Review and Unification of Methods for Computing Derivatives of Multidisciplinary Computational Models

Joaquim R. R. A. Martins[1] and John T. Hwang[2]
University of Michigan, Ann Arbor, Michigan 48109, United States

Abstract This paper presents a review of all existing discrete methods for computing the derivatives of computational models within a unified mathematical framework. This framework hinges on a new equation—the unifying chain rule—from which all the methods can be derived. The computation of derivatives is described as a two-step process: the evaluation of the partial derivatives and the computation of the total derivatives, which are dependent on the partial derivatives. Finite differences, the complex-step method, and symbolic differentiation are discussed as options for computing the partial derivatives. It is shown that these are building blocks with which the total derivatives can be assembled using algorithmic differentiation, the direct and adjoint methods, and coupled analytic methods for multidisciplinary systems. Each of these methods is presented and applied to a common numerical example to demonstrate and compare the various approaches. The paper ends with a discussion of current challenges and possible future research directions in this field.

Contents

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
N		Number of disciplines
$\boldsymbol{e}_i$	$[0,\ldots,1,\ldots,0]^T$	i^{th} Standard basis vector
$\boldsymbol{x}$	$[x_1,\ldots,x_{n_x}]^T$	Vector of input variables
$\boldsymbol{f}$	$[f_1,\ldots,f_{n_f}]^T$	Vector of output variables
$\boldsymbol{F}$	$[F_1,\ldots,F_{n_f}]^T$	Vector of output functions
General system of equations		
$\boldsymbol{c}$	$[c_1,\ldots,c_n]^T$	Vector of constraint values
$\boldsymbol{v}$	$[v_1,\ldots,v_n]^T$	Vector of variables
$\boldsymbol{C}$	$[C_1,\ldots,C_n]^T$	Vector of constraint functions
Unary and binary operations		
$\boldsymbol{t}$	$[t_1,\ldots,t_n]^T$	Vector of variables at the line-of-code level
$\boldsymbol{T}$	$[T_1,\ldots,T_n]^T$	Vector of functions at the line-of-code level
Systems with residuals		
$\boldsymbol{r}$	$[r_1,\ldots,r_{n_y}]^T$	Vector of residual values
$\boldsymbol{y}$	$[y_1,\ldots,y_{n_y}]^T$	Vector of state variables
$\boldsymbol{R}$	$[R_1,\ldots,R_{n_y}]^T$	Vector of residual functions
Multidisciplinary systems		
$\boldsymbol{r}_i$	$[r_{1,i},\ldots,r_{n_y,i}]^T$	Vector of residual values for the i^{th} discipline
$\boldsymbol{y}_i$	$[y_{1,i},\ldots,y_{n_y,i}]^T$	Vector of state variables for the i^{th} discipline
$\boldsymbol{R}_i$	$[R_{1,i},\ldots,R_{n_y,i}]^T$	Vector of residual functions for the i^{th} discipline
$\boldsymbol{Y}_i$	$[Y_{1,i},\ldots,Y_{n_y,i}]^T$	Vector of functions for the i^{th} discipline
Derivatives		
$\dfrac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}}$	$\left[\dfrac{\mathrm{d}f_i}{\mathrm{d}x_j}\right]_{n_f\times n_x}$	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 $\boldsymbol{x}$

1 Introduction

The computation of derivatives is part of the broader field of sensitivity analysis. Sensitivity analysis is the study of how the outputs of a model change in response to changes in its inputs. It 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 [82]. 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 it is typically used in physics-based models where the uncertainties tend to be lower. In this paper, we focus on the computation of local sensitivities in the form of *first-order total derivatives*, where the model is a numerical algorithm representing a deterministic model. 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. While we focus on first-order derivatives, the techniques that we present can all be extended to compute higher-order derivatives.

In the engineering optimization literature, the term "sensitivity analysis" is often used to refer to the computation of derivatives. In addition, derivatives are sometimes referred to as "sensitivity derivatives" [8, 84, 97] or "design sensitivities" [7, 32, 95]. While these terms are not incorrect, we prefer to use the more restrictive and succinct term, i.e., *derivative.*

Derivatives play a central role in several numerical algorithms. In many cases, such as Newton-based methods, the computational effort of an algorithm depends heavily on the run time and memory requirements of the derivative computation. Examples of such algorithms include Newton–Krylov methods applied to the solution of the Euler equations [34] and coupled aerostructural equations [11, 12, 42, 44]. There are general-purpose numerical libraries that implement many of these methods [6, 33]. In these applications, it is desirable to have the Jacobian matrix of the system (the derivatives of the residuals of the equations to be solved with respect to the state variables) or at least an approximation of this Jacobian to help precondition the system.

In other applications, it is desirable to compute the derivatives of the complete system with respect to a set of parameters. Such applications include gradient-enhanced surrogate models [20], the sensitivity analysis of eigenvalue problems [83], structural topology optimization [36, 48, 49, 86], aerostructural optimization [43, 62], and the computation of aircraft stability derivatives [55, 56]. As we will see, some methods for computing these derivatives involve computing the Jacobian mentioned above. While gradient-based optimization algorithms are usually more difficult to use than gradient-free algorithms, they are currently the only way to solve optimization problems with $\mathcal{O}(10^2)$ or more design variables. This is especially true when the gradient-based optimization algorithm is used in conjunction with an efficient computation of the derivatives.

The accuracy of the derivative computation affects the convergence behavior of 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 optimal solution, and inaccurate gradients can cause the optimizer to halt or to take a less direct route to the optimum that involves more iterations.

Despite the usefulness of derivatives, and the significant progress made in derivative computation in the last two decades, only a small fraction of software packages for computational models include efficient derivative computations. Therefore, much of the research in this area has not yet been widely used in engineering practice. There are various reasons for this, including the fact that gradient-based methods require some expertise and are sensitive to noise and discontinuities in the computational models, while gradient-free methods are much easier to use. In addition, few of the experts in a given computational model have the necessary background to implement efficient derivative computations for that model. This is because derivative computation methods are not usually included in courses on computational models. This paper aims to address this by providing an overview of all the existing methods, complete with examples.

In this review, we assume that the numerical models are 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 uses 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 between the output and the unperturbed value and dividing by the value of the perturbation. Although finite differences are not usually accurate or computationally efficient, they are extremely easy to implement and therefore widely used.

In addition to the inaccuracies inherent in finite differences, computing derivatives with respect to a large number of inputs using these methods is prohibitively expensive when the computational cost of the simulations is significant. While finite differences are adequate for certain applications, other 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 discrete 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 [1, 8, 32, 72, 95] or limited in scope [88–90]. Spurred by the recent advances in this area, we decided to make a connection between derivative computation methods that are usually not explained together, leading to new insights and a broader view of the subject. We also aim to help researchers and practitioners decide which method is suitable for their particular needs.

We start by defining the nomenclature and by presenting a framework that unifies all derivative computation methods. Then we progress through the methods for differentiating functions, which represent the options available for computing partial derivatives. Finally, we review the methods for computing total derivatives in single- or multi-disciplinary systems, and we show how each is derived from the unified theory. Each of these methods is applied to a common numerical example to illustrate how the method works and how it is related to the others. The historical literature is cited and recent advances are mentioned as the theory is presented.

2 Mathematical Framework

To ensure that the methods are explained for the most general case and to show how they can be derived under a single theoretical framework, it is important to characterize the computational model that is involved and to 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, and it 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 might refer to the model or program as a *system*, since it is an interconnected series of computations.

2.1 Definitions

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. Multiple lines of code are grouped into an organizational unit, such as a subroutine, method, or class, that contains a sequence of explicit functions, denoted 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

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

Here we adopt the convention that the lower case represents the *value* of a variable, and the upper case represents the *function* that computes that 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.,

$$v_i = V_i(v_1, v_2, \ldots, v_i, \ldots, v_n). \tag{2}$$

The solution of such systems requires iterative numerical methods that can be programmed via loops where the variables are updated. Such 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 or dependencies—as written in Eq. (1)—if we *unroll all the loops* and represent all values a variable might take in successive iterations as separate variables that are never overwritten. This means that if the solution of a system of equations requires iteration, we would treat each iteration as a new set of variables that is dependent on the previous set.

When a computational model involves the solution of a system of equations, it is helpful to denote the computation as a vector of *residual equations*,

$$\boldsymbol{r} = \boldsymbol{R}(\boldsymbol{v}) = 0, \tag{3}$$

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

To relate these concepts to the usual conventions in sensitivity analysis, we now separate $\boldsymbol{v}$ into the independent variables $\boldsymbol{x}$, state variables $\boldsymbol{y}$, and quantities of interest $\boldsymbol{f}$. Using this notation, we can write the residual equations as

$$\boldsymbol{r} = \boldsymbol{R}(\boldsymbol{x}, \boldsymbol{Y}(\boldsymbol{x})) = 0 \tag{4}$$

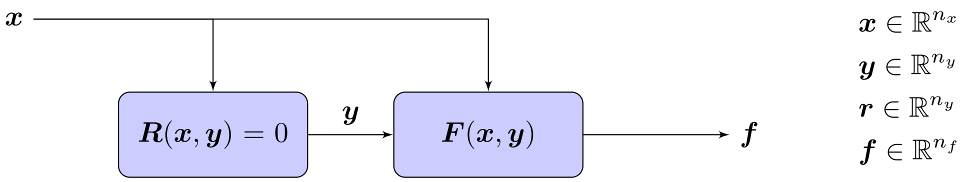

$\boldsymbol{x}$
$\boldsymbol{R}(\boldsymbol{x},\boldsymbol{y})=0$
$\boldsymbol{y}$
$\boldsymbol{F}(\boldsymbol{x},\boldsymbol{y})$
$\boldsymbol{f}$
$\boldsymbol{x}\in\mathbb{R}^{n_x}$
$\boldsymbol{y}\in\mathbb{R}^{n_y}$
$\boldsymbol{r}\in\mathbb{R}^{n_y}$
$\boldsymbol{f}\in\mathbb{R}^{n_f}$

Figure 1: Dependency of the quantities of interest on the independent variables both directly and through the residual equations that determine the state variables.

where $\boldsymbol{Y}(\boldsymbol{x})$ denotes the fact that $\boldsymbol{y}$ depends *implicitly* on $\boldsymbol{x}$ through the solution of the residual equations (4). 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 outputs) also have the same type of variable dependence in the general case, i.e.,

$$\boldsymbol{f}=\boldsymbol{F}(\boldsymbol{x},\boldsymbol{Y}(\boldsymbol{x})). \tag{5}$$

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 (4). The dependencies involved in the computation of the functions of interest are represented in Fig. 1. Our assumption is that we are ultimately interested in the total derivatives of $\boldsymbol{f}$ with respect to $\boldsymbol{x}$.

2.2 The Unifying Chain Rule

In this section, we derive and present a matrix equation from which we can obtain all the known methods for computing the total derivatives of a system. All methods can be derived from this equation through the appropriate choice of the level of decomposition of the system, since the methods share a common origin: a basic relationship between partial derivatives and total derivatives.

Consider a set of n variables, denoted $\boldsymbol{v}=[v_1,\ldots,v_n]^T$, and n functions, denoted $\boldsymbol{C}=[C_1(\boldsymbol{v}),\ldots,C_n(\boldsymbol{v})]^T$. The value of $\boldsymbol{v}$ is uniquely defined by the n constraint equations $c_i=C_i(\boldsymbol{v})=0$, for $1\leq i\leq n$. If we define $\bar{C}_i(\boldsymbol{v})$ as the linearization of $C_i(\boldsymbol{v})$ about the point $\boldsymbol{v}^0$, the multivariate Taylor series expansion of the vector-valued function $\boldsymbol{C}$ is

$$\Delta\bar{\boldsymbol{c}}=\frac{\partial\boldsymbol{C}}{\partial\boldsymbol{v}}\Delta\boldsymbol{v}, \tag{6}$$

since all higher-order derivatives are zero, and $\partial\bar{\boldsymbol{C}}/\partial\boldsymbol{v}=\partial\boldsymbol{C}/\partial\boldsymbol{v}$. The solution of the linear system (6) yields the vector of changes to $\boldsymbol{v}$ required to obtain the perturbations in the linearized constraints, $\Delta\bar{\boldsymbol{c}}$, assuming $\partial\boldsymbol{C}/\partial\boldsymbol{v}$ is invertible.

Since this equation uniquely defines $\Delta\boldsymbol{v}$ for any left-hand side, we choose the vectors of the standard basis for $\mathbb{R}^n$ with the j^{th} vector multiplied by $\Delta\bar{c}^{(j)}$, i.e.,

$$\left[\boldsymbol{e}_1\Delta\bar{c}^{(1)}\Big|\cdots\Big|\boldsymbol{e}_j\Delta\bar{c}^{(j)}\Big|\cdots\Big|\boldsymbol{e}_n\Delta\bar{c}^{(n)}\right]=\frac{\partial\boldsymbol{C}}{\partial\boldsymbol{v}}\left[\Delta\boldsymbol{v}^{(1)}\Big|\cdots\Big|\Delta\boldsymbol{v}^{(j)}\Big|\cdots\Big|\Delta\boldsymbol{v}^{(n)}\right], \tag{7}$$

where each vector $\Delta\boldsymbol{v}^{(j)}$ represents the changes to $\boldsymbol{v}$ that produce the variation $[0,\ldots,0,\Delta\bar{c}^{(j)},0,\ldots,0]^T$. We now move the scalars $\Delta\bar{c}^{(j)}$ in Eq. (7) to the right-hand side to obtain

$$[\boldsymbol{e}_1|\cdots|\boldsymbol{e}_j|\cdots|\boldsymbol{e}_n]=\frac{\partial\boldsymbol{C}}{\partial\boldsymbol{v}}\left[\frac{\Delta\boldsymbol{v}^{(1)}}{\Delta\bar{c}^{(1)}}\Bigg|\cdots\Bigg|\frac{\Delta\boldsymbol{v}^{(j)}}{\Delta\bar{c}^{(j)}}\Bigg|\cdots\Bigg|\frac{\Delta\boldsymbol{v}^{(n)}}{\Delta\bar{c}^{(n)}}\right], \tag{8}$$

where we now have an identity matrix on the left-hand side.

Alternatively, $\Delta\boldsymbol{v}^{(j)}$ can be interpreted as the direction in $\mathbb{R}^n$ along which only the j^{th} constraint $C_j(\boldsymbol{v})$ changes, while all other constraints remain unchanged to the first order. Therefore, $\Delta\boldsymbol{v}^{(i)}/\Delta\bar{c}^{(j)}$ is the j^{th} column of the $\mathrm{d}\boldsymbol{v}/\mathrm{d}\boldsymbol{c}$ matrix, since it represents the vector of variations in $\boldsymbol{v}$ that comes about through an implicit dependence on $\bar{\boldsymbol{c}}$, which is perturbed only in the j^{th} entry.

Thus, the $\partial \boldsymbol{C}/\partial \boldsymbol{v}$ and $\mathrm{d}\boldsymbol{v}/\mathrm{d}\boldsymbol{c}$ matrices are inverses of each other and they commute, so we can switch the order and take the transpose to get an alternative form. Therefore, we can write

$$\boxed{\frac{\partial \boldsymbol{C}}{\partial \boldsymbol{v}}\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}\boldsymbol{c}} = \boldsymbol{I} = \frac{\partial \boldsymbol{C}}{\partial \boldsymbol{v}}^T \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}\boldsymbol{c}}^T.} \tag{9}$$

We call the left-hand side the *forward chain rule* and the right-hand side the *reverse chain rule*. As we will see throughout the remainder of this paper: *All methods for derivative computation can be derived from one of the forms of the chain rule* (9) *by changing what we mean by "variables" and "constraints,"* which can be seen as a level of decomposition. We will refer to this equation as the *unifying chain rule*.

The derivatives of interest, $\mathrm{d}\boldsymbol{f}/\mathrm{d}\boldsymbol{x}$, are typically the derivatives of some of the last variables in the sequence $(v_1, \ldots, v_n)$ with respect to some of the first variables in the same sequence. The variables and constraints are not necessarily in sequence, but we denote them as such for convenience without loss of generality.

Now we recall that each variable v_i is associated with a constraint $C_i(v_i)$. In the case of input variables, which are independent variables, this constraint is simply $c_i = x_i - x_i^0 = 0$, where x_i^0 is the chosen value for the given input. Thus, x_i^0 is the set of values at which we are evaluating the computational model and its derivatives.

Defining the constraints on the input variables in this way, we can write

$$\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}} = \begin{bmatrix} \frac{\mathrm{d}f_1}{\mathrm{d}x_1} & \cdots & \frac{\mathrm{d}f_1}{\mathrm{d}x_{n_x}} \\ \vdots & \ddots & \vdots \\ \frac{\mathrm{d}f_{n_f}}{\mathrm{d}x_1} & \cdots & \frac{\mathrm{d}f_{n_f}}{\mathrm{d}x_{n_x}} \end{bmatrix} = \begin{bmatrix} \frac{\mathrm{d}v_{(n-n_f+1)}}{\mathrm{d}c_1} & \cdots & \frac{\mathrm{d}v_{(n-n_f+1)}}{\mathrm{d}c_{n_x}} \\ \vdots & \ddots & \vdots \\ \frac{\mathrm{d}v_n}{\mathrm{d}c_1} & \cdots & \frac{\mathrm{d}v_n}{\mathrm{d}c_{n_x}} \end{bmatrix}. \tag{10}$$

This is an $n_f \times n_x$ matrix that corresponds to the lower-left block of $\mathrm{d}\boldsymbol{v}/\mathrm{d}\boldsymbol{c}$, or the corresponding transposed upper-right block of $\mathrm{d}\boldsymbol{v}/\mathrm{d}\boldsymbol{c}^T$.

The two sections that follow show how to use the unifying chain rule (9) to compute total derivatives in practice. Section 3 reviews the methods for computing the partial derivatives in the $\partial \boldsymbol{C}/\partial \boldsymbol{v}$ matrix. Section 4 presents the methods for computing total derivatives—monolithic differentiation, algorithmic differentiation, analytic methods, and coupled analytic methods—that correspond to different choices for $\boldsymbol{v}$ and $\boldsymbol{C}$, choices that reflect the level of decomposition of the system.

3 Methods for Computing Partial Derivatives

For a given choice of $\boldsymbol{v}$ in the unifying chain rule (9), the $\partial \boldsymbol{C}/\partial \boldsymbol{v}$ matrix must be assembled by computing partial derivatives in the appropriate context. Any variable that is used in computing C_i but is not one of the entries of $\boldsymbol{v}$ is treated as part of the black-box function C_i. These variables are not held fixed while computing a partial derivative of C_i. Thus, the meaning of a partial derivative changes based on the variables, $\boldsymbol{v}$, that are exposed in the current context.

Consider a vector-valued function $\boldsymbol{F}$ with respect to a vector of independent variables $\boldsymbol{x}$. We are interested in the Jacobian,

$$\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial F_1}{\partial x_1} & \cdots & \frac{\partial F_1}{\partial x_{n_x}} \\ \vdots & \ddots & \vdots \\ \frac{\partial F_{n_f}}{\partial x_1} & \cdots & \frac{\partial F_{n_f}}{\partial x_{n_x}} \end{bmatrix}, \tag{11}$$

which is an $n_f \times n_x$ matrix. In this section, we examine three methods for computing the entries of this matrix.

3.1 Finite Differences

Finite-difference formulas are derived by 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

$$\frac{\partial \boldsymbol{F}}{\partial x_j} = \frac{\boldsymbol{F}(\boldsymbol{x} + \boldsymbol{e}_j h) - \boldsymbol{F}(\boldsymbol{x})}{h} + \mathcal{O}(h), \tag{12}$$

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 computational model. The application of this formula requires the evaluation of the model at the reference point $\boldsymbol{x}$ and one perturbed point $\boldsymbol{x} + \boldsymbol{e}_j h$, and yields one column of the Jacobian (11). Each additional column requires an additional evaluation of the computational model. Hence, the cost of computing the complete Jacobian is proportional to the number of input variables of interest, n_x. When the computational model is nonlinear, the constant of proportionality with respect to the number of variables can be less than one if the solutions for the successive steps are warm-started with the previous solution.

Finite-difference methods are widely used to compute derivatives because of their simplicity and the fact that they can be implemented even when a given computational model is a black box. Most gradient-based optimization algorithms perform finite differences by default when the user does not provide the required gradients [28, 71].

When it comes to accuracy, we can see from the forward-difference formula (12) 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. With 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 [61].

Despite these inherent problems, finite differences can sometimes be advantageous when the function of interest is noisy. This numerical noise could be, for example, due to poor convergence of the computational model or re-meshing of the computational domain. In such cases, using a large enough step can effectively smooth the derivative and capture the correct trend in the function of interest [95].

Because of 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 functions.

3.2 Complex Step

The complex-step derivative approximation computes derivatives of real functions using complex variables. This method originated with the work of Lyness [52] and Lyness and Moler [53]. 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 [91], who derived a simple formula for estimating the first derivative.

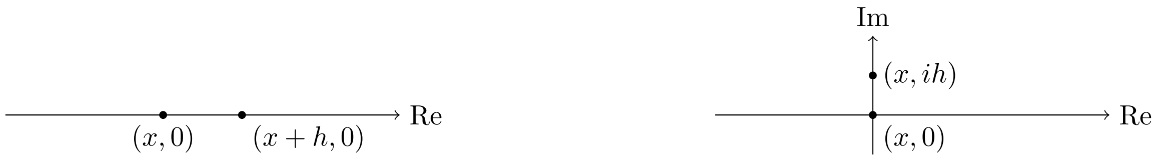
The complex-step derivative approximation, like finite-difference formulas, can also be derived using a Taylor series expansion. Rather than using a real step h, we now use a pure imaginary step, ih. If $\boldsymbol{f}$ is a real function in real variables and it is also analytic, we can expand it in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

$$\boldsymbol{F}(\boldsymbol{x} + ih\boldsymbol{e}_j) = \boldsymbol{F}(\boldsymbol{x}) + ih\frac{\partial \boldsymbol{F}}{\partial x_j} - \frac{h^2}{2}\frac{\partial^2 \boldsymbol{F}}{\partial x_j^2} - \frac{ih^3}{6}\frac{\partial^3 \boldsymbol{F}}{\partial x_j^3} + \ldots \tag{13}$$

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

$$\frac{\partial \boldsymbol{F}}{\partial x_j} = \frac{\operatorname{Im}\left[\boldsymbol{F}(\boldsymbol{x} + ih\boldsymbol{e}_j)\right]}{h} + \mathcal{O}(h^2). \tag{14}$$

Hence, the approximation is an $\mathcal{O}(h^2)$ estimate of the derivative. Like a finite-difference formula, each additional evaluation results in a column of the Jacobian (11), and the cost of computing the required derivatives is proportional to the number of design variables, n_x.



$$\frac{\partial F}{\partial x} \approx \frac{F(x+h)-F(x)}{h} \qquad \frac{\partial F}{\partial x} \approx \frac{\mathrm{Im}[F(x+ih)]-\mathrm{Im}[F(x)]}{\mathrm{Im}[ih]} = \frac{\mathrm{Im}[F(x+ih)]}{h}$$

Figure 2: Computation of $\partial F/\partial x$ using a forward step in the real axis via a forward difference (left) and via a step in the complex axis (right). Here, F and x are scalar-valued.

Because there is no subtraction operation in the complex-step derivative approximation (14), the only source of numerical error is the truncation error, which is $\mathcal{O}(h^2)$. By decreasing h to a small enough value, we can ensure that the truncation error is 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. [4], who used it to compute the derivatives of a Navier–Stokes solver and later multidisciplinary systems [68]. Martins et al. [61] showed that the complex-step method is generally applicable to any algorithm and described a detailed procedure for its implementation. They also presented an alternative way of deriving and understanding the complex step and connected it to algorithmic differentiation.

The complex-step method requires access to the source code of the given computational model, and thus it cannot be applied to black-box models without additional effort in most cases. To implement the complex-step method, the source code 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. [61] provide a script that facilitates the implementation of the complex-step method in 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. In 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.

The complex-step approach is now widely used, with applications ranging from the verification of high-fidelity aerostructural derivatives [44, 63] to the development of immunology models [51]. In one case, it was implemented in an aircraft design optimization framework that involves multiple languages (Python, C, C++, and Fortran) [**?**], demonstrating the flexibility of this approach. Other applications include uncertainty propagation [16], Hessian matrices in statistics [79], derivatives of matrix functions [2], Jacobian matrices in liquid chromatography [25], first and second derivatives in Kalman filters [45], local sensitivities in biotechnology [21], and climate modeling [80]. The complex-step approach has also been extended to enable the computation of higher-order derivatives using multicomplex variables [47] and hyper-dual numbers [24].

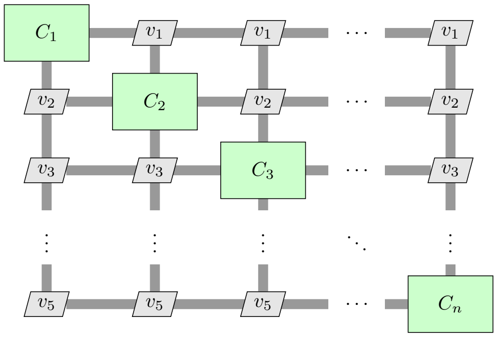
3.3 Symbolic Differentiation

Symbolic differentiation is possible only for explicit functions and can either be done by hand or by appropriate software, such as Maple, Mathematica, or Matlab. For a sequence of composite functions, it is possible to use the chain rule, but for general algorithms it may be impossible to produce a closed-form expression for the derivatives.

Nevertheless, symbolic differentiation is useful in the context of computational models, since it often can be applied to simple functions within the model, reducing the computational cost of obtaining certain partial derivatives that are needed in other methods.

4 Methods for Computing Total Derivatives in a System

Variables and Constraints



$$\boldsymbol{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

$$\boldsymbol{C}(\boldsymbol{v}) = \begin{bmatrix} C_1(v_1, \ldots, v_n) \\ C_2(v_1, \ldots, v_n) \\ \vdots \\ C_n(v_1, \ldots, v_n) \end{bmatrix}$$

Derivation

$$\left[\frac{\partial \boldsymbol{C}}{\partial \boldsymbol{v}}\right] \left[\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}\boldsymbol{c}}\right] = \boldsymbol{I} = \left[\frac{\partial \boldsymbol{C}}{\partial \boldsymbol{v}}\right]^T \left[\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}\boldsymbol{c}}\right]^T$$

$$\begin{bmatrix} \frac{\partial C_1}{\partial v_1} & \cdots & \frac{\partial C_1}{\partial v_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial C_n}{\partial v_1} & \cdots & \frac{\partial C_n}{\partial v_n} \end{bmatrix} \begin{bmatrix} \boxed{\begin{matrix} \frac{\mathrm{d}v_1}{\mathrm{d}c_1} & \cdots & \frac{\mathrm{d}v_1}{\mathrm{d}c_n} \\ \vdots & \ddots & \vdots \\ \frac{\mathrm{d}v_n}{\mathrm{d}c_1} & \cdots & \frac{\mathrm{d}v_n}{\mathrm{d}c_n} \end{matrix}} \end{bmatrix} = \boldsymbol{I} = \begin{bmatrix} \frac{\partial C_1}{\partial v_1} & \cdots & \frac{\partial C_n}{\partial v_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial C_1}{\partial v_n} & \cdots & \frac{\partial C_n}{\partial v_n} \end{bmatrix} \begin{bmatrix} \boxed{\begin{matrix} \frac{\mathrm{d}v_1}{\mathrm{d}c_1} & \cdots & \frac{\mathrm{d}v_n}{\mathrm{d}c_1} \\ \vdots & \ddots & \vdots \\ \frac{\mathrm{d}v_1}{\mathrm{d}c_n} & \cdots & \frac{\mathrm{d}v_n}{\mathrm{d}c_n} \end{matrix}} \end{bmatrix}$$

Forward form

$$\sum_{k=1}^{n} \frac{\partial C_i}{\partial v_k} \frac{\mathrm{d}v_k}{\mathrm{d}c_j} = \delta_{ij}$$

Reverse form

$$\sum_{k=1}^{n} \frac{\mathrm{d}v_i}{\mathrm{d}c_k} \frac{\partial C_k}{\partial v_j} = \delta_{ij}$$

Figure 3: Equations for computing total derivatives in a general system of equations.

As mentioned before, we are going to derive all the methods from the unifying chain rule (9). Each method corresponds to a specific choice of the variables $\boldsymbol{v}$ and the constraints $\boldsymbol{C}$. To facilitate the unification of the theory, we present these choices and the derivation from the chain rule in figures with a common layout. Figure 3 shows the layout for the general case, with no specific choice of variables or constraints.

In the top box we show the definition of the variables and constraints on the right, and a diagram showing the dependence of the constraint functions on the variables on the left. For this diagram, we use the extended design structure matrix (XDSM) standard developed by Lambe and Martins [46], which enables the representation of both the data dependency and the procedure for a given algorithm. The diagonal entries are the functions in the process, and the off-diagonal entries represent the data. For the purposes of this work, we only need the data dependency information, which is expressed by the thick gray lines. The XDSM diagram in Figure 3 shows the constraints or vectors of constraints along the diagonal, and variables or vectors of variables in the off-diagonal positions. The off-diagonal entry in row i and column j expresses the dependence of the j^{th} constraint on the i^{th} variable.

The middle box is used for the derivation of the method from the unifying chain rule (9). In this general case, we have no derivation and just expand the matrices in the chain rule. The two boxes at the bottom show the forward and reverse forms of the method.

In the sections that follow, we present one such table for each of the methods, where the variables and constraints are specified differently depending on the method, and the application of the unifying chain rule to those variables and constraints yields the method.

```fortran
FUNCTION F(x)
  REAL :: x(2), det, y(2), f(2)
  det = 2 + x(1)*x(2)**2
  y(1) = x(2)**2*SIN(x(1))/det
  y(2) = SIN(x(1))/det
  f(1) = y(1)
  f(2) = y(2)*SIN(x(1))
  RETURN
END FUNCTION F
```

Figure 4: Fortran code for the computational model of the numerical example.

Numerical Example

Before discussing the methods, we introduce a simple numerical example that we use to demonstrate each method. The example can be interpreted as an explicit function, a model with states constrained by residuals, or a multidisciplinary system, depending on what is needed.

This example has two inputs, $\boldsymbol{x} = [x_1, x_2]^T$, and the residual equations are

$$\boldsymbol{R} = \begin{bmatrix} R_1(x_1, x_2, y_1, y_2) \\ R_2(x_1, x_2, y_1, y_2) \end{bmatrix} = \begin{bmatrix} x_1 y_1 + 2y_2 - \sin x_1 \\ -y_1 + x_2^2 y_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{15}$$

where $\boldsymbol{y} = [y_1 \quad y_2]^T$ are the state variables. The output functions are

$$\boldsymbol{F} = \begin{bmatrix} F_1(x_1, x_2, y_1, y_2) \\ F_2(x_1, x_2, y_1, y_2) \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \sin x_1 \end{bmatrix}. \tag{16}$$

Since the residuals (15) are linear in the state variables in this problem, we solve the following linear system:

$$\begin{bmatrix} x_1 & 2 \\ -1 & x_2^2 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \sin x_1 \\ 0 \end{bmatrix}. \tag{17}$$

In Fig. 4 we list a Fortran program that takes the two input variables, solves the above linear system, and computes the two output variables. In this case, the algorithm solves the system directly and there are no loops. The v's introduced above correspond to each variable assignment, i.e.,

$$\boldsymbol{v} = [\texttt{x(1)} \quad \texttt{x(2)} \quad \texttt{det} \quad \texttt{y(1)} \quad \texttt{y(2)} \quad \texttt{f(1)} \quad \texttt{f(2)}]^T. \tag{18}$$

The objective is to compute the derivatives at $\boldsymbol{x} = [1, 1]$ of both outputs with respect to both inputs, i.e., the Jacobian,

$$\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}} = \begin{bmatrix} \frac{\mathrm{d}f_1}{\mathrm{d}x_1} & \frac{\mathrm{d}f_1}{\mathrm{d}x_2} \\ \frac{\mathrm{d}f_2}{\mathrm{d}x_1} & \frac{\mathrm{d}f_2}{\mathrm{d}x_2} \end{bmatrix}. \tag{19}$$

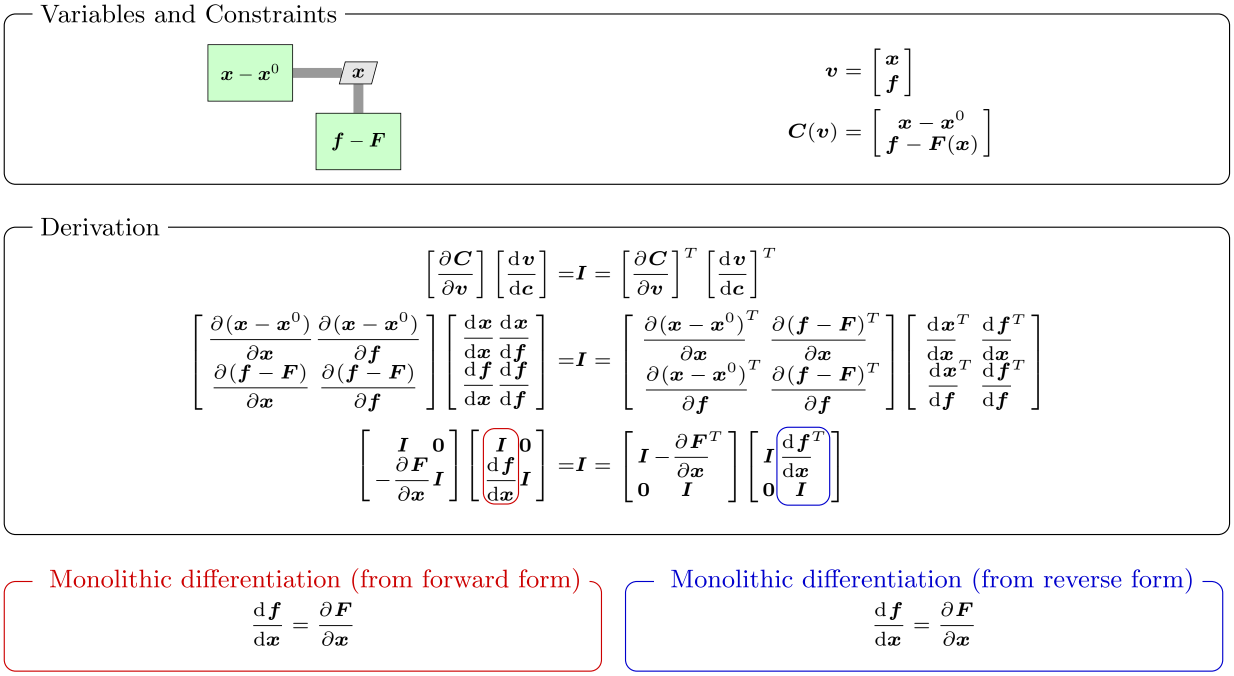
4.1 Monolithic Differentiation

In monolithic differentiation, the entire computational model is treated as a “black box.” This may be the only option in cases for which the source code is not available, or if it is not deemed worthwhile to implement more sophisticated approaches for computing the derivatives.

In monolithic differentiation, the only variables that are tracked are the inputs $\boldsymbol{x}$ and the outputs $\boldsymbol{f}$. Thus, the variables are defined as $\boldsymbol{v} = \left[\boldsymbol{x}^T, \boldsymbol{f}^T\right]^T$, as shown in Fig. 5. The constraints are just the residuals of the inputs and outputs, i.e., the differences between the actual values of the variables and the corresponding functions. Thus, the input variables $\boldsymbol{x}$ are simply forced to match the specified values $\boldsymbol{x}^0$, and the output variables $\boldsymbol{f}$ are forced to match the results of the computational model, $\boldsymbol{F}$.

The result of replacing these definitions of the variables and constraints is rather simple; we just obtain

$$\frac{\mathrm{d}f_i}{\mathrm{d}x_j} = \frac{\partial F_i}{\partial x_j} \tag{20}$$


Variables and Constraints
$\boldsymbol{x}-\boldsymbol{x}^0$
$\boldsymbol{x}$
$\boldsymbol{f}-\boldsymbol{F}$
Derivation
Monolithic differentiation (from forward form)
Monolithic differentiation (from reverse form)

$$\boldsymbol{v}=\begin{bmatrix}\boldsymbol{x}\\ \boldsymbol{f}\end{bmatrix}$$

$$\boldsymbol{C}(\boldsymbol{v})=\begin{bmatrix}\boldsymbol{x}-\boldsymbol{x}^0\\ \boldsymbol{f}-\boldsymbol{F}(\boldsymbol{x})\end{bmatrix}$$

$$\left[\frac{\partial\boldsymbol{C}}{\partial\boldsymbol{v}}\right]\left[\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}\boldsymbol{c}}\right]=\boldsymbol{I}=\left[\frac{\partial\boldsymbol{C}}{\partial\boldsymbol{v}}\right]^T\left[\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}\boldsymbol{c}}\right]^T$$

$$\begin{bmatrix}\frac{\partial(\boldsymbol{x}-\boldsymbol{x}^0)}{\partial\boldsymbol{x}} & \frac{\partial(\boldsymbol{x}-\boldsymbol{x}^0)}{\partial\boldsymbol{f}}\\ \frac{\partial(\boldsymbol{f}-\boldsymbol{F})}{\partial\boldsymbol{x}} & \frac{\partial(\boldsymbol{f}-\boldsymbol{F})}{\partial\boldsymbol{f}}\end{bmatrix}\begin{bmatrix}\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}\boldsymbol{x}} & \frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}\boldsymbol{f}}\\ \frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}} & \frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{f}}\end{bmatrix}=\boldsymbol{I}=\begin{bmatrix}\frac{\partial(\boldsymbol{x}-\boldsymbol{x}^0)}{\partial\boldsymbol{x}}^T & \frac{\partial(\boldsymbol{f}-\boldsymbol{F})}{\partial\boldsymbol{x}}^T\\ \frac{\partial(\boldsymbol{x}-\boldsymbol{x}^0)}{\partial\boldsymbol{f}}^T & \frac{\partial(\boldsymbol{f}-\boldsymbol{F})}{\partial\boldsymbol{f}}^T\end{bmatrix}\begin{bmatrix}\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}\boldsymbol{x}}^T & \frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}}^T\\ \frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}\boldsymbol{f}}^T & \frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{f}}^T\end{bmatrix}$$

$$\begin{bmatrix}\boldsymbol{I} & \boldsymbol{0}\\ -\frac{\partial\boldsymbol{F}}{\partial\boldsymbol{x}} & \boldsymbol{I}\end{bmatrix}\begin{bmatrix}\boldsymbol{I} & \boldsymbol{0}\\ \boxed{\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}}} & \boldsymbol{I}\end{bmatrix}=\boldsymbol{I}=\begin{bmatrix}\boldsymbol{I} & -\frac{\partial\boldsymbol{F}}{\partial\boldsymbol{x}}^T\\ \boldsymbol{0} & \boldsymbol{I}\end{bmatrix}\begin{bmatrix}\boldsymbol{I} & \boxed{\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}}^T}\\ \boldsymbol{0} & \boldsymbol{I}\end{bmatrix}$$

$$\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}}=\frac{\partial\boldsymbol{F}}{\partial\boldsymbol{x}}$$

$$\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}}=\frac{\partial\boldsymbol{F}}{\partial\boldsymbol{x}}$$

Figure 5: Monolithic (black box) differentiation.

for each input x_j and output variable f_i. This relationship is simply stating that the total derivatives of interest are the partial derivatives of the model when considered in this context. The derivation shown in Fig. 5 is a rather roundabout way of obtaining such a trivial result, but we felt it was important to unify all the methods from the same equation.

Numerical Example

The monolithic approach treats the entire piece of code as a black box whose internal variables and computations are unknown. Thus, the tracked variables at this level of decomposition are

$$v_1=x_1,\quad v_2=x_2,\quad v_3=f_1,\quad v_4=f_2. \tag{21}$$

Inserting this choice of v_i into Eq. (9), we find that both the forward and reverse chain-rule equations yield

$$\frac{\mathrm{d}f_1}{\mathrm{d}x_1}=\frac{\partial f_1}{\partial x_1},\quad \frac{\mathrm{d}f_1}{\mathrm{d}x_2}=\frac{\partial f_1}{\partial x_2},\dots$$

For this method, computing df_1/dx_1 simply amounts to computing $\partial f_1/\partial x_1$, which we can do with the forward-difference formula (with step size $h=10^{-5}$), yielding

$$\frac{\partial f_1}{\partial x_1}(1,1)\approx\frac{f_1(x_1+h,x_2)-f_1(x_1,x_2)}{h}=0.0866023014079,$$

or the complex-step method (with step size $h=10^{-15}$). At $[x_1,x_2]=[1,1]$, this yields

$$\frac{\partial f_1}{\partial x_1}(1,1)\approx\frac{\mathrm{Im}\left[f_1(x_1+ih,x_2)\right]}{h}=0.0866039925329.$$

The digits that agree with the exact derivative are shown in blue and those that are incorrect are in red. Figure 6 confirms the expected orders of convergence for these two formulas and illustrates how subtractive cancellation error severely limits the accuracy of finite-difference methods.

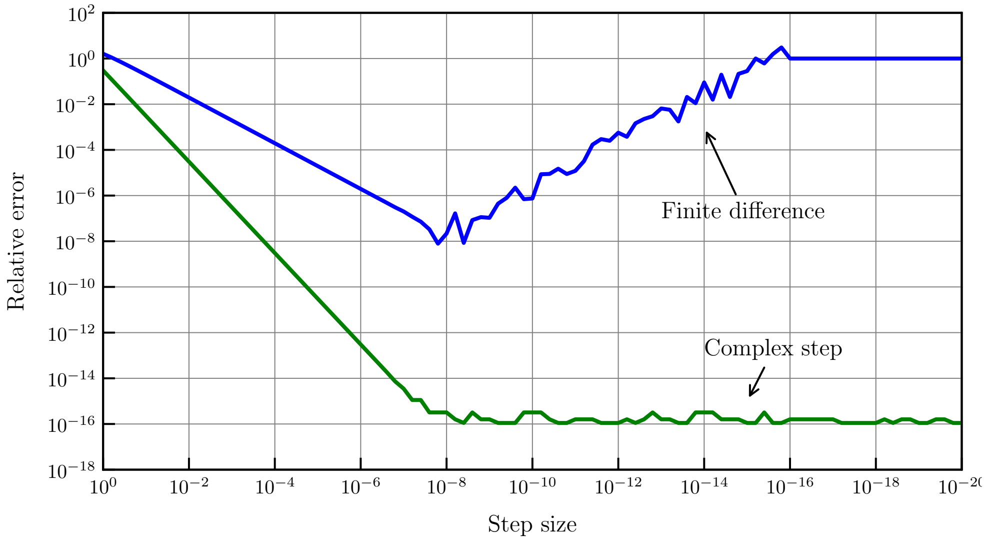


Figure 6: Relative error in the sensitivity estimate vs. step size. Because of subtractive cancellation in finite differencing, the error worsens with decreasing step size once it is below a critical value; eventually the subtraction operation yields zero, resulting in a relative error of 1. The first-order forward difference formula is used, which is consistent with the observed slope of 1, while the complex-step formula has second-order convergence, which agrees with the slope of 2 in the plot.

4.2 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 [29, 66]. 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 unifying chain rule (9) introduced in the previous section. We then explain how the method is implemented in practice and show an example.

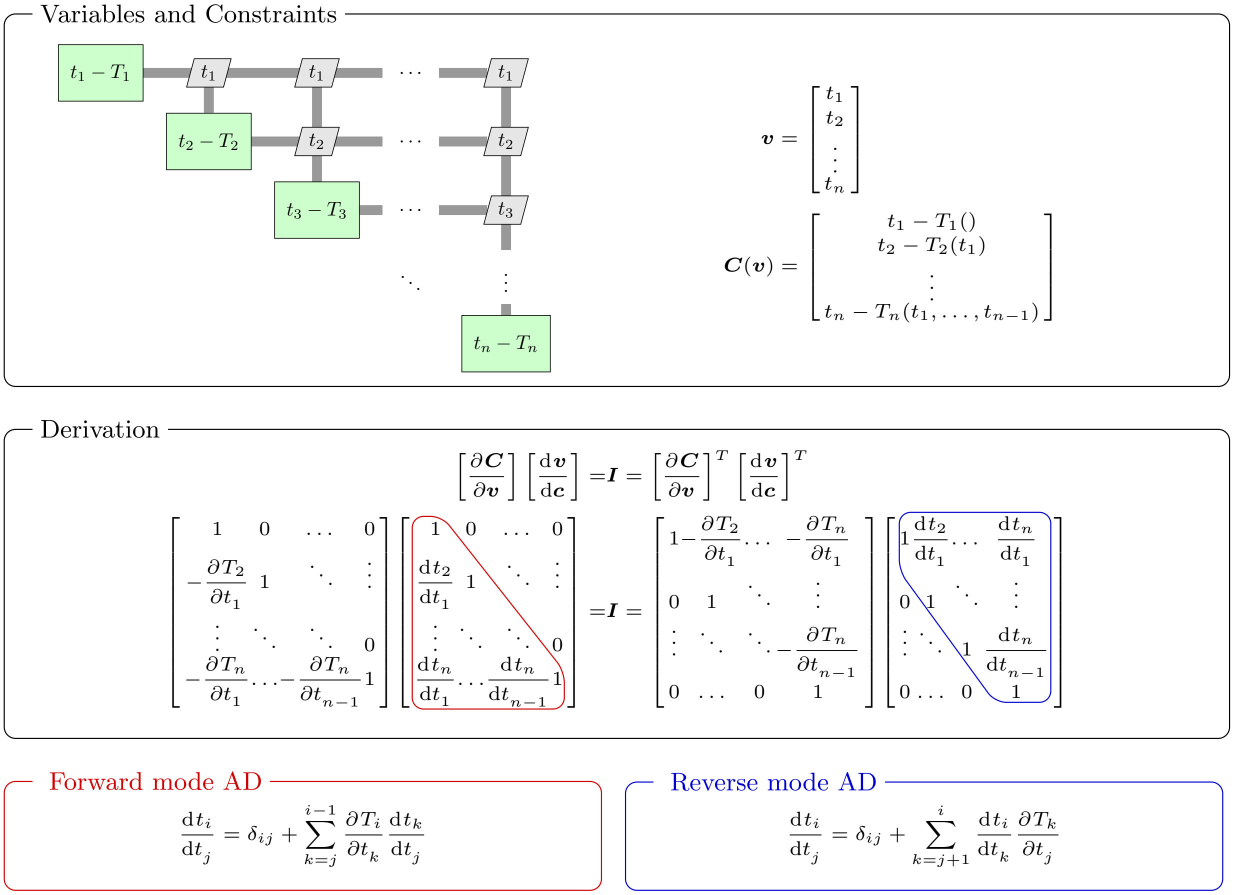
From the AD perspective, the variables $\boldsymbol{v}$ in the chain rule (9) are all of the variables assigned in the computer program, denoted $\boldsymbol{t}$, and AD applies the chain rule for every single line in the program. The computer program can thus be considered a sequence of explicit functions T_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 (1).

As mentioned in Section 2.1, for this assumption to hold, we assume that all of the loops in the program are *unrolled*. Therefore, no variables are overwritten, and each variable depends only on earlier variables in the sequence. This assumption is not restrictive, since 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 $\boldsymbol{t}$. To make clear the connection to the other derivative computation methods, we group these variables as follows:

$$\boldsymbol{v} = [\underbrace{t_1, \ldots, t_{n_x}}_{\boldsymbol{x}}, \ldots, t_j, \ldots, t_i, \ldots, \underbrace{t_{(n-n_f)}, \ldots, t_n}_{\boldsymbol{f}}]^T. \tag{22}$$

Figure 7 shows this definition and the resulting derivation. Note that the XDSM diagram shows that all variables are above the diagonal, indicating that there is only forward dependence, because of the unrolling of all loops. The constraints just enforce that the variables must be equal to the corresponding function values. Using these definitions in the unifying chain rule, we obtain a matrix equation, where the matrix that contains the unknowns (the total derivatives that we want to compute) is either lower triangular or upper triangular. The lower triangular system corresponds to the forward mode and can be solved using



$$\boldsymbol{v} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_n \end{bmatrix} \qquad \boldsymbol{C}(\boldsymbol{v}) = \begin{bmatrix} t_1 - T_1() \\ t_2 - T_2(t_1) \\ \vdots \\ t_n - T_n(t_1, \dots, t_{n-1}) \end{bmatrix}$$

$$\left[\frac{\partial \boldsymbol{C}}{\partial \boldsymbol{v}}\right]\left[\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}\boldsymbol{c}}\right] = \boldsymbol{I} = \left[\frac{\partial \boldsymbol{C}}{\partial \boldsymbol{v}}\right]^T\left[\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}\boldsymbol{c}}\right]^T$$

$$\begin{bmatrix} 1 & 0 & \dots & 0 \\ -\frac{\partial T_2}{\partial t_1} & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ -\frac{\partial T_n}{\partial t_1} & \dots & -\frac{\partial T_n}{\partial t_{n-1}} & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & \dots & 0 \\ \frac{\mathrm{d}t_2}{\mathrm{d}t_1} & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ \frac{\mathrm{d}t_n}{\mathrm{d}t_1} & \dots & \frac{\mathrm{d}t_n}{\mathrm{d}t_{n-1}} & 1 \end{bmatrix} = \boldsymbol{I} = \begin{bmatrix} 1 & -\frac{\partial T_2}{\partial t_1} & \dots & -\frac{\partial T_n}{\partial t_1} \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & -\frac{\partial T_n}{\partial t_{n-1}} \\ 0 & \dots & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & \frac{\mathrm{d}t_2}{\mathrm{d}t_1} & \dots & \frac{\mathrm{d}t_n}{\mathrm{d}t_1} \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & 1 & \frac{\mathrm{d}t_n}{\mathrm{d}t_{n-1}} \\ 0 & \dots & 0 & 1 \end{bmatrix}$$

$$\frac{\mathrm{d}t_i}{\mathrm{d}t_j} = \delta_{ij} + \sum_{k=j}^{i-1} \frac{\partial T_i}{\partial t_k}\frac{\mathrm{d}t_k}{\mathrm{d}t_j}$$

$$\frac{\mathrm{d}t_i}{\mathrm{d}t_j} = \delta_{ij} + \sum_{k=j+1}^{i} \frac{\mathrm{d}t_i}{\mathrm{d}t_k}\frac{\partial T_k}{\partial t_j}$$

Figure 7: Derivation of algorithmic differentiation.

forward substitution, while the upper triangular system corresponds to the reverse mode of AD and can be solved using back substitution.

These matrix equations can be rewritten as shown at the bottom of Fig. 7. The equation on the left represents forward-mode AD. In this case, we choose one t_j and keep j fixed. Then we work our way forward in the index $i = 1, 2, \dots, n$ until we get the desired total derivative. In the process, we obtain a whole column of the lower triangular matrix, i.e., the derivatives of all the variables with respect to the chosen variable.

Using the reverse mode, shown on the bottom right of Fig. 7, we choose a t_i (the quantity we want to differentiate) and work our way backward in the index $j = n, n-1, \dots, 1$ all of the way to the independent variables. This corresponds to obtaining a column of the upper triangular matrix, i.e., the derivatives of the chosen quantity with respect to all other variables.

Given these properties, the forward mode is more efficient for problems where there are more outputs of interest than inputs, while the opposite is true for the reverse mode.

Although AD is more accurate than finite differences, it can require more computational time. This is generally the case when using forward-mode AD. The computational cost of both methods scales linearly with the number of inputs. As previously mentioned, when finite differences are used to compute the derivatives of a nonlinear model, the perturbed solution can be warm-started using the previous solution, so the constant of proportionality can be less than one. However, the standard forward-mode AD always has a constant of proportionality approximately equal to one.

Numerical Example

We now illustrate the application of algorithmic differentiation to the numerical example introduced in Section 4. If we use the program listed in Fig. 4, seven variables are required to relate the input variables to the output variables through the lines of code in this particular implementation. These variables are

$$\boldsymbol{v} = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \\ t_7 \end{bmatrix} = \begin{bmatrix} \texttt{x(1)} \\ \texttt{x(2)} \\ \texttt{det} \\ \texttt{y(1)} \\ \texttt{y(2)} \\ \texttt{f(1)} \\ \texttt{f(2)} \end{bmatrix}. \tag{23}$$

As shown in Fig. 7, AD defines constraints of the form $C_i(\boldsymbol{v}) = t_i - T_i(t_1, \ldots, t_{i-1})$. For this problem, the functions T_i are

$$\boldsymbol{T}(\boldsymbol{t}) = \begin{bmatrix} T_1() \\ T_2() \\ T_3(t_1, t_2) \\ T_4(t_1, t_2, t_3) \\ T_5(t_1, t_3) \\ T_6(t_4) \\ T_7(t_1, t_5) \end{bmatrix} = \begin{bmatrix} x(1) \\ x(2) \\ 2 + t_1 t_2^2 \\ t_2^2 \sin(t_1)/t_3 \\ \sin(t_1)/t_3 \\ t_4 \\ t_5 \sin(t_1) \end{bmatrix}. \tag{24}$$

Applying AD to the function in Fig. 4 is mathematically equivalent to solving a linear system that arises from the unifying chain rule with this choice of constraints and variables. In this case, we are interested in a 2×2 submatrix of $\mathrm{d}\boldsymbol{t}/\,\mathrm{d}\boldsymbol{t}$, specifically,

$$\begin{bmatrix} \frac{\mathrm{d}t_6}{\mathrm{d}t_1} & \frac{\mathrm{d}t_6}{\mathrm{d}t_2} \\ \frac{\mathrm{d}t_7}{\mathrm{d}t_1} & \frac{\mathrm{d}t_7}{\mathrm{d}t_2} \end{bmatrix} = \begin{bmatrix} \frac{\mathrm{d}f_1}{\mathrm{d}x_1} & \frac{\mathrm{d}f_1}{\mathrm{d}x_2} \\ \frac{\mathrm{d}f_2}{\mathrm{d}x_1} & \frac{\mathrm{d}f_2}{\mathrm{d}x_2} \end{bmatrix}, \tag{25}$$

which corresponds to the bottom left-most 2×2 block in the $\mathrm{d}\boldsymbol{t}/\,\mathrm{d}\boldsymbol{t}$ matrix.

To compute these four derivatives, we can apply the forward mode or reverse mode of AD. These methods are equivalent to solving the corresponding form of the unifying chain rule, Eq. (7), using forward or back substitution, respectively, after assembling the Jacobian of partial derivatives using symbolic differentiation.

For the forward mode, we have

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -t_2^2 & -2t_1t_2 & 1 & 0 & 0 & 0 & 0 \\ -\frac{t_2^2 \cos t_1}{t_3} & -\frac{2t_2 \sin t_1}{t_3} & \frac{t_2^2 \sin t_1}{t_3^2} & 1 & 0 & 0 & 0 \\ -\frac{\cos t_1}{t_3} & 0 & \frac{\sin t_1}{t_3^2} & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 & 0 \\ -t_5 \cos t_1 & 0 & 0 & 0 & -\sin t_1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \frac{\mathrm{d}t_2}{\mathrm{d}t_1} & 1 \\ \frac{\mathrm{d}t_3}{\mathrm{d}t_1} & \frac{\mathrm{d}t_3}{\mathrm{d}t_2} \\ \frac{\mathrm{d}t_4}{\mathrm{d}t_1} & \frac{\mathrm{d}t_4}{\mathrm{d}t_2} \\ \frac{\mathrm{d}t_5}{\mathrm{d}t_1} & \frac{\mathrm{d}t_5}{\mathrm{d}t_2} \\ \frac{\mathrm{d}t_6}{\mathrm{d}t_1} & \frac{\mathrm{d}t_6}{\mathrm{d}t_2} \\ \frac{\mathrm{d}t_7}{\mathrm{d}t_1} & \frac{\mathrm{d}t_7}{\mathrm{d}t_2} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{26}$$

where the large matrix is the lower-triangular Jacobian of partial derivatives in Eq. (7), the solution vectors represent the first two columns of the Jacobian of total derivatives, and the right-hand side is the first two columns of the identity matrix. The derivatives of interest are in the lower 2×2 block in the $\mathrm{d}\boldsymbol{t}/\,\mathrm{d}\boldsymbol{t}$ matrix. The four derivatives of interest can be computed by performing two forward substitutions to obtain the first two columns of $\mathrm{d}\boldsymbol{t}/\,\mathrm{d}\boldsymbol{t}$. In addition to computing the derivatives of interest, we have also computed the derivatives of all variables with respect to t_1 and t_2. In the next subsection we will see how this is implemented in practice.

Now we demonstrate the mathematical analog of the reverse mode of AD. Again, since we are interested only in the derivatives of the outputs with respect to the inputs, we can ignore some of the columns of $\mathrm{d}\boldsymbol{t}/\,\mathrm{d}\boldsymbol{t}$.

In this case we want to keep only the last two columns, so we obtain

$$\begin{bmatrix} 1 & 0 & -t_2^2 & -\frac{t_2^2 \cos t_1}{t_3} & -\frac{\cos t_1}{t_3} & 0 & -t_5 \cos t_1 \\ 0 & 1 & -2t_1 t_2 & -\frac{2t_2 \sin t_1}{t_3} & 0 & 0 & 0 \\ 0 & 0 & 1 & \frac{t_2^2 \sin t_1}{t_3^2} & \frac{\sin t_1}{t_3^2} & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & -\sin t_1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\mathrm{d}t_6}{\mathrm{d}t_1} & \frac{\mathrm{d}t_7}{\mathrm{d}t_1} \\ \frac{\mathrm{d}t_6}{\mathrm{d}t_2} & \frac{\mathrm{d}t_7}{\mathrm{d}t_2} \\ \frac{\mathrm{d}t_6}{\mathrm{d}t_3} & \frac{\mathrm{d}t_7}{\mathrm{d}t_3} \\ \frac{\mathrm{d}t_6}{\mathrm{d}t_4} & \frac{\mathrm{d}t_7}{\mathrm{d}t_4} \\ \frac{\mathrm{d}t_6}{\mathrm{d}t_5} & \frac{\mathrm{d}t_7}{\mathrm{d}t_5} \\ 1 & \frac{\mathrm{d}t_7}{\mathrm{d}t_6} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{27}$$

where the large matrix is the upper-triangular, transposed Jacobian of partial derivatives in Eq. (7), the solution vectors represent the last two columns of the transposed Jacobian of total derivatives, and the right-hand side is the last two columns of the identity matrix. The derivatives of interest are in the upper 2×2 block in the $\mathrm{d}\boldsymbol{t}/\,\mathrm{d}\boldsymbol{t}^T$ matrix. In contrast to the forward mode, the derivatives of interest are computed by performing two back substitutions, through which the derivatives of t_6 and t_7 with respect to all variables are computed incidentally.

Implementation and Tools

There are two main approaches to AD. The *source-code transformation* approach intersperses lines of code that compute the derivatives of the original code [29, 40] and is demonstrated in the next subsection. The additional lines of code and other required changes to the original source code are made automatically using an AD tool. The other approach to AD uses *operator overloading* [29, 75]. In this approach, the original code does not change, but the variable types and the operations are redefined. 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 [29].

When we introduced the complex-step method, we mentioned that there is a close connection between the complex-step method and AD. In essence, the complex-step method can be viewed as an implementation of operator-loading AD, since the complex number is a type that includes the original real number, and the complex part represents the derivative. All the complex arithmetic operations result in an approximation of the derivative in the complex part [61]. In theory, the complex step yields the same result as AD only in the limit, as the complex-step size approaches zero, but if this step is made small enough, identical results can be obtained with finite-precision arithmetic.

There are a variety of AD tools available for the most popular programming languages used in scientific computing, including Fortran, C/C++, 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. For C/C++, the software packages ADIC [15], OpenAD [93], and Tapenade [31] use the source-code transformation approach, while ADOL-C [30] and FADBAD++ [10] use the operator-loading approach. For Fortran, the following tools implement source-code transformation: ADIF95 [92], ADIFOR [19], OpenAD/F [94], TAF [26], and Tapenade [31, 70]. The operator-overloading approach is implemented in AD01 [75], ADOL-F [85], IMAS [78], and OPFAD/OPRAD [9, 18]. There are also AD tools for Matlab [14], including ADiMat [13], which combines source transformation and operator overloading.

Source-Code Transformation Example

In practice, automatic differentiation performs the sequence of operations shown at the bottom of Fig. 7 by changing the original source code. Applying the forward mode of the source-code transformation tool Tapenade [31] to the Fortran code listed in Fig. 4 results in the code listed in Fig. 8. Note that for each variable assignment, Tapenade added an assignment (with a `d` added to the variable name) that corresponds to the symbolic differentiation of the operators in the original assignment. The variable `xd` is the *seed vector* that determines the input variable of interest with respect to which we differentiate. In our case we want to compute all four terms in the Jacobian (25), so we need to run this code twice, first with `xd(1)=1`, `xd(2)=0` and then with `xd(1)=0`, `xd(2)=1`, to get the first and second columns of the Jacobian, respectively. If desired, directional derivatives can also be computed by combining multiple nonzero entries in the seed vector.

```
FUNCTION F_D(x, xd, f)
  REAL :: x(2), xd(2)
  REAL :: det, detd
  REAL :: y(2), yd(2)
  REAL :: f(2), f_d(2)
  detd = xd(1)*x(2)**2 + x(1)*2*x(2)*xd(2)
  det = 2 + x(1)*x(2)**2
  yd = 0.0
  yd(1) = ((2*x(2)*xd(2)*SIN(x(1))+x(2)**2*xd(1)*COS(x(1)))*det-x(2)**2*&
&     SIN(x(1))*detd)/det**2
  y(1) = x(2)**2*SIN(x(1))/det
  yd(2) = (xd(1)*COS(x(1))*det-SIN(x(1))*detd)/det**2
  y(2) = SIN(x(1))/det
  f_d = 0.0
  f_d(1) = yd(1)
  f(1) = y(1)
  f_d(2) = yd(2)*SIN(x(1)) + y(2)*xd(1)*COS(x(1))
  f(2) = y(2)*SIN(x(1))
  RETURN
END FUNCTION F_D
```

Figure 8: Fortran code differentiated using the forward mode of the algorithmic differentiation tool Tapenade.

Applying the reverse mode of Tapenade to our original Fortran code results in the code listed in Fig. 9. The resulting code is more complex than that generated in the forward mode because the reverse mode needs to do the computations in reverse (which corresponds to the back substitution step in the solution of Eq. (27)). To complicate things further, this reverse sequence of computations requires the values of various variables in the original code. The resulting procedure is one in which the original calculations are first performed (lines up to the assignment of `y(2)`), and only then do the reverse computations take place, ending with the final assignment of the derivatives of interest, `xb`.

In this example, no variables are overwritten. If that were the case, then Tapenade would store all temporary values of the variables in the original computation using its own memory space in order to be able to recall these variables at the reverse computation stage.

Tapenade and other AD tools have many advanced features that are not mentioned here. For example, it is possible to significantly decrease the memory requirement of the reverse mode. For more details, please see the references for each AD tool.

4.3 Analytic Methods

Like AD, the numerical precision of analytic methods is the same as that of the original algorithm. In addition, analytic methods are usually more efficient than AD for a given problem. However, analytic methods are much more involved than 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 Eq. (5), which we repeat here for convenience:

$$\boldsymbol{f} = \boldsymbol{F}(\boldsymbol{x}, \boldsymbol{Y}(\boldsymbol{x})). \tag{28}$$

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:

$$\boldsymbol{r} = \boldsymbol{R}(\boldsymbol{x}, \boldsymbol{Y}(\boldsymbol{x})) = 0. \tag{29}$$

By writing the computational model in this form, we have assumed 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

```fortran
SUBROUTINE F_B(x, xb, fb)
  REAL :: x(2), xb(2),
  REAL :: y(2), yb(2)
  REAL :: f(2), fb(2)
  REAL :: det, detb, tempb, temp
  det = 2 + x(1)*x(2)**2
  y(1) = x(2)**2*SIN(x(1))/det
  y(2) = SIN(x(1))/det
  xb = 0.0
  yb = 0.0
  yb(2) = yb(2) + SIN(x(1))*fb(2)
  xb(1) = xb(1) + y(2)*COS(x(1))*fb(2)
  fb(2) = 0.0
  yb(1) = yb(1) + fb(1)
  xb(1) = xb(1) + COS(x(1))*yb(2)/det
  detb = -(SIN(x(1))*yb(2)/det**2)
  yb(2) = 0.0
  tempb = SIN(x(1))*yb(1)/det
  temp = x(2)**2/det
  xb(2) = xb(2) + 2*x(2)*tempb
  detb = detb - temp*tempb
  xb(1) = xb(1) + x(2)**2*detb + temp*COS(x(1))*yb(1)
  xb(2) = xb(2) + x(1)*2*x(2)*detb
END SUBROUTINE F_B
```

Figure 9: Fortran code differentiated using the reverse mode of the algorithmic differentiation tool Tapenade.

the continuous approach in this paper, but ample literature can be found on the subject [3, 27, 37, 38, 50, 87], including discussions comparing the two approaches [23, 65, 72].

In this section we derive the two forms of the analytic method—the direct and the adjoint forms—in two ways. The first derivation follows the derivation that is typically presented in the literature, while the second derivation is based on the unifying chain rule (9), and is a new perspective that connects analytic methods to AD.

4.3.1 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

$$\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}} = \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}} + \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}}\frac{\mathrm{d}\boldsymbol{y}}{\mathrm{d}\boldsymbol{x}}, \tag{30}$$

where the result is an $n_f \times n_x$ matrix. As previously mentioned, it is 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 (29) 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 functions.

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 (AD).

Since the governing equations must always be satisfied, the total derivative of the residuals (29) with respect to the design variables must also be zero. Thus, using the chain rule we obtain

$$\frac{\mathrm{d}\boldsymbol{r}}{\mathrm{d}\boldsymbol{x}} = \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}} + \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{y}}\frac{\mathrm{d}\boldsymbol{y}}{\mathrm{d}\boldsymbol{x}} = 0. \tag{31}$$

The computation of the total derivative matrix $\mathrm{d}\boldsymbol{y}/\,\mathrm{d}\boldsymbol{x}$ in Eqs. (30) and (31) 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 (31) provide the means for computing the total sensitivity matrix $\mathrm{d}\boldsymbol{y}/\,\mathrm{d}\boldsymbol{x}$. We rewrite these equations as

$$\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{y}}\frac{\mathrm{d}\boldsymbol{y}}{\mathrm{d}\boldsymbol{x}} = -\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}. \tag{32}$$

Substituting this result into the total derivative equation (30), we obtain

$$\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}} = \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}} - \underbrace{\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}}\overbrace{\left[\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{y}}\right]^{-1}\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}}^{-\frac{\mathrm{d}\boldsymbol{y}}{\mathrm{d}\boldsymbol{x}}}}_{\boldsymbol{\psi}}. \tag{33}$$

The inverse of the square Jacobian matrix $\partial \boldsymbol{R}/\partial \boldsymbol{y}$ is not necessarily calculated explicitly. 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 (33) 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. Figure 10 shows the sizes of the matrices in Eq. (33), which depend on the shape of $\mathrm{d}\boldsymbol{f}/\,\mathrm{d}\boldsymbol{x}$. The diagrams in Fig. 10 illustrate why the direct method is preferable when $n_x < n_f$ and why the adjoint method is more efficient when $n_x > n_f$. This is the same fundamental difference that we observed between the forward and reverse forms of the unifying chain rule.

4.3.2 Direct Method

The direct method involves solving the linear system with $-\partial \boldsymbol{R}/\partial \boldsymbol{x}$ as the right-hand side vector, which results in the linear system (32). 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 Eq. (30) to obtain the derivatives of interest, $\mathrm{d}\boldsymbol{f}/\,\mathrm{d}\boldsymbol{x}$.

As in the case of finite differences, the cost of computing derivatives with the direct method is proportional to the number of design variables, n_x. In a case where the computational model is a nonlinear system, the direct method can be advantageous. Both methods require the solution of a system of the same size n_x times, but the direct method just solves the linear system (32), while the finite-difference method solves the original nonlinear system (4). Even though the various solutions required for the finite-difference method can be warm-started from a previous solution, a nonlinear solution will typically require multiple iterations to converge. The direct method is even more advantageous when a factorization of $\partial \boldsymbol{R}/\partial \boldsymbol{y}$ is available, since each solution of the linear system consists in an inexpensive back substitution. There are some cases where a hybrid approach that combines the direct and adjoint methods is advantageous [41].

4.3.3 Numerical Example: Direct Method

We now return to the numerical example introduced in Section 4 and apply the direct method. Since there are just two state variables, Eq. (32) is the following 2×2 linear system:

$$\begin{bmatrix} -\dfrac{\partial R_1}{\partial y_1} & -\dfrac{\partial R_1}{\partial y_2} \\ -\dfrac{\partial R_2}{\partial y_1} & -\dfrac{\partial R_2}{\partial y_2} \end{bmatrix} \begin{bmatrix} \dfrac{\mathrm{d}y_1}{\mathrm{d}x_1} & \dfrac{\mathrm{d}y_1}{\mathrm{d}x_2} \\ \dfrac{\mathrm{d}y_2}{\mathrm{d}x_1} & \dfrac{\mathrm{d}y_2}{\mathrm{d}x_2} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial R_1}{\partial x_1} & \dfrac{\partial R_1}{\partial x_2} \\ \dfrac{\partial R_2}{\partial x_1} & \dfrac{\partial R_2}{\partial x_2} \end{bmatrix}.$$

In a more realistic example, the computation of the partial derivatives would be more involved, since the residuals typically do not have simple analytical expressions. In this case, the residuals do have simple

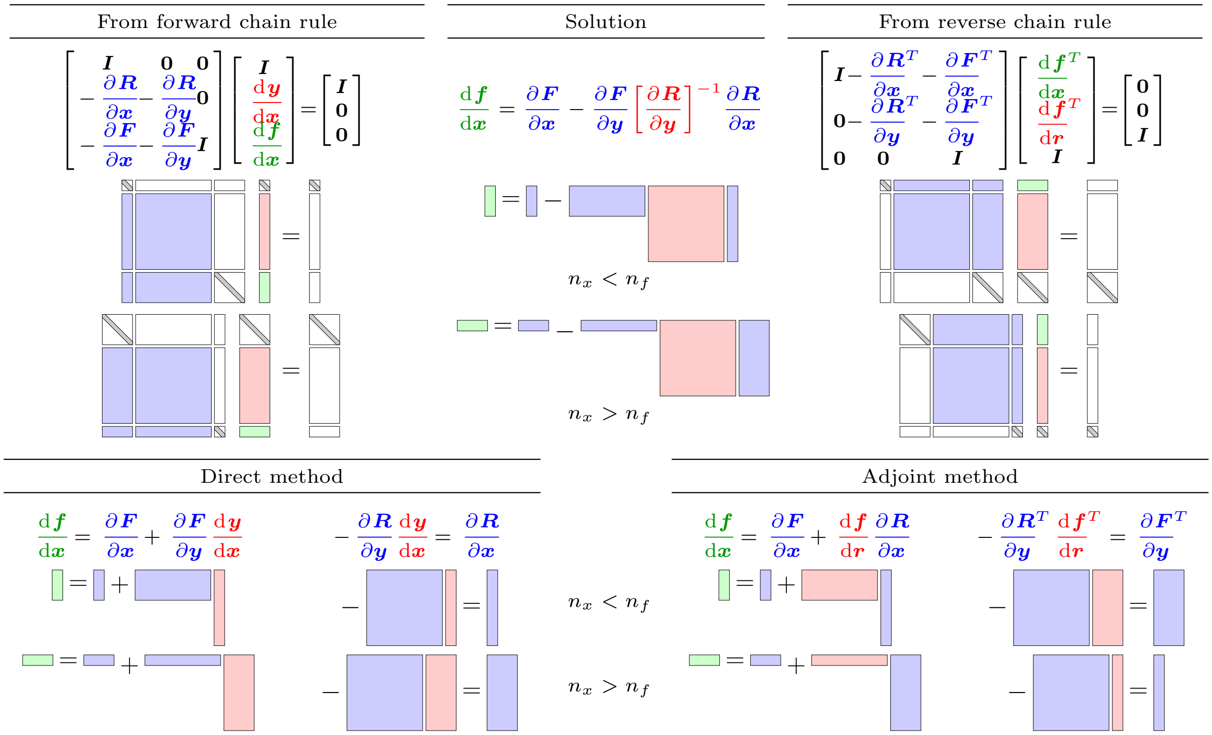

From forward chain rule
Solution
From reverse chain rule
$\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}} = \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}} - \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}} \left[\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{y}}\right]^{-1} \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}$
$n_x < n_f$
$n_x > n_f$
Direct method
Adjoint method
$\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}} = \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}} + \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}} \frac{\mathrm{d}\boldsymbol{y}}{\mathrm{d}\boldsymbol{x}}$
$-\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{y}} \frac{\mathrm{d}\boldsymbol{y}}{\mathrm{d}\boldsymbol{x}} = \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}$
$\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}} = \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}} + \frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{r}} \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}$
$-\frac{\partial \boldsymbol{R}^T}{\partial \boldsymbol{y}} \frac{\mathrm{d}\boldsymbol{f}^T}{\mathrm{d}\boldsymbol{r}} = \frac{\partial \boldsymbol{F}^T}{\partial \boldsymbol{y}}$

Figure 10: Block matrix diagrams illustrating the structure of the direct and adjoint equations, assuming that $n_y \gg n_x, n_f$. The blue matrices contain partial derivatives, which are relatively cheap to compute, and the red matrices contain the total derivatives computed by solving the linear systems.

forms, so we can use symbolic differentiation to compute each partial derivative to obtain

$$\begin{bmatrix} -x_1 & -2 \\ 1 & -x_2^2 \end{bmatrix} \begin{bmatrix} \frac{\mathrm{d}y_1}{\mathrm{d}x_1} & \frac{\mathrm{d}y_1}{\mathrm{d}x_2} \\ \frac{\mathrm{d}y_2}{\mathrm{d}x_1} & \frac{\mathrm{d}y_2}{\mathrm{d}x_2} \end{bmatrix} = \begin{bmatrix} y_1 - \cos x_1 & 0 \\ 0 & 2x_2 y_2 \end{bmatrix}.$$

Although we are not interested in finding any derivatives with respect to x_2, both right-hand sides of the system are shown for completeness. Since the analytic methods are derived based on a linearization of the system at a converged state, we must evaluate the system at $[x_1, x_2] = [1, 1]$ and $[y_1, y_2] = [\sin 1/3, \sin 1/3]$. The computed values for $\mathrm{d}y_1/\mathrm{d}x_1$ and $\mathrm{d}y_2/\mathrm{d}x_1$ can be used to find $\mathrm{d}f_1/\mathrm{d}x_1$ via the following equation:

$$\frac{\mathrm{d}f_1}{\mathrm{d}x_1} = \frac{\partial F_1}{\partial x_1} + \frac{\partial F_1}{\partial y_1}\frac{\mathrm{d}y_1}{\mathrm{d}x_1} + \frac{\partial F_1}{\partial y_2}\frac{\mathrm{d}y_2}{\mathrm{d}x_1},$$

which simplifies to

$$\frac{\mathrm{d}f_1}{\mathrm{d}x_1} = \frac{\mathrm{d}y_1}{\mathrm{d}x_1}$$

when the partial derivatives are evaluated.

4.3.4 Adjoint Method

Adjoint methods have been known and used for over three decades. They were first applied to solve optimal control problems and thereafter used to perform sensitivity analysis of linear structural finite element models. The first application to fluid dynamics is due to Pironneau [73]. The method was then extended by Jameson to perform airfoil shape optimization [37], and since then it has been used to optimize airfoils suitable for multipoint operation [67] and to design laminar-flow airfoils [22]. The adjoint method has been extended to three-dimensional problems, leading to applications such as the aerodynamic shape optimization of complete aircraft configurations [54, 76, 77] and shape optimization considering both aerodynamics and structures [43, 62]. The adjoint method has since been generalized for multidisciplinary systems [63].

The adjoint equation can be seen in the total derivative equation (33), where 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*,

$$\left[\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{y}}\right]^T \boldsymbol{\psi} = -\left[\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}}\right]^T, \tag{34}$$

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. 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 Eq. (33) to find the total sensitivity,

$$\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}} = \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}} + \boldsymbol{\psi}^T \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}. \tag{35}$$

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 or 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 [44, 57].

By comparing the direct and adjoint method, we also notice that all the partial derivative terms that need to be computed are identical, and that the difference in their relative cost comes only from the choice of which right-hand side to use with the residual Jacobian.

4.3.5 Numerical Example: Adjoint Method

We now apply the adjoint method to compute $\mathrm{d}f_1/\mathrm{d}x_1$ in the numerical example from Section 4. In this case, the linear system is

$$\begin{bmatrix} -\dfrac{\partial R_1}{\partial y_1} & -\dfrac{\partial R_2}{\partial y_1} \\ -\dfrac{\partial R_1}{\partial y_2} & -\dfrac{\partial R_2}{\partial y_2} \end{bmatrix} \begin{bmatrix} \Psi_{11} & \Psi_{12} \\ \Psi_{21} & \Psi_{22} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial F_1}{\partial y_1} & \dfrac{\partial F_2}{\partial y_1} \\ \dfrac{\partial F_1}{\partial y_2} & \dfrac{\partial F_2}{\partial y_2} \end{bmatrix}.$$

We can obtain the partial derivatives through symbolic differentiation, and the second column of the adjoint matrix is not required since we are interested only in f_1, so the linear system becomes

$$\begin{bmatrix} -x_1 & 1 \\ -2 & -x_2^2 \end{bmatrix} \begin{bmatrix} \Psi_{11} \\ \Psi_{21} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & \sin x_1 \end{bmatrix}.$$

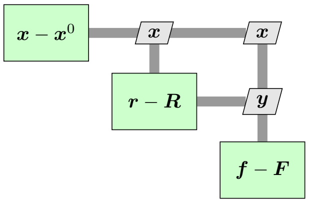
This linear system can now be solved to find the adjoint variables Ψ_{11} and Ψ_{21}. After evaluating the system at $[x_1, x_2] = [1, 1]$ and using the resulting values of the state variables, $[y_1, y_2] = [\frac{\sin 1}{3}, \frac{\sin 1}{3}]$, we can compute $\mathrm{d}f_1/\mathrm{d}x_1$ using

$$\frac{\mathrm{d}f_1}{\mathrm{d}x_1} = \frac{\partial F_1}{\partial x_1} + \Psi_{11}\frac{\partial R_1}{\partial x_1} + \Psi_{21}\frac{\partial R_2}{\partial x_1}$$

where we can replace the partial derivatives with the symbolically differentiated terms to obtain

$$\frac{\mathrm{d}f_1}{\mathrm{d}x_1} = \Psi_{11}(y_1 - \cos x_1).$$

Variables and Constraints


$\boldsymbol{x}-\boldsymbol{x}^0$
$\boldsymbol{x}$
$\boldsymbol{x}$
$\boldsymbol{r}-\boldsymbol{R}$
$\boldsymbol{y}$
$\boldsymbol{f}-\boldsymbol{F}$

$$\boldsymbol{v}=\begin{bmatrix}\boldsymbol{x}\\ \boldsymbol{y}\\ \boldsymbol{f}\end{bmatrix}$$

$$\boldsymbol{C}(\boldsymbol{v})=\begin{bmatrix}\boldsymbol{x}-\boldsymbol{x}^0\\ \boldsymbol{r}-\boldsymbol{R}(\boldsymbol{x},\boldsymbol{y})\\ \boldsymbol{f}-\boldsymbol{F}(\boldsymbol{x},\boldsymbol{y})\end{bmatrix}$$

Derivation

$$\left[\frac{\partial \boldsymbol{C}}{\partial \boldsymbol{v}}\right]\left[\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}\mathrm{c}}\right]=\boldsymbol{I}=\left[\frac{\partial \boldsymbol{C}}{\partial \boldsymbol{v}}\right]^T\left[\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}\mathrm{c}}\right]^T$$

$$\begin{bmatrix}\frac{\partial(\boldsymbol{x}-\boldsymbol{x}^0)}{\partial\boldsymbol{x}} & \frac{\partial(\boldsymbol{x}-\boldsymbol{x}^0)}{\partial\boldsymbol{y}} & \frac{\partial(\boldsymbol{x}-\boldsymbol{x}^0)}{\partial\boldsymbol{f}}\\ \frac{\partial(\boldsymbol{r}-\boldsymbol{R})}{\partial\boldsymbol{x}} & \frac{\partial(\boldsymbol{r}-\boldsymbol{R})}{\partial\boldsymbol{y}} & \frac{\partial(\boldsymbol{r}-\boldsymbol{R})}{\partial\boldsymbol{f}}\\ \frac{\partial(\boldsymbol{f}-\boldsymbol{F})}{\partial\boldsymbol{x}} & \frac{\partial(\boldsymbol{f}-\boldsymbol{F})}{\partial\boldsymbol{y}} & \frac{\partial(\boldsymbol{f}-\boldsymbol{F})}{\partial\boldsymbol{f}}\end{bmatrix}\begin{bmatrix}\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}\boldsymbol{x}} & \frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}\boldsymbol{r}} & \frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}\boldsymbol{f}}\\ \frac{\mathrm{d}\boldsymbol{y}}{\mathrm{d}\boldsymbol{x}} & \frac{\mathrm{d}\boldsymbol{y}}{\mathrm{d}\boldsymbol{r}} & \frac{\mathrm{d}\boldsymbol{y}}{\mathrm{d}\boldsymbol{f}}\\ \frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}} & \frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{r}} & \frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{f}}\end{bmatrix}=\boldsymbol{I}=\begin{bmatrix}\frac{\partial(\boldsymbol{x}-\boldsymbol{x}^0)^T}{\partial\boldsymbol{x}} & \frac{\partial(\boldsymbol{r}-\boldsymbol{R})^T}{\partial\boldsymbol{x}} & \frac{\partial(\boldsymbol{f}-\boldsymbol{F})^T}{\partial\boldsymbol{x}}\\ \frac{\partial(\boldsymbol{x}-\boldsymbol{x}^0)^T}{\partial\boldsymbol{y}} & \frac{\partial(\boldsymbol{r}-\boldsymbol{R})^T}{\partial\boldsymbol{y}} & \frac{\partial(\boldsymbol{f}-\boldsymbol{F})^T}{\partial\boldsymbol{y}}\\ \frac{\partial(\boldsymbol{x}-\boldsymbol{x}^0)^T}{\partial\boldsymbol{f}} & \frac{\partial(\boldsymbol{r}-\boldsymbol{R})^T}{\partial\boldsymbol{f}} & \frac{\partial(\boldsymbol{f}-\boldsymbol{F})^T}{\partial\boldsymbol{f}}\end{bmatrix}\begin{bmatrix}\frac{\mathrm{d}\boldsymbol{x}^T}{\mathrm{d}\boldsymbol{x}} & \frac{\mathrm{d}\boldsymbol{y}^T}{\mathrm{d}\boldsymbol{x}} & \frac{\mathrm{d}\boldsymbol{f}^T}{\mathrm{d}\boldsymbol{x}}\\ \frac{\mathrm{d}\boldsymbol{x}^T}{\mathrm{d}\boldsymbol{r}} & \frac{\mathrm{d}\boldsymbol{y}^T}{\mathrm{d}\boldsymbol{r}} & \frac{\mathrm{d}\boldsymbol{f}^T}{\mathrm{d}\boldsymbol{r}}\\ \frac{\mathrm{d}\boldsymbol{x}^T}{\mathrm{d}\boldsymbol{f}} & \frac{\mathrm{d}\boldsymbol{y}^T}{\mathrm{d}\boldsymbol{f}} & \frac{\mathrm{d}\boldsymbol{f}^T}{\mathrm{d}\boldsymbol{f}}\end{bmatrix}$$

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

Direct method

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

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

Adjoint method

$$\frac{\partial\boldsymbol{R}^T}{\partial\boldsymbol{y}}\frac{\mathrm{d}\boldsymbol{f}^T}{\mathrm{d}\boldsymbol{r}}=-\frac{\partial\boldsymbol{F}^T}{\partial\boldsymbol{y}}$$

$$\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}}=\frac{\partial\boldsymbol{F}}{\partial\boldsymbol{x}}+\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{r}}\frac{\partial\boldsymbol{R}}{\partial\boldsymbol{x}}$$

Figure 11: Derivation of the analytic methods: direct and adjoint.

4.3.6 Derivation from Unifying Chain Rule

Figure 11 shows the derivation of the analytic methods from the unifying chain rule. In this case, the variables are the independent variables $\boldsymbol{x}$, the state variables $\boldsymbol{y}$, and the functions of interest, $\boldsymbol{f}$. The constraints force the independent variables to be equal to the specified values, and the residuals and functions of interest to be equal to the values resulting from the computational model.

Substituting these definitions in the unifying chain rule yields the familiar direct and adjoint methods, whose equations are shown at the bottom of Fig. 11. As we can see, the direct method comes from the forward chain rule, while the adjoint method is derived from the reverse chain rule.

Note that the adjoint variables here are represented by $\mathrm{d}\boldsymbol{f}/\mathrm{d}\boldsymbol{r}$, which is a notation that has not been used previously in the literature. However, this notation is consistent with the physical interpretation that has been noted before, which is that the adjoint variables represent the derivative of the functions of interest with respect to variations in the residuals of the governing equations. Also, note that $\mathrm{d}\boldsymbol{f}/\mathrm{d}\boldsymbol{r}=\boldsymbol{\Psi}^T$, because the convention adopted in the literature defines an adjoint vector for a given output quantity as a column vector.

Although we do not address time-dependent problems here, the equations in Fig. 11 can be extended to handle such problems. To accomplish this, we would include the states for all the time instances in the state variable vector, i.e., $\boldsymbol{y}=\left[\boldsymbol{y}_1^T,\ldots,\boldsymbol{y}_{n_t}^T\right]^T$. Substituting this $\boldsymbol{y}$ into the unifying chain rule would yield a block lower triangular $\partial\boldsymbol{R}/\partial\boldsymbol{y}$. Since this matrix is transposed in the adjoint equations, the linear system becomes block upper triangular. The consequence of this is that the time-dependent adjoint equations need to be solved using block back substitution, starting from the last time instance. Because the adjoint equations depend on the states themselves, we must store the complete time history of the states when the time-dependent solution of the states is obtained. More details on the time-dependent adjoint method and proposed solutions to handle the memory requirements have been presented by various authors [58, 81, 96].

4.4 Coupled Analytic Methods

We now extend the analytic methods derived in the previous section to multidisciplinary systems. The direct and adjoint methods for multidisciplinary systems can be derived by partitioning the various variables by discipline as follows:

$$\boldsymbol{R} = [\boldsymbol{R}_1^T, \ldots, \boldsymbol{R}_N^T]^T, \quad \boldsymbol{y} = [\boldsymbol{y}_1^T, \ldots, \boldsymbol{y}_N^T]^T \tag{36}$$

where N is the number of disciplines. All the design variables are included in $\boldsymbol{x}$. If we substitute these vectors into the unifying chain rule, we obtain the block matrix equations shown at the bottom of Fig. 11.

These are the coupled versions of the direct and adjoint methods, respectively. The coupled direct method was first developed by Bloebaum [17] and Sobieszczanski-Sobieski [88] while the coupled adjoint was originally developed by Martins et al. [60, 63]. Both the coupled direct and adjoint methods have since been applied to the aerostructural design optimization of aircraft wings [44, 62, 64, 69].

Figure 13 shows another alternative for obtaining the total derivatives of multidisciplinary systems that was first developed by Sobieszczanski-Sobieski [88] for the direct method, and by Martins et al. [63] for the adjoint method. The advantage of this approach is that we do not need to know the residuals of a given disciplinary solver but can instead use the *coupling variables.* To derive the direct and adjoint versions of this approach within our mathematical framework, we consider the same definition of the variables but replace the constraints with the residuals of the coupling variables,

$$\boldsymbol{C}_i = \boldsymbol{Y}_i - \boldsymbol{y}_i \tag{37}$$

where the $\boldsymbol{y}_i$ vector contains the coupling variables of the i^{th} discipline, and $\boldsymbol{Y}_i$ is the vector of functions that explicitly defines these variables. This leads to the equations at the bottom of Fig. 13, which we call the *functional* form. This contrasts with the *residual* form shown in Fig. 12.

A useful outcome of the unification of these two forms using Eq. (9) is a natural extension to *hybrid* methods that combine disciplines that use the residual form with other disciplines that use the functional form. To derive and prove the validity of such a method, we can set the appropriate entries of $\boldsymbol{C}$ to be $\boldsymbol{r}_i - \boldsymbol{R}_i$ or $\boldsymbol{y}_i - \boldsymbol{Y}_i$, based on which form is appropriate for the i^{th} discipline. This hybrid form for computing coupled derivatives is likely to be very useful for multidisciplinary systems in which the residuals of the governing equations and their partial derivatives are available for some disciplines but not for others, such as disciplines that run black-box codes. The hybrid form is demonstrated in Fig. 14 for the numerical example.

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 coupling variables in the i^{th} discipline as well as any other disciplines, signifying that this development allows for coupling to be present among the coupling variables, even within a discipline. The only change in the final equations would be the replacement of the identity submatrices in the $\partial \boldsymbol{Y}/\partial \boldsymbol{y}$ matrix with submatrices that have 1 on the diagonal and $-\partial \boldsymbol{Y}_i/\partial \boldsymbol{y}_i$ in the off-diagonals representing the dependence of coupling variables on each other within the same discipline.

4.4.1 Numerical Example

To illustrate the coupled analytic methods, we now interpret the numerical example from Section 4 as a multidisciplinary system. In this interpretation, the state variable y_1 belongs to discipline 1 and is solely constrained by residual R_1, while y_2 and R_2 are the corresponding variables associated with discipline 2.

From Fig. 12, we can see that the residual approach yields the same equations as the analytic methods, except for the classification of blocks corresponding to different disciplines in the former case. In this example, each block is 1 × 1, since both disciplines have only one state variable.

For the functional approach, the residuals are eliminated from the equations and it is assumed that explicit expressions can be found for all state variables in terms of input variables and state variables from other disciplines. In most cases, the explicit computation of state variables involves solving the nonlinear system corresponding to the discipline. However, in this example, this is simplified because the residuals are linear in the state variables and each discipline has only one state variable. Thus, the explicit forms are

$$Y_1(x_1, x_2, y_2) = -\frac{2y_2}{x_1} + \frac{\sin x_1}{x_1} \tag{38}$$

$$Y_2(x_1, x_2, y_1) = \frac{y_1}{x_2^2}. \tag{39}$$

Figure 14 shows how $\mathrm{d}f_1/\,\mathrm{d}x_1$ can be computed using the coupled residual, functional, and hybrid approaches, which correspond to cases in which neither discipline, both disciplines, or just discipline 2 has this explicit form.

5 Conclusions

In this paper, we presented a simple matrix equation, the unifying chain rule (9), which achieves a full unification of all known discrete methods for computing derivatives for general computational models. By selecting the right variables to represent the computational model and the right equations to constrain them, we can derive any existing derivative computation method from the unifying chain rule in a straightforward manner.

Within this unification, any method is defined by three choices: the variables and constraints exposed in the method, the strategy for solving the linear system, and the method for computing the Jacobian of partial derivatives in the unifying chain rule (9).

When it comes to methods for computing partial derivatives, we presented finite differences, the complex-step method, and symbolic differentiation, and we discussed their relative advantages and disadvantages.

We then discussed the methods for computing total derivatives—monolithic differentiation, algorithmic differentiation, analytic methods, and coupled analytic methods—that come about from inserting particular choices of variables and constraints into the unifying chain rule (9).

Black-box finite differences and the complex-step method fit into this unification if we consider the inputs and outputs as the only variables of interest. In this case, which we called monolithic differentiation, we showed that the unifying chain rule (9) yields a trivial identity: each total derivative of an output with respect to an input is equal to the corresponding partial derivative, which in turn is computed using finite differences or the complex-step method.

Algorithmic differentiation exposes quantities computed at the line-of-code level as the relevant variables. Since each variable depends only on earlier ones, the Jacobian of partial derivatives is lower or upper triangular, and the nonzero terms are evaluated using symbolic differentiation. For the same reason, the Jacobian of total derivatives is also triangular, allowing the use of forward or back substitution to solve the linear system in the forward or reverse form of the unifying chain rule (9), respectively, yielding the corresponding mode of AD.

For analytic methods, the inputs, outputs, states, and residuals represent the variables and functions of interest. We showed that in this case the unifying chain rule (9) contains a block triangular Jacobian of partial derivatives. After we block solve to simplify the linear system, the forward and reverse forms of the unifying chain rule produce the direct and adjoint methods if the right column in the right-hand side is used.

The Jacobian of the residuals with respect to the states and the other partial derivatives can be computed using any of the monolithic methods or AD. The selective use of AD to compute these terms—with either the forward or the reverse mode—can be particularly advantageous, because it can dramatically decrease the effort of implementing analytic methods [57, 72].

For multidisciplinary systems, two different approaches are possible for both the coupled direct and coupled adjoint methods, resulting in four possible combinations. We showed that separating the states and residuals by discipline yields the residual approach, while the functional approach is obtained by treating each discipline as a black box where we can see only its output variables. In addition, we showed that it is possible to take any combination of 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, because of the limitations of the disciplinary solvers. This perspective on the computation of derivatives of coupled systems extends the previous work of Sobieszczanski-Sobieski [88] and Martins et al. [63].

While the present work is largely theoretical, it has recently led to more practical results. Hwang et al. [35] used the mathematical formulation developed herein as the basis for a computational framework that was applied to the gradient-based MDO of a small satellite. The MDO problem involved tens of thousands of design variables and millions of state variables. The framework greatly facilitated the implementation of the problem and also led to high efficiency in both the multidisciplinary analysis and the optimization. We believe this computational framework greatly reduces development time, computation time, or both for many gradient-based multidisciplinary analysis and optimization applications.

6 Future Research Directions

The development of methods for obtaining the derivatives of computational models has generally lagged the development of the computational models themselves. However, the theory and implementation of efficient methods for computing derivatives is now well understood, so it is now possible to implement the derivative computations concurrently with the code for the computational model.

We would argue that this concurrent development is advisable, since it pays off in the long term to keep in mind that the code is going to be differentiated. It often happens, for example, that artificial discontinuities or noise are introduced by certain numerical procedures, even when the original computational model is smooth and differentiable. These avoidable discontinuities or noise may result in inaccurate derivatives and also cause difficulties for gradient-based optimizers. The concurrent implementation of the model and its derivatives can also provide insight into how to implement the model in a way that makes the derivative computation more efficient. In addition, when derivatives are readily available, they can be used to accelerate the convergence of the model solver using Newton-type methods. Thus, we expect that the development of new solvers will increasingly be accompanied by the development of an efficient method for computing the derivatives of these solvers.

As previously mentioned, one of the major applications of derivatives is gradient-based optimization. Most gradient-based optimization algorithms perform finite differences to estimate the gradients by default, and indeed most users of this software do not change this default setting. As we have discussed in this paper, finite differences are costly when the number of inputs is large, adding to the computational cost of the optimization. Furthermore, finite differences are subject to large errors, which are responsible for many failures in numerical optimization. These problems have prevented the more widespread use of gradient-based optimization because many users erroneously blame the computational cost and lack of robustness on the optimization algorithm itself. The careful development of codes that do not introduce artificial discontinuities to smooth models, and the development of adjoint solvers that can compute gradients much more efficiently will go a long way toward increasing the size of optimization problems that can be solved.

Given the usefulness of having the derivatives of computational models and the maturity of this field of research, it is surprising that the efficient computation of derivatives is not more widespread in commercial software. Adjoint methods have been in place in some structural analysis software for some time [39], but they are only now starting to appear in CFD commercial software [5]. We predict that these advances in commercial software will result in a drastic increase in the use of numerical optimization for engineering design applications.

One of the most significant current challenges in gradient-based optimization is that there is no efficient way for computing all the terms in a large square Jacobian (i.e., when $n_x \approx n_f$), since neither the direct nor the adjoint approaches are advantageous. In design optimization, these large square Jacobians arise when we use gradient-based optimization to solve a problem with many constraints and many design variables, since we need the derivatives of all the constraints with respect to all the variables. This is often the case for structural optimization problems where the weight is minimized with respect to sizing variables, subject to stress constraints. Currently, one way to circumvent this problem is to aggregate the constraints to make the adjoint method more advantageous, but this usually slows down the convergence of the optimization [74]. One promising approach is to develop a matrix-free optimizer that does not require the Jacobian to be stored explicitly but instead relies on matrix-vector products that can be efficiently computed using techniques based on those presented in this review.

The use of efficient methods for the computation of derivatives of multidisciplinary systems is far from widespread. Although the basic theory has been developed over the last two decades, with the coupled adjoint being developed about ten years ago, the practical implementation of these methods remains challenging. If coupling disciplines to obtain a multidisciplinary model is challenging, implementing the computation of the coupled derivatives of that model is bound to be even more so. There are some promising techniques that can automate the computation of coupled derivatives, albeit with some sacrifices in efficiency [59].

Looking into the future, one major challenge is that the coupled derivative methods as presented herein do not scale well with the number of disciplines, because of the potentially large off-diagonal blocks in the Jacobian of partial derivatives, which represent the interdisciplinary derivatives. One possible solution might be to approximate some of the terms and use an indirect method to solve for the derivatives.

Another avenue for future work is the extension of this unification to higher-order derivatives. Equation (9) yields derivatives of any quantity, so we could choose to append first derivatives to the vector of variables and the expressions that define them to the vector of constraints. The solution of the resulting linear system would simultaneously yield first and second derivatives, which could in turn be selected as additional variables in Eq. (9) to compute third derivatives. This technique could be repeatedly applied to obtain expressions for derivatives of any order, though it remains to be seen how efficient such an approach could be and which method—AD, analytic, etc.—could and should be used. At each of the n steps for n^{th} order derivatives, the forward or reverse form of Eq. (9) must be selected, yielding various combinations such as the direct-adjoint and adjoint-direct methods for second derivatives.

Acknowledgments

We would like to thank Graeme J. Kennedy for his insights and Kenneth T. Moore for his suggestions. This paper would not be the same without the valuable feedback from the anonymous reviewers, to whom we are indebted. This work was partially supported by NASA's Subsonic Fixed Wing program through award number NNX11AI19A.

References

[1] H. M. Adelman and R. T. Haftka. Sensitivity analysis of discrete structural systems. *AIAA Journal*, 24(5):823–832, 1986. doi:10.2514/3.48671.

[2] A. H. Al-Mohy and N. J. Higham. The complex step approximation to the fréchet derivative of a matrix function. *Numerical Algorithms*, 53:133–148, 2010. doi:10.1007/s11075-009-9323-y.

[3] W. K. Anderson and V. Venkatakrishnan. Aerodynamic design optimization on unstructured grids with a continuous adjoint formulation. *Computers and Fluids*, 28(4):443–480, 1999.

[4] W. K. Anderson, J. C. Newman, D. L. Whitfield, and E. J. Nielsen. Sensitivity analysis for Navier–Stokes equations on unstructured meshes using complex variables. 39(1):56–63, January 2001. doi:10.2514/2.1270.

[5] Anon. *ANSYS FLUENT Adjoint Solver*. ANSYS, Canonsburg, PA, November 2011.

[6] S. Balay, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.3, Argonne National Laboratory, 2012.

[7] M. Barcelos, H. Bavestrello, and K. Maute. A Schur–Newton–Krylov solver for steady-state aeroelastic analysis and design sensitivity analysis. *Computer Methods in Applied Mechanics and Engineering*, 195: 2050–2069, 2006.

[8] J.-F. Barthelemy and J. Sobieszczanski-Sobieski. Optimum sensitivity derivatives of objective functions in nonlinear programming. *AIAA Journal*, 21:913–915, 1982.

[9] M. Bartholomew-Biggs. *OPFAD — A Users Guide to the OPtima Forward Automatic Differentiation Tool*. Numerical Optimization Centre, University of Hertfordshire, 1995.

[10] C. Bendtsen and O. Stauning. FADBAD, a flexible C++ package for automatic differentiation — using the forward and backward methods. Technical Report IMM-REP-1996-17, Technical University of Denmark, DK-2800 Lyngby, Denmark, 1996. URL `citeseer.nj.nec.com/bendtsen96fadbad.html`.

[11] G. Biros and O. Ghattas. Parallel Lagrange–Newton–Krylov–Schur methods for PDE-constrained optimization. part i: The Krylov–Schur solver. *SIAM Journal on Scientific Computing*, 27(2):687–713, 2005.

[12] G. Biros and O. Ghattas. Parallel Lagrange–Newton–Krylov–Schur methods for PDE-constrained optimization. part ii: The Lagrange–Newton solver and its application to optimal control of steady viscous flows. *SIAM Journal on Scientific Computing*, 27(2):687–713, 2005.

[13] C. Bischof, H. Bucker, B. Lang, A. Rasch, and A. Vehreschild. Combining source transformation and operator overloading techniques to compute derivatives for matlab programs. In *Proceedings of the 2nd IEEE International Workshop on Source Code Analysis and Manipulation, 2002.*, pages 65–72, 2002. doi:10.1109/SCAM.2002.1134106.

[14] C. Bischof, B. Lang, and A. Vehreschild. Automatic differentiation for MATLAB programs. *Proceedings in Applied Mathematics and Mechanics*, 2(1):50–53, 2003. ISSN 1617-7061. doi:10.1002/pamm.200310013.

[15] C. H. Bischof, L. Roh, and A. J. Mauer-Oats. ADIC: An extensible automatic differentiation tool for ANSI-C. *Software — Practice and Experience*, 27(12):1427–1456, 1997.

[16] B. F. Blackwell and K. J. Dowding. Sensitivity analysis and uncertainty propagation of computational models. In W. J. Minkowycz, E. M. Sparrow, and J. Y. Murthy, editors, *Handbook of Numerical Heat Transfer*, chapter 14, pages 443–469. Wiley, 2009. doi:10.1002/9780470172599.ch14.

[17] C. Bloebaum. Global sensitivity analysis in control-augmented structural synthesis. In *Proceedings of the 27th AIAA Aerospace Sciences Meeting*, Reno, NV, Jan. 1989. AlAA 1989-0844.

[18] S. Brown. *OPRAD — A Users Guide to the OPtima Reverse Automatic Differentiation Tool.* Numerical Optimization Centre, University of Hertfordshire, 1995.

[19] A. Carle and M. Fagan. ADIFOR 3.0 overview. Technical Report CAAM-TR-00-02, Rice University, 2000.

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

[21] D. J. W. De Pauw and P. A. Vanrolleghem. Using the complex-step derivative approximation method to calculate local sensitivity functions of highly nonlinear bioprocess models. In *Proceedings 17th IMACS World Congress on Scientific Computation, Applied Mathematics and Simulation (IMACS 2005)*, Paris, France, July 2005.

[22] J. Driver and D. W. Zingg. Numerical aerodynamic optimization incorporating laminar-turbulent transition prediction. *AIAA Journal*, 45(8):1810–1818, 2007. doi:10.2514/1.23569.

[23] R. P. Dwight and J. Brezillon. Effect of approximations of the discrete adjoint on gradient-based optimization. *AIAA Journal*, 44(12):3022–3031, 2006.

[24] J. A. Fike and J. J. Alonso. Automatic differentiation through the use of hyper-dual numbers for second derivatives. In S. Forth, P. Hovland, E. Phipps, J. Utke, and A. Walther, editors, *Recent Advances in Algorithmic Differentiation*, volume 87 of *Lecture Notes in Computational Science and Engineering*, pages 163–173. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-30022-6. doi:10.1007/978-3-642-30023-3_15.

[25] P. Forssen, R. Arnell, and T. Fornstedt. An improved algorithm for solving inverse problems in liquid chromatography. *Computers & Chemical Engineering*, 30(9):1381–1391, 2006. ISSN 0098-1354. doi:10.1016/j.compchemeng.2006.03.004.

[26] R. Giering and T. Kaminski. Applying TAF to generate efficient derivative code of Fortran 77-95 programs. In *Proceedings of GAMM 2002, Augsburg, Germany*, 2002.

[27] M. B. Giles and N. A. Pierce. An introduction to the adjoint approach to design. *Flow, Turbulence and Combustion*, 65:393–415, 2000.

[28] P. Gill, W. Murray, and M. Saunders. SNOPT: An SQP algorithm for large-scale constraint optimization. *SIAM Journal of Optimization*, 12(4):979–1006, 2002.

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

[30] A. Griewank, D. Juedes, and J. Utke. Algorithm 755: ADOL-C: A package for the automatic differentiation of algorithms written in C/C++. *ACM Transactions on Mathematical Software*, 22(2):131–167, 1996. ISSN 0098-3500. URL `http://www.acm.org/pubs/citations/journals/toms/1996-22-2/p131-griewank/`.

[31] L. Hascoët and V. Pascual. Tapenade 2.1 user's guide. Technical report 300, INRIA, 2004. URL `http://www.inria.fr/rrrt/rt-0300.html`.

[32] E. J. Haug, K. K. Choi, and V. Komkov. *Design Sensitivity Analysis of Structural Systems*, volume 177 of *Mathematics in Science and Engineering.* Academic Press, London, UK, 1986. URL `citeseer.ist.psu.edu/625763.html`.

[33] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley. An overview of the trilinos project. *ACM Trans. Math. Softw.*, 31(3):397–423, 2005. ISSN 0098-3500. doi:10.1145/1089014.1089021.

[34] J. Hicken and D. Zingg. A parallel Newton–Krylov solver for the Euler equations discretized using simultaneous approximation terms. *AIAA Journal*, 46(11), 2008.

[35] J. T. Hwang, D. Y. Lee, J. W. Cutler, and J. R. R. A. Martins. Large-scale MDO of a small satellite using a novel framework for the solution of coupled systems and their derivatives. In *Proceedings of the 54th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, Boston, MA, April 2013. doi:10.2514/6.2013-1599.

[36] K. James, J. S. Hansen, and J. R. R. A. Martins. Structural topology optimization for multiple load cases using a dynamic aggregation technique. *Engineering Optimization*, 41(12):1103–1118, Dec. 2009. doi:10.1080/03052150902926827.

[37] A. Jameson. Aerodynamic design via control theory. *Journal of Scientific Computing*, 3(3):233–260, Sept. 1988.

[38] A. Jameson, L. Martinelli, and N. A. Pierce. Optimum aerodynamic design using the Navier–Stokes equations. *Theoretical and Computational Fluid Dynamics*, 10(1–4):213–237, 1998. doi:10.1007/s001620050060.

[39] E. H. Johnson. Adjoint sensitivity analysis in MSC/NASTRAN. In *Proceedings of the MSC 1997 Aerospace Users' Conference*, number 2897, Los Angeles, CA, 1997. MSC Software.

[40] D. Juedes. A taxonomy of automatic differentiation tools. In A. Griewank and G. F. Corliss, editors, *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, pages 315–329. SIAM, Philadelphia, Penn., 1991.

[41] G. J. Kennedy and J. S. Hansen. The hybrid-adjoint method: A semi-analytic gradient evaluation technique applied to composite cure cycle optimization. *Optimization and Engineering*, 11:23–43, 2010. doi:10.1007/s11081-008-9068-9.

[42] G. J. Kennedy and J. R. R. A. Martins. Parallel solution methods for aerostructural analysis and design optimization. In *Proceedings of the 13th AIAA/ISSMO Multidisciplinary Analysis Optimization Conference*, Fort Worth, TX, Sept. 2010. AIAA 2010–9308.

[43] G. K. W. Kenway and J. R. R. A. Martins. Multi-point high-fidelity aerostructural optimization of a transport aircraft configuration. *Journal of Aircraft*, 2014. doi:10.2514/1.C032150. (In press).

[44] G. K. W. Kenway, G. J. Kennedy, and J. R. R. A. Martins. Scalable parallel approach for high-fidelity steady-state aeroelastic analysis and derivative computations. *AIAA Journal*, 2014. doi:10.2514/1.J052255. (In press).

[45] K.-L. Lai, J. Crassidis, Y. Cheng, and J. Kim. New complex-step derivative approximations with application to second-order kalman filtering. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, San Francisco, CA, August 2005. AIAA-2005-5944.

[46] A. B. Lambe and J. R. R. A. Martins. Extensions to the design structure matrix for the description of multidisciplinary design, analysis, and optimization processes. *Structural and Multidisciplinary Optimization*, 46:273–284, August 2012. doi:10.1007/s00158-012-0763-y.

[47] G. Lantoine, R. P. Russell, and T. Dargent. Using multicomplex variables for automatic computation of high-order derivatives. *ACM Trans. Math. Softw.*, 38(3):16:1–16:21, Apr. 2012. ISSN 0098-3500. doi:10.1145/2168773.2168774. URL `http://doi.acm.org/10.1145/2168773.2168774`.

[48] E. Lee and J. R. R. A. Martins. Structural topology optimization with design-dependent pressure loads. *Computer Methods in Applied Mechanics and Engineering*, 233–236:40–48, August 2012. ISSN 0045-7825. doi:10.1016/j.cma.2012.04.007.

[49] E. Lee, K. A. James, and J. R. R. A. Martins. Stress-constrained topology optimization with design-dependent loading. *Structural and Multidisciplinary Optimization*, 46:647–661, November 2012. doi:10.1007/s00158-012-0780-x.

[50] S. Liu and R. A. Canfield. Continuum sensitivity method for aeroelastic shape design problems. In *Proceedings of the 14th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Indianapolis, IN, Sep 2012. AIAA 2012-5480.

[51] T. Luzyanina and G. Bocharov. Critical Issues in the Numerical Treatment of the Parameter Estimation Problems in Immunology. *Journal of Computational Mathematics*, 30(1):59–79, Jan. 2012.

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

[53] J. N. Lyness and C. B. Moler. Numerical differentiation of analytic functions. *SIAM Journal on Numerical Analysis*, 4(2):202–210, 1967. ISSN 0036-1429 (print), 1095-7170 (electronic).

[54] Z. Lyu and J. R. R. A. Martins. Aerodynamic shape optimization of a blended-wing-body aircraft. In *Proceedings of the 51st AIAA Aerospace Sciences Meeting*, Grapevine, TX, Jan. 2013. doi:10.2514/6.2013-283. AIAA 2013-0283.

[55] C. A. Mader and J. R. R. A. Martins. Computation of aircraft stability derivatives using an automatic differentiation adjoint approach. *AIAA Journal*, 49(12):2737–2750, 2011. doi:10.2514/1.J051147.

[56] C. A. Mader and J. R. R. A. Martins. Derivatives for time-spectral computational fluid dynamics using an automatic differentiation adjoint. *AIAA Journal*, 50(12):2809–2819, December 2012. doi:10.2514/1.J051658.

[57] C. A. Mader, J. R. R. A. Martins, J. J. Alonso, and E. van der Weide. ADjoint: An approach for the rapid development of discrete adjoint solvers. *AIAA Journal*, 46(4):863–873, Apr. 2008. doi:10.2514/1.29123.

[58] K. Mani and D. J. Mavriplis. Unsteady discrete adjoint formulation for two-dimensional flow problems with deforming meshes. *AIAA Journal*, 46(6):1351–1364, 2008.

[59] C. J. Marriage and J. R. R. A. Martins. Reconfigurable semi-analytic sensitivity methods and MDO architectures within the πMDO framework. In *12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Victoria, BC, Sept. 2008. AIAA 2008-5956.

[60] J. R. R. A. Martins, J. J. Alonso, and J. J. Reuther. Aero-structural wing design optimization using high-fidelity sensitivity analysis. In H. Hölinger, editor, *Proceedings of the CEAS Conference on Multidisciplinary Aircraft Design and Optimization*, pages 211–226, Köln, Germany, June 2001.

[61] J. R. R. A. Martins, P. Sturdza, and J. J. Alonso. The complex-step derivative approximation. *ACM Transactions on Mathematical Software*, 29(3):245–262, 2003. doi:10.1145/838250.838251.

[62] J. R. R. A. Martins, J. J. Alonso, and J. J. Reuther. High-fidelity aerostructural design optimization of a supersonic business jet. *Journal of Aircraft*, 41(3):523–530, 2004. doi:10.2514/1.11478.

[63] J. R. R. A. Martins, J. J. Alonso, and J. J. Reuther. A coupled-adjoint sensitivity analysis method for high-fidelity aero-structural design. *Optimization and Engineering*, 6(1):33–62, Mar. 2005. doi:10.1023/B:OPTE.0000048536.47956.62.

[64] K. Maute, M. Nikbay, and C. Farhat. Coupled analytical sensitivity analysis and optimization of three-dimensional nonlinear aeroelastic systems. *AIAA Journal*, 39(11):2051–2061, 2001.

[65] S. Nadarajah and A. Jameson. A comparison of the continuous and discrete adjoint approach to automatic aerodynamic optimization. In *Proceedings of the 38th AIAA Aerospace Sciences Meeting and Exhibit*, Reno, NV, 2000. AIAA 2000-0667.

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

[67] M. Nemec, D. W. Zingg, and T. H. Pulliam. Multipoint and multi-objective aerodynamic shape optimization. *AIAA Journal*, 42(6):1057–1065, June 2004.

[68] J. C. Newman III, D. L. Whitfield, and W. K. Anderson. Step-size independent approach for multidisciplinary sensitivity analysis. *Journal of Aircraft*, 40(3):566–573, 2003. doi:10.2514/2.3131.

[69] M. Nykbay, L. Öncü, and A. Aysan. Multidisciplinary code coupling for analysis and optimization of aeroelastic systems. *Journal of Aircraft*, 46(6):1938–1944, Nov. 2009. ISSN 0021-8669. doi:10.2514/1.41491.

[70] V. Pascual and L. Hascoët. Extension of TAPENADE towards Fortran 95. In H. M. Bücker, G. Corliss, P. Hovland, U. Naumann, and B. Norris, editors, *Automatic Differentiation: Applications, Theory, and Tools*, Lecture Notes in Computational Science and Engineering. Springer, Berlin, Germany, 2005.

[71] R. E. Perez, P. W. Jansen, and J. R. R. A. Martins. pyOpt: a Python-based object-oriented framework for nonlinear constrained optimization. *Structural and Multidisciplinary Optimization*, 45(1):101–118, January 2012. doi:10.1007/s00158-011-0666-3.

[72] J. E. V. Peter and R. P. Dwight. Numerical sensitivity analysis for aerodynamic optimization: A survey of approaches. *Computers and Fluids*, 39:373–391, 2010. doi:10.1016/j.compfluid.2009.09.013.

[73] O. Pironneau. On optimum design in fluid mechanics. *Journal of Fluid Mechanics*, 64:97–110, 1974.

[74] N. M. K. Poon and J. R. R. A. Martins. An adaptive approach to constraint aggregation using adjoint sensitivity analysis. *Structural and Multidisciplinary Optimization*, 34(1):61–73, 2007. doi:10.1007/s00158-006-0061-7.

[75] J. D. Pryce and J. K. Reid. AD01, a Fortran 90 code for automatic differentiation. Report RAL-TR-1998-057, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire, OX11 OQX, U.K., 1998.

[76] J. J. Reuther, A. Jameson, J. J. Alonso, M. J. Rimlinger, and D. Saunders. Constrained multipoint aerodynamic shape optimization using an adjoint formulation and parallel computers, part 1. *Journal of Aircraft*, 36(1):51–60, 1999.

[77] J. J. Reuther, A. Jameson, J. J. Alonso, M. J. Rimlinger, and D. Saunders. Constrained multipoint aerodynamic shape optimization using an adjoint formulation and parallel computers, part 2. *Journal of Aircraft*, 36(1):61–74, 1999.

[78] A. Rhodin. IMAS — Integrated Modeling and Analysis System for the solution of optimal control problems. *Computer Physics Communications*, 107:21–38, 1997.

[79] M. S. Ridout. Statistical applications of the complex-step method of numerical differentiation. *The American Statistician*, 63(1):66–74, 2009.

[80] J. L. Roberts, A. D. Moy, T. D. van Ommen, M. A. J. Curran, A. P. Worby, I. D. Goodwin, and M. Inoue. Borehole temperatures reveal a changed energy budget at mill island, east antarctica, over recent decades. *The Cryosphere*, 7(1):263–273, 2013. doi:10.5194/tc-7-263-2013.

[81] M. Rumpfkeil and D. Zingg. The optimal control of unsteady flows with a discrete adjoint method. *Optimization and Engineering*, 11:5–22, 2010. ISSN 1389-4420. 10.1007/s11081-008-9035-5.

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

[83] A. P. Seyranian, E. Lund, and N. Olhoff. Multiple eigenvalues in structural optimization problems. *Structural and Multidisciplinary Optimization*, 8:207–227, 1994. ISSN 1615-147X. doi:10.1007/BF01742705.

[84] L. L. Sherman, A. C. Taylor III, L. L. Green, P. A. Newman, G. W. Hou, and V. M. Korivi. First- and second-order aerodynamic sensitivity derivatives via automatic differentiation with incremental iterative methods. *J. Comput. Phys.*, 129(2):307–331, 1996. ISSN 0021-9991. doi:10.1006/jcph.1996.0252.

[85] D. Shiriaev. ADOL–F automatic differentiation of Fortran codes. In M. Berz, C. H. Bischof, G. F. Corliss, and A. Griewank, editors, *Computational Differentiation: Techniques, Applications, and Tools*, pages 375–384. SIAM, Philadelphia, Penn., 1996.

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

[87] Z. Sirkes and E. Tziperman. Finite difference of adjoint or adjoint of finite difference? *Monthly Weather Review*, 125(12):3373–3378, Dec 1997.

[88] J. Sobieszczanski-Sobieski. Sensitivity of complex, internally coupled systems. *AIAA Journal*, 28(1): 153–160, 1990. doi:10.2514/3.10366.

[89] J. Sobieszczanski-Sobieski. Sensitivity analysis and multidisciplinary optimization for aircraft design: Recent advances and results. *Journal of Aircraft*, 27(12):993–1001, Dec. 1990. doi:10.2514/3.45973.

[90] J. Sobieszczanski-Sobieski. Higher order sensitivity analysis of complex, coupled systems. *AIAA Journal*, 28(4):756–758, 1990. Technical Note.

[91] W. Squire and G. Trapp. Using complex variables to estimate derivatives of real functions. *SIAM Review*, 40(1):110–112, 1998. ISSN 0036-1445 (print), 1095-7200 (electronic).

[92] C. W. Straka. Adf95: Tool for automatic differentiation of a fortran code designed for large numbers of independent variables. *Computer Physics Communications*, 168:123–139, 2005. doi:10.1016/j.cpc.2005.01.011.

[93] J. Utke. OpenAD: Algorithm implementation user guide. Technical Memorandum 274, Argonne National Laboratory, Argonne, IL, April 2004.

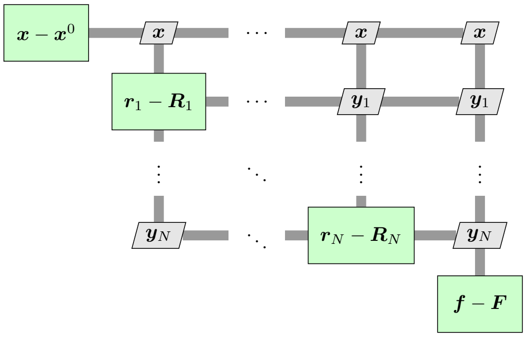
[94] J. Utke, U. Naumann, and A. Lyons. OpenAD/F: User Manual. Technical report, Argonne National Laboratory, Dec 2012.

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

[96] Q. Wang, P. Moin, and G. Iccarino. Minimal repetition dynamic checkpointing algorithm for unsteady adjoint calculation. *SIAM Journal on Scientific Computing*, 31(4):2549–2567, 2009. doi:10.1137/080727890.

[97] A. Zole and M. Karpel. Continuous gust response and sensitivity derivatives using state-space models. *Journal of Aircraft*, 31(5):1212–1214, 1994.

Variables and Constraints



$\boldsymbol{x}-\boldsymbol{x}^0$
$\boldsymbol{x}$
$\boldsymbol{r}_1-\boldsymbol{R}_1$
$\boldsymbol{y}_1$
$\boldsymbol{y}_N$
$\boldsymbol{r}_N-\boldsymbol{R}_N$
$\boldsymbol{f}-\boldsymbol{F}$

$$\boldsymbol{v}=\begin{bmatrix}\boldsymbol{x}\\ \boldsymbol{y}_1\\ \vdots\\ \boldsymbol{y}_N\\ \boldsymbol{f}\end{bmatrix}$$

$$\boldsymbol{C}(\boldsymbol{v})=\begin{bmatrix}\boldsymbol{x}-\boldsymbol{x}^0\\ \boldsymbol{r}_1-\boldsymbol{R}_1(\boldsymbol{x},\boldsymbol{y}_1,\ldots,\boldsymbol{y}_N)\\ \vdots\\ \boldsymbol{r}_N-\boldsymbol{R}_N(\boldsymbol{x},\boldsymbol{y}_1,\ldots,\boldsymbol{y}_N)\\ \boldsymbol{f}-\boldsymbol{F}(\boldsymbol{x},\boldsymbol{y}_1,\ldots,\boldsymbol{y}_N)\end{bmatrix}$$

Derivation

$$\left[\frac{\partial \boldsymbol{C}}{\partial \boldsymbol{v}}\right]\left[\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}\boldsymbol{c}}\right]=\boldsymbol{I}=\left[\frac{\partial \boldsymbol{C}}{\partial \boldsymbol{v}}\right]^T\left[\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}\boldsymbol{c}}\right]^T$$

$$\begin{bmatrix}\boldsymbol{I} & \boldsymbol{0} & \ldots & \boldsymbol{0} & \boldsymbol{0}\\ -\frac{\partial \boldsymbol{R}_1}{\partial \boldsymbol{x}} & -\frac{\partial \boldsymbol{R}_1}{\partial \boldsymbol{y}_1} & \ldots & -\frac{\partial \boldsymbol{R}_1}{\partial \boldsymbol{y}_N} & \boldsymbol{0}\\ \vdots & \vdots & \ddots & \vdots & \vdots\\ -\frac{\partial \boldsymbol{R}_N}{\partial \boldsymbol{x}} & -\frac{\partial \boldsymbol{R}_N}{\partial \boldsymbol{y}_1} & \ldots & -\frac{\partial \boldsymbol{R}_N}{\partial \boldsymbol{y}_N} & \boldsymbol{0}\\ -\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}} & -\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}_1} & \ldots & -\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}_N} & \boldsymbol{I}\end{bmatrix}\begin{bmatrix}\boldsymbol{I} & \boldsymbol{0} & \ldots & \boldsymbol{0} & \boldsymbol{0}\\ \frac{\mathrm{d}\boldsymbol{y}_1}{\mathrm{d}\boldsymbol{x}} & \frac{\mathrm{d}\boldsymbol{y}_1}{\mathrm{d}\boldsymbol{r}_1} & \ldots & \frac{\mathrm{d}\boldsymbol{y}_1}{\mathrm{d}\boldsymbol{r}_N} & \boldsymbol{0}\\ \vdots & \vdots & \ddots & \vdots & \vdots\\ \frac{\mathrm{d}\boldsymbol{y}_N}{\mathrm{d}\boldsymbol{x}} & \frac{\mathrm{d}\boldsymbol{y}_N}{\mathrm{d}\boldsymbol{r}_1} & \ldots & \frac{\mathrm{d}\boldsymbol{y}_N}{\mathrm{d}\boldsymbol{r}_N} & \boldsymbol{0}\\ \frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}} & \frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{r}_1} & \ldots & \frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{r}_N} & \boldsymbol{I}\end{bmatrix}=\boldsymbol{I}=\begin{bmatrix}\boldsymbol{I} & -\frac{\partial \boldsymbol{R}_1}{\partial \boldsymbol{x}}^T & \ldots & -\frac{\partial \boldsymbol{R}_N}{\partial \boldsymbol{x}}^T & -\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}}^T\\ \boldsymbol{0} & -\frac{\partial \boldsymbol{R}_1}{\partial \boldsymbol{y}_1}^T & \ldots & -\frac{\partial \boldsymbol{R}_N}{\partial \boldsymbol{y}_1}^T & -\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}_1}^T\\ \vdots & \vdots & \ddots & \vdots & \vdots\\ \boldsymbol{0} & -\frac{\partial \boldsymbol{R}_1}{\partial \boldsymbol{y}_N}^T & \ldots & -\frac{\partial \boldsymbol{R}_N}{\partial \boldsymbol{y}_N}^T & -\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}_N}^T\\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I}\end{bmatrix}\begin{bmatrix}\boldsymbol{I} & \frac{\mathrm{d}\boldsymbol{y}_1}{\mathrm{d}\boldsymbol{x}}^T & \ldots & \frac{\mathrm{d}\boldsymbol{y}_N}{\mathrm{d}\boldsymbol{x}}^T & \frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}}^T\\ \boldsymbol{0} & \frac{\mathrm{d}\boldsymbol{y}_1}{\mathrm{d}\boldsymbol{r}_1}^T & \ldots & \frac{\mathrm{d}\boldsymbol{y}_N}{\mathrm{d}\boldsymbol{r}_1}^T & \frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{r}_1}^T\\ \vdots & \vdots & \ddots & \vdots & \vdots\\ \boldsymbol{0} & \frac{\mathrm{d}\boldsymbol{y}_1}{\mathrm{d}\boldsymbol{r}_N}^T & \ldots & \frac{\mathrm{d}\boldsymbol{y}_N}{\mathrm{d}\boldsymbol{r}_N}^T & \frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{r}_N}^T\\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I}\end{bmatrix}$$

Coupled direct: residual form

$$\begin{bmatrix}\frac{\partial \boldsymbol{R}_1}{\partial \boldsymbol{y}_1} & \ldots & \frac{\partial \boldsymbol{R}_1}{\partial \boldsymbol{y}_N}\\ \vdots & \ddots & \vdots\\ \frac{\partial \boldsymbol{R}_N}{\partial \boldsymbol{y}_1} & \ldots & \frac{\partial \boldsymbol{R}_N}{\partial \boldsymbol{y}_N}\end{bmatrix}\begin{bmatrix}\frac{\mathrm{d}\boldsymbol{y}_1}{\mathrm{d}\boldsymbol{x}}\\ \vdots\\ \frac{\mathrm{d}\boldsymbol{y}_N}{\mathrm{d}\boldsymbol{x}}\end{bmatrix}=-\begin{bmatrix}\frac{\partial \boldsymbol{R}_1}{\partial \boldsymbol{x}}\\ \vdots\\ \frac{\partial \boldsymbol{R}_N}{\partial \boldsymbol{x}}\end{bmatrix}$$

$$\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}}=\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}}+\begin{bmatrix}\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}_1}\ldots\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}_N}\end{bmatrix}\begin{bmatrix}\frac{\mathrm{d}\boldsymbol{y}_1}{\mathrm{d}\boldsymbol{x}}\\ \vdots\\ \frac{\mathrm{d}\boldsymbol{y}_N}{\mathrm{d}\boldsymbol{x}}\end{bmatrix}$$

Coupled adjoint: residual form

$$\begin{bmatrix}\frac{\partial \boldsymbol{R}_1}{\partial \boldsymbol{y}_1}^T & \ldots & \frac{\partial \boldsymbol{R}_N}{\partial \boldsymbol{y}_1}^T\\ \vdots & \ddots & \vdots\\ \frac{\partial \boldsymbol{R}_1}{\partial \boldsymbol{y}_N}^T & \ldots & \frac{\partial \boldsymbol{R}_N}{\partial \boldsymbol{y}_N}^T\end{bmatrix}\begin{bmatrix}\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{r}_1}^T\\ \vdots\\ \frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{r}_N}^T\end{bmatrix}=-\begin{bmatrix}\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}_1}^T\\ \vdots\\ \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{y}_N}^T\end{bmatrix}$$

$$\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}}=\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}}+\begin{bmatrix}\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{r}_1}\ldots\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{r}_N}\end{bmatrix}\begin{bmatrix}\frac{\partial \boldsymbol{R}_1}{\partial \boldsymbol{x}}\\ \vdots\\ \frac{\partial \boldsymbol{R}_N}{\partial \boldsymbol{x}}\end{bmatrix}$$

Figure 12: Derivations of the residual form of the coupled derivative methods.

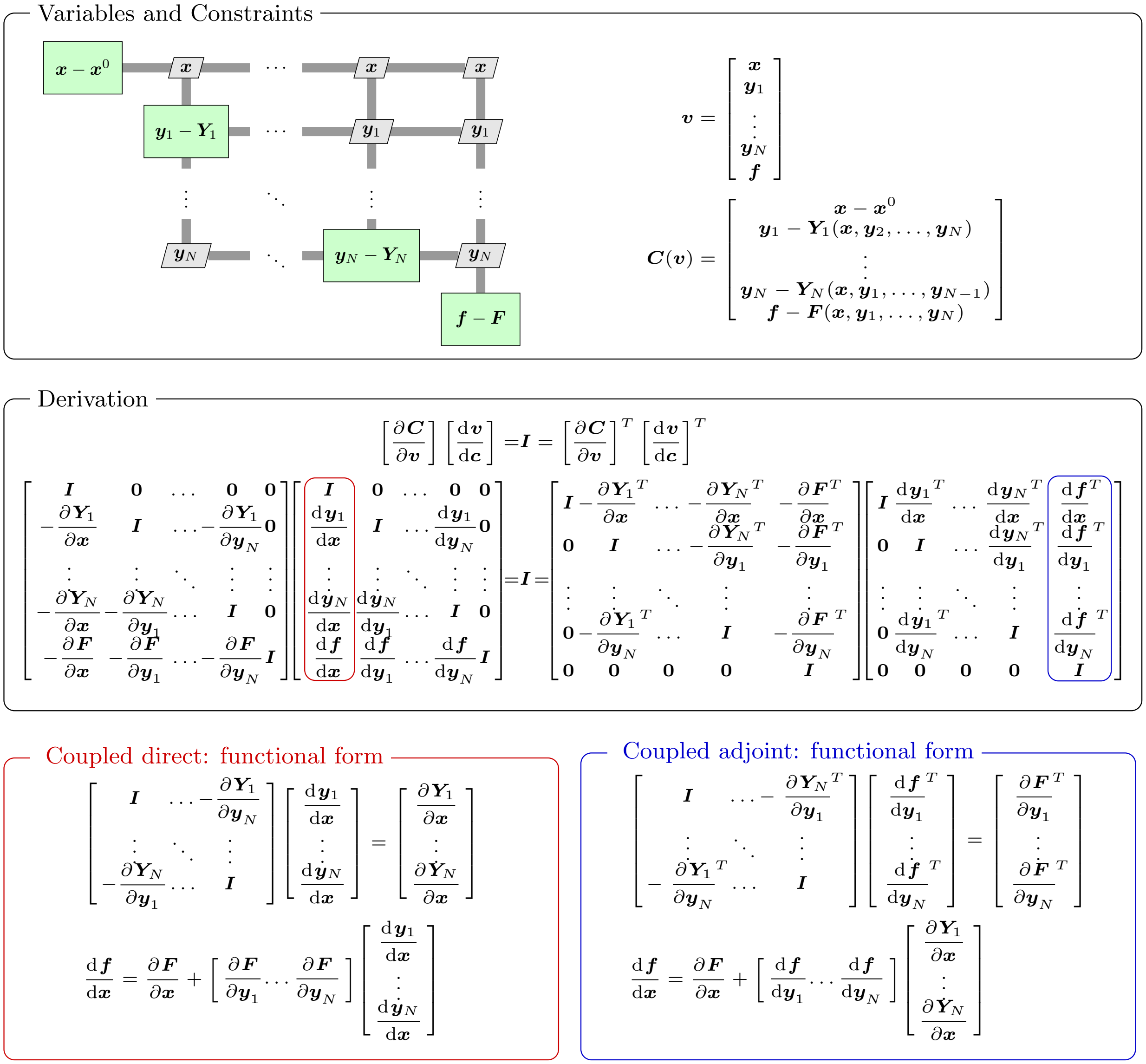

Variables and Constraints
$x - x^0$
x
$y_1 - Y_1$
y_1
y_N
$y_N - Y_N$
$f - F$
$$v = \begin{bmatrix} x \\ y_1 \\ \vdots \\ y_N \\ f \end{bmatrix}$$
$$C(v) = \begin{bmatrix} x - x^0 \\ y_1 - Y_1(x, y_2, \dots, y_N) \\ \vdots \\ y_N - Y_N(x, y_1, \dots, y_{N-1}) \\ f - F(x, y_1, \dots, y_N) \end{bmatrix}$$
Derivation
$$\left[\frac{\partial C}{\partial v}\right]\left[\frac{\mathrm{d}v}{\mathrm{d}c}\right] = I = \left[\frac{\partial C}{\partial v}\right]^T\left[\frac{\mathrm{d}v}{\mathrm{d}c}\right]^T$$
$$\begin{bmatrix} I & 0 & \dots & 0 & 0 \\ -\frac{\partial Y_1}{\partial x} & I & \dots & -\frac{\partial Y_1}{\partial y_N} & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ -\frac{\partial Y_N}{\partial x} & -\frac{\partial Y_N}{\partial y_1} & \dots & I & 0 \\ -\frac{\partial F}{\partial x} & -\frac{\partial F}{\partial y_1} & \dots & -\frac{\partial F}{\partial y_N} & I \end{bmatrix} \begin{bmatrix} I & 0 & \dots & 0 & 0 \\ \frac{\mathrm{d}y_1}{\mathrm{d}x} & I & \dots & \frac{\mathrm{d}y_1}{\mathrm{d}y_N} & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \frac{\mathrm{d}y_N}{\mathrm{d}x} & \frac{\mathrm{d}y_N}{\mathrm{d}y_1} & \dots & I & 0 \\ \frac{\mathrm{d}f}{\mathrm{d}x} & \frac{\mathrm{d}f}{\mathrm{d}y_1} & \dots & \frac{\mathrm{d}f}{\mathrm{d}y_N} & I \end{bmatrix} = I = \begin{bmatrix} I & -\frac{\partial Y_1}{\partial x}^T & \dots & -\frac{\partial Y_N}{\partial x}^T & -\frac{\partial F}{\partial x}^T \\ 0 & I & \dots & -\frac{\partial Y_N}{\partial y_1}^T & -\frac{\partial F}{\partial y_1}^T \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & -\frac{\partial Y_1}{\partial y_N}^T & \dots & I & -\frac{\partial F}{\partial y_N}^T \\ 0 & 0 & 0 & 0 & I \end{bmatrix} \begin{bmatrix} I & \frac{\mathrm{d}y_1}{\mathrm{d}x}^T & \dots & \frac{\mathrm{d}y_N}{\mathrm{d}x}^T & \frac{\mathrm{d}f}{\mathrm{d}x}^T \\ 0 & I & \dots & \frac{\mathrm{d}y_N}{\mathrm{d}y_1}^T & \frac{\mathrm{d}f}{\mathrm{d}y_1}^T \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \frac{\mathrm{d}y_1}{\mathrm{d}y_N}^T & \dots & I & \frac{\mathrm{d}f}{\mathrm{d}y_N}^T \\ 0 & 0 & 0 & 0 & I \end{bmatrix}$$
Coupled direct: functional form
$$\begin{bmatrix} I & \dots & -\frac{\partial Y_1}{\partial y_N} \\ \vdots & \ddots & \vdots \\ -\frac{\partial Y_N}{\partial y_1} & \dots & I \end{bmatrix} \begin{bmatrix} \frac{\mathrm{d}y_1}{\mathrm{d}x} \\ \vdots \\ \frac{\mathrm{d}y_N}{\mathrm{d}x} \end{bmatrix} = \begin{bmatrix} \frac{\partial Y_1}{\partial x} \\ \vdots \\ \frac{\partial Y_N}{\partial x} \end{bmatrix}$$
$$\frac{\mathrm{d}f}{\mathrm{d}x} = \frac{\partial F}{\partial x} + \left[\frac{\partial F}{\partial y_1} \dots \frac{\partial F}{\partial y_N}\right] \begin{bmatrix} \frac{\mathrm{d}y_1}{\mathrm{d}x} \\ \vdots \\ \frac{\mathrm{d}y_N}{\mathrm{d}x} \end{bmatrix}$$
Coupled adjoint: functional form
$$\begin{bmatrix} I & \dots & -\frac{\partial Y_N}{\partial y_1}^T \\ \vdots & \ddots & \vdots \\ -\frac{\partial Y_1}{\partial y_N}^T & \dots & I \end{bmatrix} \begin{bmatrix} \frac{\mathrm{d}f}{\mathrm{d}y_1}^T \\ \vdots \\ \frac{\mathrm{d}f}{\mathrm{d}y_N}^T \end{bmatrix} = \begin{bmatrix} \frac{\partial F}{\partial y_1}^T \\ \vdots \\ \frac{\partial F}{\partial y_N}^T \end{bmatrix}$$
$$\frac{\mathrm{d}f}{\mathrm{d}x} = \frac{\partial F}{\partial x} + \left[\frac{\mathrm{d}f}{\mathrm{d}y_1} \dots \frac{\mathrm{d}f}{\mathrm{d}y_N}\right] \begin{bmatrix} \frac{\partial Y_1}{\partial x} \\ \vdots \\ \frac{\partial Y_N}{\partial x} \end{bmatrix}$$

Figure 13: Derivation of the functional form of the coupled derivative methods.

Coupled direct: residual form

$$\begin{bmatrix} -\frac{\partial R_1}{\partial y_1} & -\frac{\partial R_1}{\partial y_2} \\ -\frac{\partial R_2}{\partial y_1} & -\frac{\partial R_2}{\partial y_2} \end{bmatrix} \begin{bmatrix} \frac{\mathrm{d}y_1}{\mathrm{d}x_1} & \frac{\mathrm{d}y_1}{\mathrm{d}x_2} \\ \frac{\mathrm{d}y_2}{\mathrm{d}x_1} & \frac{\mathrm{d}y_2}{\mathrm{d}x_2} \end{bmatrix} = \begin{bmatrix} \frac{\partial R_1}{\partial x_1} & \frac{\partial R_1}{\partial x_2} \\ \frac{\partial R_2}{\partial x_1} & \frac{\partial R_2}{\partial x_2} \end{bmatrix}$$

$$\begin{bmatrix} -x_1 & -2 \\ 1 & -x_2^2 \end{bmatrix} \begin{bmatrix} \frac{\mathrm{d}y_1}{\mathrm{d}x_1} & \frac{\mathrm{d}y_1}{\mathrm{d}x_2} \\ \frac{\mathrm{d}y_2}{\mathrm{d}x_1} & \frac{\mathrm{d}y_2}{\mathrm{d}x_2} \end{bmatrix} = \begin{bmatrix} y_1 - \cos x_1 & 0 \\ 0 & 2x_2y_2 \end{bmatrix}$$

$$\frac{\mathrm{d}f_1}{\mathrm{d}x_1} = \frac{\partial F_1}{\partial x_1} + \frac{\partial F_1}{\partial y_1}\frac{\mathrm{d}y_1}{\mathrm{d}x_1} + \frac{\partial F_1}{\partial y_2}\frac{\mathrm{d}y_2}{\mathrm{d}x_1}$$

$$\frac{\mathrm{d}f_1}{\mathrm{d}x_1} = 0 + 1\frac{\mathrm{d}y_1}{\mathrm{d}x_1} + 0\frac{\mathrm{d}y_2}{\mathrm{d}x_1}$$

Coupled adjoint: residual form

$$\begin{bmatrix} -\frac{\partial R_1}{\partial y_1} & -\frac{\partial R_2}{\partial y_1} \\ -\frac{\partial R_1}{\partial y_2} & -\frac{\partial R_2}{\partial y_2} \end{bmatrix} \begin{bmatrix} \frac{\mathrm{d}f_1}{\mathrm{d}r_1} & \frac{\mathrm{d}f_2}{\mathrm{d}r_1} \\ \frac{\mathrm{d}f_1}{\mathrm{d}r_2} & \frac{\mathrm{d}f_2}{\mathrm{d}r_2} \end{bmatrix} = \begin{bmatrix} \frac{\partial F_1}{\partial y_1} & \frac{\partial F_2}{\partial y_1} \\ \frac{\partial F_1}{\partial y_2} & \frac{\partial F_2}{\partial y_2} \end{bmatrix}$$

$$\begin{bmatrix} -x_1 & 1 \\ -2 & -x_2^2 \end{bmatrix} \begin{bmatrix} \frac{\mathrm{d}f_1}{\mathrm{d}r_1} & \frac{\mathrm{d}f_2}{\mathrm{d}r_1} \\ \frac{\mathrm{d}f_1}{\mathrm{d}r_2} & \frac{\mathrm{d}f_2}{\mathrm{d}r_2} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & \sin x_1 \end{bmatrix}$$

$$\frac{\mathrm{d}f_1}{\mathrm{d}x_1} = \frac{\partial F_1}{\partial x_1} + \frac{\mathrm{d}f_1}{\mathrm{d}r_1}\frac{\partial R_1}{\partial x_1} + \frac{\mathrm{d}f_1}{\mathrm{d}r_2}\frac{\partial R_2}{\partial x_1}$$

$$\frac{\mathrm{d}f_1}{\mathrm{d}x_1} = 0 + \frac{\mathrm{d}f_1}{\mathrm{d}r_1}(y_1 - \cos x_1) + \frac{\mathrm{d}f_1}{\mathrm{d}r_2}0$$

Coupled direct: functional form

$$\begin{bmatrix} 1 & -\frac{\partial Y_1}{\partial y_2} \\ -\frac{\partial Y_2}{\partial y_1} & 1 \end{bmatrix} \begin{bmatrix} \frac{\mathrm{d}y_1}{\mathrm{d}x_1} & \frac{\mathrm{d}y_1}{\mathrm{d}x_2} \\ \frac{\mathrm{d}y_2}{\mathrm{d}x_1} & \frac{\mathrm{d}y_2}{\mathrm{d}x_2} \end{bmatrix} = \begin{bmatrix} \frac{\partial Y_1}{\partial x_1} & \frac{\partial Y_1}{\partial x_2} \\ \frac{\partial Y_2}{\partial x_1} & \frac{\partial Y_2}{\partial x_2} \end{bmatrix}$$

$$\begin{bmatrix} 1 & \frac{2}{x_1} \\ -\frac{1}{x_2^2} & 1 \end{bmatrix} \begin{bmatrix} \frac{\mathrm{d}y_1}{\mathrm{d}x_1} & \frac{\mathrm{d}y_1}{\mathrm{d}x_2} \\ \frac{\mathrm{d}y_2}{\mathrm{d}x_1} & \frac{\mathrm{d}y_2}{\mathrm{d}x_2} \end{bmatrix} = \begin{bmatrix} \frac{2y_2}{x_1^2} + \frac{\cos x_1}{x_1} - \frac{\sin x_1}{x_1^2} & 0 \\ 0 & -\frac{2y_1}{x_2^3} \end{bmatrix}$$

$$\frac{\mathrm{d}f_1}{\mathrm{d}x_1} = \frac{\partial F_1}{\partial x_1} + \frac{\partial F_1}{\partial y_1}\frac{\mathrm{d}y_1}{\mathrm{d}x_1} + \frac{\partial F_1}{\partial y_2}\frac{\mathrm{d}y_2}{\mathrm{d}x_1}$$

$$\frac{\mathrm{d}f_1}{\mathrm{d}x_1} = 0 + 1\frac{\mathrm{d}y_1}{\mathrm{d}x_1} + 0\frac{\mathrm{d}y_2}{\mathrm{d}x_1}$$

Coupled adjoint: functional form

$$\begin{bmatrix} 1 & -\frac{\partial Y_2}{\partial y_1} \\ -\frac{\partial Y_1}{\partial y_2} & 1 \end{bmatrix} \begin{bmatrix} \frac{\mathrm{d}f_1}{\mathrm{d}y_1} & \frac{\mathrm{d}f_2}{\mathrm{d}y_1} \\ \frac{\mathrm{d}f_1}{\mathrm{d}y_2} & \frac{\mathrm{d}f_2}{\mathrm{d}y_2} \end{bmatrix} = \begin{bmatrix} \frac{\partial F_1}{\partial y_1} & \frac{\partial F_2}{\partial y_1} \\ \frac{\partial F_1}{\partial y_2} & \frac{\partial F_2}{\partial y_2} \end{bmatrix}$$

$$\begin{bmatrix} 1 & -\frac{1}{x_2^2} \\ \frac{2}{x_1} & 1 \end{bmatrix} \begin{bmatrix} \frac{\mathrm{d}f_1}{\mathrm{d}y_1} & \frac{\mathrm{d}f_2}{\mathrm{d}y_1} \\ \frac{\mathrm{d}f_1}{\mathrm{d}y_2} & \frac{\mathrm{d}f_2}{\mathrm{d}y_2} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & \sin x_1 \end{bmatrix}$$

$$\frac{\mathrm{d}f_1}{\mathrm{d}x_1} = \frac{\partial F_1}{\partial x_1} + \frac{\mathrm{d}f_1}{\mathrm{d}y_1}\frac{\partial Y_1}{\partial x_1} + \frac{\mathrm{d}f_1}{\mathrm{d}y_2}\frac{\partial Y_2}{\partial x_1}$$

$$\frac{\mathrm{d}f_1}{\mathrm{d}x_1} = 0 + \frac{\mathrm{d}f_1}{\mathrm{d}y_1}\left(\frac{2y_2}{x_1^2} + \frac{\cos x_1}{x_1} - \frac{\sin x_1}{x_1^2}\right) + \frac{\mathrm{d}f_1}{\mathrm{d}y_2}0$$

Coupled direct: hybrid form

$$\begin{bmatrix} -\frac{\partial R_1}{\partial y_1} & -\frac{\partial R_1}{\partial y_2} \\ -\frac{\partial Y_2}{\partial y_1} & 1 \end{bmatrix} \begin{bmatrix} \frac{\mathrm{d}y_1}{\mathrm{d}x_1} & \frac{\mathrm{d}y_1}{\mathrm{d}x_2} \\ \frac{\mathrm{d}y_2}{\mathrm{d}x_1} & \frac{\mathrm{d}y_2}{\mathrm{d}x_2} \end{bmatrix} = \begin{bmatrix} \frac{\partial R_1}{\partial x_1} & \frac{\partial R_1}{\partial x_2} \\ \frac{\partial Y_2}{\partial x_1} & \frac{\partial Y_2}{\partial x_2} \end{bmatrix}$$

$$\begin{bmatrix} -x_1 & -2 \\ -\frac{1}{x_2^2} & 1 \end{bmatrix} \begin{bmatrix} \frac{\mathrm{d}y_1}{\mathrm{d}x_1} & \frac{\mathrm{d}y_1}{\mathrm{d}x_2} \\ \frac{\mathrm{d}y_2}{\mathrm{d}x_1} & \frac{\mathrm{d}y_2}{\mathrm{d}x_2} \end{bmatrix} = \begin{bmatrix} y_1 - \cos x_1 & 0 \\ 0 & -\frac{2y_1}{x_2^3} \end{bmatrix}$$

$$\frac{\mathrm{d}f_1}{\mathrm{d}x_1} = \frac{\partial F_1}{\partial x_1} + \frac{\partial F_1}{\partial y_1}\frac{\mathrm{d}y_1}{\mathrm{d}x_1} + \frac{\partial F_1}{\partial y_2}\frac{\mathrm{d}y_2}{\mathrm{d}x_1}$$

$$\frac{\mathrm{d}f_1}{\mathrm{d}x_1} = 0 + 1\frac{\mathrm{d}y_1}{\mathrm{d}x_1} + 0\frac{\mathrm{d}y_2}{\mathrm{d}x_1}$$

Coupled adjoint: hybrid form

$$\begin{bmatrix} -\frac{\partial R_1}{\partial y_1} & -\frac{\partial Y_2}{\partial y_1} \\ -\frac{\partial R_1}{\partial y_2} & 1 \end{bmatrix} \begin{bmatrix} \frac{\mathrm{d}f_1}{\mathrm{d}r_1} & \frac{\mathrm{d}f_2}{\mathrm{d}r_1} \\ \frac{\mathrm{d}f_1}{\mathrm{d}y_2} & \frac{\mathrm{d}f_2}{\mathrm{d}y_2} \end{bmatrix} = \begin{bmatrix} \frac{\partial F_1}{\partial y_1} & \frac{\partial F_2}{\partial y_1} \\ \frac{\partial F_1}{\partial y_2} & \frac{\partial F_2}{\partial y_2} \end{bmatrix}$$

$$\begin{bmatrix} -x_1 & -\frac{1}{x_2^2} \\ -2 & 1 \end{bmatrix} \begin{bmatrix} \frac{\mathrm{d}f_1}{\mathrm{d}r_1} & \frac{\mathrm{d}f_2}{\mathrm{d}r_1} \\ \frac{\mathrm{d}f_1}{\mathrm{d}y_2} & \frac{\mathrm{d}f_2}{\mathrm{d}y_2} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & \sin x_1 \end{bmatrix}$$

$$\frac{\mathrm{d}f_1}{\mathrm{d}x_1} = \frac{\partial F_1}{\partial x_1} + \frac{\mathrm{d}f_1}{\mathrm{d}r_1}\frac{\partial R_1}{\partial x_1} + \frac{\mathrm{d}f_1}{\mathrm{d}y_2}\frac{\partial Y_2}{\partial x_1}$$

$$\frac{\mathrm{d}f_1}{\mathrm{d}x_1} = 0 + \frac{\mathrm{d}f_1}{\mathrm{d}r_1}(y_1 - \cos x_1) + \frac{\mathrm{d}f_1}{\mathrm{d}y_2}0$$

Figure 14: Application of the coupled direct and coupled adjoint methods in the residual, functional, and hybrid forms to the numerical example.