v_1 + v_{10} \times v_2 = \left(3v_2^2 + 6\right)\left(\cos v_1 + v_2\right)
\end{aligned}
$$

Note that unlike the forward mode, the above computation cannot be interspersed with the original code. This is because throughout the reverse computation, the values for the v's are required (e.g., v_7 in the second line). Thus the original code must run first, and all the intermediate variables $\boldsymbol{v}$ must be stored for later use in the reverse mode computations. The dependency graph of the whole code must also be stored before the reverse computation is performed. This contrasts with the forward mode for which the only terms computed for the current derivative $\mathrm{d}v_i/\,\mathrm{d}v_j$ are for those variables that v_i depends on. For the reverse mode, the terms that must be computed are for those functions v_j affects, which is information that must be determined and stored beforehand. For an iterative code, the memory requirements for the reverse mode can therefore be prohibitive, although there has been some progress toward alleviating these requirements [56, 57, 58, 59]. Figure 15 shows the result of running the algorithmic tool Tapenade [55] in reverse mode.

To complete the example, we now form the chain rule equation (18) and evaluate it at $\boldsymbol{x}^T = [\pi/4, 2]$. The result is,

$$
\left(\boldsymbol{I} - \boldsymbol{D_V}\right)\boldsymbol{D_v} = \boldsymbol{I} \quad \Rightarrow
$$

$$
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-\sqrt{2}/2 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-2 & -\pi/4 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -4 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -3 & -4 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -18 & 0 & 0 & -2.28 & 1 & 0 \\
0 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
\frac{\partial v_1}{\partial x_1} & \frac{\partial v_1}{\partial x_2} \\
\frac{\partial v_2}{\partial x_1} & \frac{\partial v_2}{\partial x_2} \\
\frac{\partial v_3}{\partial x_1} & \frac{\partial v_3}{\partial x_2} \\
\vdots & \vdots \\
\frac{\partial v_9}{\partial x_1} & \frac{\partial v_9}{\partial x_2} \\
\frac{\partial v_{10}}{\partial x_1} & \frac{\partial v_{10}}{\partial x_2} \\
\frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\
\frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2}
\end{bmatrix}
=
\begin{bmatrix}
1 & 0 \\
0 & 1 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0
\end{bmatrix}
\tag{46}
$$

where we have highlighted the total derivatives of interest.

The result in reverse mode is

$$
\left(\boldsymbol{I} - \boldsymbol{D_V}\right)^T \boldsymbol{D_v}^T = \boldsymbol{I} \quad \Rightarrow
$$

$$\begin{bmatrix}
1 & 0 & -\frac{\sqrt{2}}{2} & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & -\frac{\pi}{4} & -4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & -3 & 0 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & -4 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -18 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2.28 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
\frac{df_1}{dx_1} & \frac{df_2}{dx_1} \\
\frac{df_1}{dx_2} & \frac{df_2}{dx_2} \\
\frac{df_1}{dv_3} & \frac{df_2}{dv_3} \\
\vdots & \vdots \\
\frac{df_1}{dv_9} & \frac{df_2}{dv_9} \\
\frac{df_1}{dv_{10}} & \frac{df_2}{dv_{10}} \\
\frac{df_1}{dv_{11}} & \frac{df_2}{dv_{11}} \\
\frac{df_1}{dv_{12}} & \frac{df_2}{dv_{12}}
\end{bmatrix}
=
\begin{bmatrix}
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
1 & 0 \\
0 & 1
\end{bmatrix} \tag{47}$$

where the total derivatives of interest are highlighted, as before.

```
SUBROUTINE CALCF(x, f)
  REAL :: x(2), f(2), v(12)
  v(1) = x(1)
  v(2) = x(2)
  v(3) = SIN(v(1))
  v(4) = v(1) * v(2)
  v(5) = v(2)**2
  v(6) = 3
  v(7) = v(3) + v(4)
  v(8) = v(5) * v(6)
  v(9) = 6
  v(10) = v(8) + v(9)
  v(11) = v(7) * v(10)
  v(12) = v(4) + v(5)
  f(1) = v(11)
  f(2) = v(12)
END SUBROUTINE CALCF
```

Figure 11: Fortran source code for simple function

In the real world, you would not code this way, instead, you would use two lines of code, as shown in Figure 13.

```
SUBROUTINE CALCF_D(x, xd, f, fd)
  REAL :: x(2), f(2), v(12)
  REAL :: xd(2), fd(2), vd(12)
  vd = 0.0
  vd(1) = xd(1)
  v(1) = x(1)
  vd(2) = xd(2)
  v(2) = x(2)
  vd(3) = vd(1)*COS(v(1))
  v(3) = SIN(v(1))
  vd(4) = vd(1)*v(2) + v(1)*vd(2)
  v(4) = v(1)*v(2)
  vd(5) = 2*v(2)*vd(2)
  v(5) = v(2)**2
  vd(6) = 0.0
  v(6) = 3
  vd(7) = vd(3) + vd(4)
  v(7) = v(3) + v(4)
  vd(8) = vd(5)*v(6) + v(5)*vd(6)
  v(8) = v(5)*v(6)
  vd(9) = 0.0
  v(9) = 6
  vd(10) = vd(8) + vd(9)
  v(10) = v(8) + v(9)
  vd(11) = vd(7)*v(10) + v(7)*vd(10)
  v(11) = v(7)*v(10)
  vd(12) = vd(4) + vd(5)
  v(12) = v(4) + v(5)
  fd(1) = vd(11)
  f(1) = v(11)
  fd(2) = vd(12)
  f(2) = v(12)
END SUBROUTINE CALCF_D
```

Figure 12: Fortran source code for simple function differentiated in forward mode using source code transformation

```
SUBROUTINE CALCF2(x, f)
  REAL :: x(2), f(2)
  f(1) = (x(1)*x(2) + SIN(x(1))) * (3*x(2)**2 + 6)
  f(2) = x(1)*x(2) + x(2)**2
END SUBROUTINE CALCF2
```

Figure 13: Fortran source code for simple function

```
SUBROUTINE CALCF2_D(x, xd, f, fd)
  REAL :: x(2), f(2)
  REAL :: xd(2), fd(2)
  fd(1) = (xd(1)*x(2)+x(1)*xd(2)+xd(1)*COS(x(1)))*(3*x(2)**2+6) + (x(1)*&
&    x(2)+SIN(x(1)))*3*2*x(2)*xd(2)
  f(1) = (x(1)*x(2)+SIN(x(1)))*(3*x(2)**2+6)
  fd(2) = xd(1)*x(2) + x(1)*xd(2) + 2*x(2)*xd(2)
  f(2) = x(1)*x(2) + x(2)**2
END SUBROUTINE CALCF2_D
```

Figure 14: Fortran source code for simple function differentiated in forward mode using source code transformation

```
SUBROUTINE CALCF_B(x, xb, f, fb)
  REAL :: x(2), f(2), v(12)
  REAL :: xb(2), fb(2), vb(12)
  INTRINSIC SIN
  v(1) = x(1)
  v(2) = x(2)
  CALL PUSHREAL4(v(3))
  v(3) = SIN(v(1))
  CALL PUSHREAL4(v(4))
  v(4) = v(1)*v(2)
  CALL PUSHREAL4(v(5))
  v(5) = v(2)**2
  CALL PUSHREAL4(v(6))
  v(6) = 3
  CALL PUSHREAL4(v(7))
  v(7) = v(3) + v(4)
  CALL PUSHREAL4(v(8))
  v(8) = v(5)*v(6)
  CALL PUSHREAL4(v(9))
  v(9) = 6
  CALL PUSHREAL4(v(10))
  v(10) = v(8) + v(9)
  vb = 0.0
  vb(12) = vb(12) + fb(2)
  fb(2) = 0.0
  vb(11) = vb(11) + fb(1)
  fb(1) = 0.0
  vb(4) = vb(4) + vb(12)
  vb(5) = vb(5) + vb(12)
  vb(12) = 0.0
  vb(7) = vb(7) + v(10)*vb(11)
  vb(10) = vb(10) + v(7)*vb(11)
  vb(11) = 0.0
  CALL POPREAL4(v(10))
  vb(8) = vb(8) + vb(10)
  vb(9) = vb(9) + vb(10)
  vb(10) = 0.0
  CALL POPREAL4(v(9))
  vb(9) = 0.0
  CALL POPREAL4(v(8))
  vb(5) = vb(5) + v(6)*vb(8)
  vb(6) = vb(6) + v(5)*vb(8)
  vb(8) = 0.0
  CALL POPREAL4(v(7))
  vb(3) = vb(3) + vb(7)
  vb(4) = vb(4) + vb(7)
  vb(7) = 0.0
  CALL POPREAL4(v(6))
  vb(6) = 0.0
  CALL POPREAL4(v(5))
  vb(2) = vb(2) + 2*v(2)*vb(5)
  vb(5) = 0.0
  CALL POPREAL4(v(4))
  vb(1) = vb(1) + v(2)*vb(4)
  vb(2) = vb(2) + v(1)*vb(4)
  vb(4) = 0.0
  CALL POPREAL4(v(3))
  vb(1) = vb(1) + COS(v(1))*vb(3)
  vb(3) = 0.0
  xb = 0.0
  xb(2) = xb(2) + vb(2)
  vb(2) = 0.0
  xb(1) = xb(1) + vb(1)
END SUBROUTINE CALCF_B
```

Figure 15: Fortran source code for simple function differentiated in reverse mode using source code transformation

```
SUBROUTINE CALCF2_B(x, xb, f, fb)
  IMPLICIT NONE
  REAL :: x(2), f(2)
  REAL :: xb(2), fb(2)
  INTRINSIC SIN
  REAL :: tempb
  xb = 0.0
  xb(1) = xb(1) + x(2)*fb(2)
  xb(2) = xb(2) + (2*x(2)+x(1))*fb(2)
  fb(2) = 0.0
  tempb = (3*x(2)**2+6)*fb(1)
  xb(1) = xb(1) + (COS(x(1))+x(2))*tempb
  xb(2) = xb(2) + 3*(x(1)*x(2)+SIN(x(1)))*2*x(2)*fb(1) + x(1)*tempb
  fb(1) = 0.0
END SUBROUTINE CALCF2_B
```

Figure 16: Fortran source code for simple function differentiated in reverse mode using source code transformation