A Hitchhiker's Guide to Automatic Differentiation

Philipp H. W. Hoffmann

October 2, 2015

Preprint of an article to appear in:
Numerical Algorithms[*]

Abstract

This article provides an overview of some of the mathematical principles of Automatic Differentiation (AD). In particular, we summarise different descriptions of the Forward Mode of AD, like the matrix-vector product based approach, the idea of lifting functions to the algebra of dual numbers, the method of Taylor series expansion on dual numbers and the application of the push-forward operator, and explain why they all reduce to the same actual chain of computations. We further give a short mathematical description of some methods of higher-order Forward AD and, at the end of this paper, briefly describe the Reverse Mode of Automatic Differentiation.

Keywords: Automatic Differentiation, Forward AD, Reverse AD, Dual Numbers

AMS Subject Classification (2010): 65-02, 65K99

1 Introduction

Automatic Differentiation (short AD), also called Algorithmic or Computational Differentiation, is a method to evaluate derivatives of functions which differs significantly from the classical ways of computer-based differentiation through either approximative, numerical methods, or through symbolic differentiation, using computer algebra systems. While approximative methods (which are usually based on finite differences) are inherently prone to truncation and rounding errors and suffer from numerical instability, symbolic differentiation may (in certain cases) lead to significant long computation times. Automatic Differentiation suffers from none of these problems and is, in particular, well-suited for the differentiation of functions implemented as computer code. Furthermore,

[*]The final publication will be available at www.springerlink.com
Journal URL: `http://link.springer.com/journal/11075`

while Automatic Differentiation is also numerical differentiation, in the sense that it computes numerical values, it computes derivatives up to machine precision. That is, the only inaccuracies which occur are those which appear due to rounding errors in floating-point arithmetic or due to imprecise evaluations of elementary functions. For these reasons, AD has received significant interest from computer scientists and applied mathematicians, in the last decades.

The very first article on this procedure is probably due to Wengert [26] and appeared already in 1964. Two further major publications regarding AD were published by Rall in the 1980s [22], [23] and, since then, there has been a growing community of researcher interested in this topic.

So what is Automatic Differentiation? The answer to this question may be sought in one of the many publications on this topic, which usually provide a short introduction to the general theory. Furthermore, there are also excellent and comprehensive publications which describe the area as a whole (see for example Griewank [8] and Griewank and Walther [9]). However, an unfamiliar reader may find it nevertheless difficult to grasp the essence of Automatic Differentiation. The problem lies in the diversity with which the (actual simple) ideas can be described. While in [8] and [21, Section 2] the step-wise evaluation of a matrix-vector product is described as the basic procedure behind AD, in [14] Automatic Differentiation is defined via a certain multiplication on pairs (namely the multiplication which defines the algebra of *dual numbers*). Similarly, in [25] the lifting of a function to said dual numbers is presented as the core principle of AD, where in [20, Section 2], the evaluation of the Taylor series expansion of a function on dual numbers appears to be the main idea. Finally, Manzyuk [19] bases his description on the push-forward operator known from differential geometry and, again, gives a connection to functions on dual numbers. While the latter descriptions at least appear to be similar (although not identical), certainly the matrix-vector product based approach seems to differ from the remaining methods quite a lot. Of course, all the publications mentioned contain plenty of cross-references and each different description of AD has its specific purpose. However, for somebody unfamiliar with the theory, it may still be difficult to see why the described techniques are essentially all equivalent. This article hopes to clarify the situation.

We will in the following give short overviews of the distinct descriptions of AD[1] mentioned above and show, why they all are just different expressions of the same principle. It is clear that the purpose of this article is mainly educational and there is little intrinsically new in our elaborations. Indeed, in particular with regards to [8], we only give a extremely shorted and simplified version of the work in the original publication. Furthermore, there are actually at least two distinct versions, or modes, of AD. The so-called *Forward Mode* and the *Reverse Mode* (along with variants such as *Checkpoint Reverse Mode* [7]). The different descriptions mentioned above all refer to the Forward Mode only. We are, therefore, mainly concerned with Forward AD. We will discuss the standard Reverse Mode only in the preliminaries and in a section at the end of this paper.

[1]To be more precise, of the Forward Mode of AD.

In addition, we will mainly restrict ourselves to AD in its simplest form. Namely, Automatic Differentiation to compute (directional) first order derivatives, of a differentiable, multivariate function $f : X \to \mathbb{R}^m$, on an open set $X \subset \mathbb{R}^n$. We only briefly discuss the computation of higher-order partial derivatives in Section 8, referring mainly to the works of Berz [1] and Karczmarczuk [17]. There is also a rich literature on the computation of whole Hessians (see, for instance, [4] or [6]), however, we will not be concerned with this extension of AD in this article. The same holds for Nested Automatic Differentiation, which involves a kind of recursive calling of AD (see, for example, [25]). Again, we will not be concerned with this topic in this paper.

As mention above, Automatic Differentiation is often (and predominantly) used to differentiate computer programs, that is, implementations of mathematical functions as code. In the case of first order Forward AD, the mathematical principle used is usually the lifting of functions to dual numbers (see Figure 6 for an implementation example with test case). More information on this topic can, for example, be found in [2], [9] or (in particular considering higher-order differentiation) [17].

The notation we are using is basically standard. As mentioned above, the function we want to differentiate will be denoted by f and will be defined on an open set $X \subset \mathbb{R}^n$ (denoted by U in Section 8 to avoid confusion).[2] In particular, in this paper n always denotes the number of variables of f, while m denotes the dimension of its co-domain. In Sections 3 and 9, the notation x_i is reserved for variables of the function f, while other variables are denoted by v_i. The symbol c always denotes a fixed value (a constant). For real vectors, we use boldface letters like $\mathbf{x}$ or $\mathbf{c}$ (where the latter will be a constant vector). Furthermore, $\overrightarrow{\mathbf{x}}$ and $\overleftarrow{\mathbf{y}}$ will be (usually fixed) directional vectors or 1-row matrices, respectively. Entries of $\overrightarrow{\mathbf{x}}$ or $\overleftarrow{\mathbf{y}}$ will be denoted by x'_i or y'_i, respectively[3]. Finally, we denote all multiplications (of numbers, as well as matrix-vector multiplication) mostly by a simple dot. The symbol $*$ will be used sometimes when we want to emphasize that multiplication of numbers is a differentiable function on $\mathbb{R}^2$.

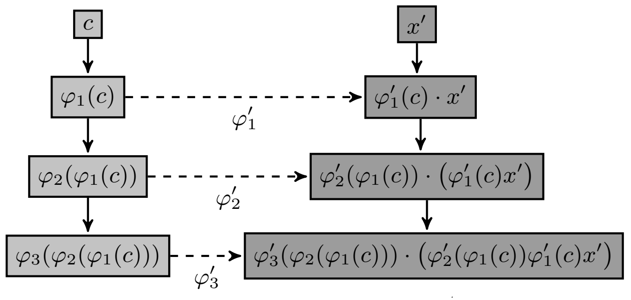
2 Preliminaries

2.1 The basic ideas of Automatic Differentiation

Before we start with the theory, let us demonstrate the ideas of AD in a very easy case: Let $f, \varphi_1, \varphi_2, \varphi_3 : \mathbb{R} \to \mathbb{R}$ be differentiable functions with $f = \varphi_3 \circ \varphi_2 \circ \varphi_1$. Let further $c, x', y' \in \mathbb{R}$ be real numbers. Assume we want to compute $f'(c) \cdot x'$ or $y' \cdot f'(c)$, respectively. (Of course, the distinction between multiplication from

[2]In principle, one can also consider functions of complex variables. Since the rules of real and complex differential calculus are the same, this does not lead to any changes in the theory.

[3]The notation x'_i for entries of $\overrightarrow{\mathbf{x}}$ is somewhat historical and based on the idea that, very often, x'_i may be considered as a derivative of either the identity function, or a constant function. For us, however, each $x'_i \in \mathbb{R}$ is simply a chosen real number. The same holds for the notation y'_i.



c
x'
$\varphi_1(c)$
$\varphi'_1(c)\cdot x'$
φ'_1
$\varphi_2(\varphi_1(c))$
$\varphi'_2(\varphi_1(c))\cdot(\varphi'_1(c)x')$
φ'_2
$\varphi_3(\varphi_2(\varphi_1(c)))$
$\varphi'_3(\varphi_2(\varphi_1(c)))\cdot(\varphi'_2(\varphi_1(c))\varphi'_1(c)x')$
φ'_3

Figure 1: Computational graph for the computation of $f'(c)\cdot x'$.

the left and from the right is motivated by the more general case of multivariate functions.)

By the chain rule,

$$f'(c)\cdot x' = \varphi'_3\left(\varphi_2\left(\varphi_1(c)\right)\right)\cdot\varphi'_2\left(\varphi_1(c)\right)\cdot\varphi'_1(c)\cdot x'$$

As one easily sees, the evaluation of $f'(c)\cdot x'$ can be achieved by computing successively the following pairs of real numbers:

$$\begin{gathered}(c,\ x')\\ (\varphi_1(c),\ \varphi'_1(c)\cdot x')\\ \left(\varphi_2\left(\varphi_1(c)\right),\ \varphi'_2\left(\varphi_1(c)\right)\cdot\varphi'_1(c)x'\right)\\ \left(\varphi_3\left(\varphi_2\left(\varphi_1(c)\right)\right),\ \varphi'_3\left(\varphi_2\left(\varphi_1(c)\right)\right)\cdot\varphi'_2\left(\varphi_1(c)\right)\varphi'_1(c)x'\right)\end{gathered}$$

and taking the second entry of the final pair. As we see, the first element of each pair appears as an argument of the functions φ_i, φ'_i in the following pair, while the second element appears as a factor (from the right) to the second element in the following pair.

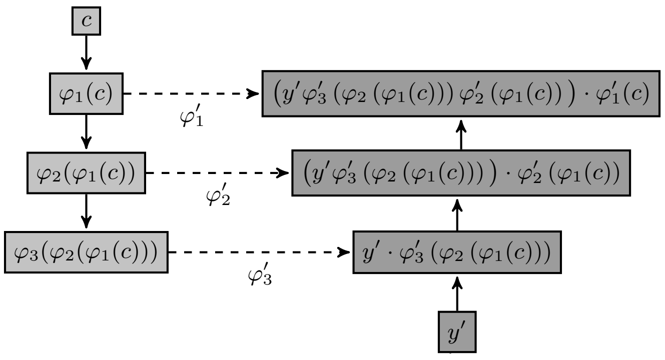
Regarding the computation of $y'\cdot f'(c)$, we have obviously

$$y'\cdot f'(c) = y'\cdot\varphi'_3\left(\varphi_2\left(\varphi_1(c)\right)\right)\cdot\varphi'_2\left(\varphi_1(c)\right)\cdot\varphi'_1(c).$$

The computation of this derivative can now be achieved by the computing the following two lists of real numbers:

$$\begin{array}{cc} c & y' \\ \varphi_1(c) & y'\cdot\varphi'_3\left(\varphi_2\left(\varphi_1(c)\right)\right) \\ \varphi_2(\varphi_1(c)) & y'\varphi'_3\left(\varphi_2\left(\varphi_1(c)\right)\right)\cdot\varphi'_2\left(\varphi_1(c)\right) \\ \varphi_3(\varphi_2(\varphi_1(c))) & y'\varphi'_3\left(\varphi_2\left(\varphi_1(c)\right)\right)\varphi'_2\left(\varphi_1(c)\right)\cdot\varphi'_1(c) \end{array}$$

and taking the last entry of the second list. Here, each entry (apart from y') in the second list consists of values of φ'_i evaluated at an element of the first list (note that the order is reversed) and the previous entry as a factor (from the left).



c
$\varphi_1(c)$
$\varphi_2(\varphi_1(c))$
$\varphi_3(\varphi_2(\varphi_1(c)))$
φ_1'
φ_2'
φ_3'
$(y'\varphi_3'(\varphi_2(\varphi_1(c)))\varphi_2'(\varphi_1(c)))\cdot\varphi_1'(c)$
$(y'\varphi_3'(\varphi_2(\varphi_1(c))))\cdot\varphi_2'(\varphi_1(c))$
$y'\cdot\varphi_3'(\varphi_2(\varphi_1(c)))$
y'

Figure 2: Computational graph for the computation of $y' \cdot f'(c)$.

In both examples, the computation of $\varphi_3(\varphi_2(\varphi_1(c)))$ is actually unnecessary to obtain the sought derivative. This value is, however, computed in all models we will consider in this article.

If now the functions $\varphi_1, \varphi_2, \varphi_3$ and their derivatives $\varphi_1', \varphi_2', \varphi_3'$ are implemented in the system, then the evaluation of values $\varphi_i(v_i), \varphi_i'(v_i)$ for some v_i means simply calling these functions/derivatives with suitable inputs. The computation of $f'(c) \cdot x'$ or $y' \cdot f'(c)$ then becomes nothing else than obtaining values $\varphi_i(v_i), \varphi_i'(v_i)$, performing a multiplication and passing the results on. That is, neither is some derivative evaluated symbolically, nor is some differential or difference quotient computed. In that sense, the derivative of f is computed 'automatically'.

2.2 The setting in general

As mentioned above, (First Order) Automatic Differentiation, in its simplest form, is concerned with the computation of derivatives of a differentiable function $f : X \to \mathbb{R}^m$, on an open set $X \subset \mathbb{R}^n$. The assumption made is that each $f_j : X \to \mathbb{R}$ in

$$f(x_1, ..., x_n) = \begin{pmatrix} f_1(x_1, ..., x_n) \\ \vdots \\ f_m(x_1, ..., x_n) \end{pmatrix}, \quad \text{for all } (x_1, ..., x_n) \in X,$$

consists (to be defined more precisely later) of several sufficiently smooth so-called *elementary* (or *elemental*) *functions* $\varphi_i : U_i \to \mathbb{R}$, defined on open sets $U_i \subset \mathbb{R}^{n_i}$, with $i \in I$ for some index set I. The set of elementary functions $\{\varphi_i \mid i \in I\}$ has to be given and can, in principle, consist of arbitrary functions as long as these are sufficiently often differentiable. However, certain functions are essential for computational means, including addition and multiplication[4],

[4]Here, we consider indeed addition and multiplication as differentiable functions $+ : \mathbb{R}^2 \to \mathbb{R}$ and $* : \mathbb{R}^2 \to \mathbb{R}$.

Domain	Functions	Remarks
$\mathbb{R}^2$	$+, *$	
$\mathbb{R}$	$x \mapsto -x,\ x \mapsto c$	$c \in \mathbb{R}$
$\mathbb{R} \setminus \{0\}$	$x \mapsto \frac{1}{x}$	
$\mathbb{R}$	$\exp, \sin, \cos$	
$(0, \infty)$	$\ln$	
$\mathbb{R} \setminus \{0\}$	$x \mapsto \lvert x \rvert$	
$\mathbb{R}$	$x \mapsto \lvert x \rvert^c$	$c > 1$
$\mathbb{R}^2 \setminus \{(0,0)\}$	$(x,y) \mapsto \lVert (x,y) \rVert_2 = \sqrt{x^2+y^2}$	
$\mathbb{R} \setminus \{0\}$	$\text{heav} := \left(x \mapsto \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases} \right)$	

Figure 3: Table of essential elementary functions according to Griewank and Walther [9]. The domains are chosen such that the functions are differentiable.

constant, trigonometric, exponential functions etc. Figure 3 shows a table of such a (minimal) list. A more comprehensive list can be found, for example, in [9, Table 2.3]. All elementary functions will be implemented in the system together with their gradients.

Automatic Differentiation now does not compute the actual mapping $\mathbf{x} \mapsto J_f(\mathbf{x})$, which maps a vector $\mathbf{x} \in X$ to the Jacobian $J_f(\mathbf{x})$ of f at $\mathbf{x}$. Instead, directional derivatives of f or left-hand products of row-vectors with its Jacobian at a fixed vector $\mathbf{c} \in X$ are determined. That is, given $\mathbf{c} \in X$ and $\overrightarrow{\mathbf{x}} \in \mathbb{R}^n$ or $\overleftarrow{\mathbf{y}} \in \mathbb{R}^{1 \times m}$, we determine either

$$J_f(\mathbf{c}) \cdot \overrightarrow{\mathbf{x}} \qquad \text{or} \qquad \overleftarrow{\mathbf{y}} \cdot J_f(\mathbf{c}).$$

(This is not a subtle difference, since, while $\mathbf{x} \mapsto J_f(\mathbf{x})$ is a matrix-valued function, $J_f(\mathbf{c}) \cdot \overrightarrow{\mathbf{x}}$ and $\overleftarrow{\mathbf{y}} \cdot J_f(\mathbf{c})$ are vectors or one-row matrices, respectively, in euclidean space.)

The computation of directional derivatives of $J_f(\mathbf{c})$ is referred to as the Forward Mode of AD, or Forward AD, while the computation of $\overleftarrow{\mathbf{y}} \cdot J_f(\mathbf{c})$ is referred to as the Reverse Mode of AD, or Reverse AD. We may give the following, informal descriptions:

Let $f : X \to \mathbb{R}^m$ consist of (not necessarily distinct!) elementary functions $\varphi_1, ..., \varphi_\mu$. Then

- *Forward Automatic Differentiation is the computation of* $J_f(\mathbf{c}) \cdot \overrightarrow{\mathbf{x}}$ *for fixed* $\mathbf{c} \in X$ *and* $\overrightarrow{\mathbf{x}} \in \mathbb{R}^n$ *through the successive computation of pairs of real numbers*

$$\left(\varphi_1(\mathbf{c_1}),\ \nabla\varphi_1(\mathbf{c_1}) \cdot \overrightarrow{\mathbf{x_1}}\right), ..., \left(\varphi_\mu(\mathbf{c}_\mu),\ \nabla\varphi_\mu(\mathbf{c}_\mu) \cdot \overrightarrow{\mathbf{x}_\mu}\right) \in \mathbb{R}^2$$

for suitable vectors $\mathbf{c_i} \in U_i, \overrightarrow{\mathbf{x_i}} \in \mathbb{R}^{n_i}$, $i = 1, ..., \mu$.

- *Reverse Automatic Differentiation is the computation of* $\overleftarrow{\mathbf{y}} \cdot J_f(\mathbf{c})$ *for fixed* $\mathbf{c} \in X$ *and* $\overleftarrow{\mathbf{y}} \in \mathbb{R}^{1\times m}$ *through the computation of the two lists of real numbers*

$$\varphi_1(\mathbf{c_1}), ..., \varphi_\mu(\mathbf{c}_\mu) \in \mathbb{R}$$

$$\textit{and} \quad v_1 \cdot \frac{\partial \varphi_1}{\partial v_k}(\mathbf{c_1}) + v_{1,k} \; , ..., \; v_\mu \cdot \frac{\partial \varphi_\mu}{\partial v_k}(\mathbf{c}_\mu) + v_{\mu,k} \; \in \mathbb{R}, \quad k = 1, ..., n_i,$$

for suitable vectors $\mathbf{c_i} \in U_i$ *and suitable numbers* $v_i, v_{i,k} \in \mathbb{R}$*,* $i = 1, ..., \mu$*.*

Of course, the vectors and numbers $\mathbf{c_i}, \overrightarrow{\mathbf{x_i}}, v_i, v_{i,k}$ are determined in a certain way; as is the order of the $\varphi_1, ..., \varphi_\mu$.

As mentioned before, the function f has to be constructed using elementary functions. Loosely speaking, we may say that f has to be a composition of elements of $\{\varphi_i \mid i \in I\}$. However, this is not quite correct from a strictly mathematically point of view. Since all elementary functions are real-valued, it is clear that a composition $\varphi_\mu \circ \cdots \circ \varphi_1$ can not be defined, as soon as one of the $\varphi_2, ..., \varphi_\mu$ is multivariate.[5] Admittedly, this is a rather technical and not really important issue, but, for completeness, we give the following inductive definition:

Definition 2.1. (i) We call a function $h : X \to \mathbb{R}$ on open $X \subset \mathbb{R}^n$ *automatically differentiable*, if

- $h \in \{\varphi_i \mid i \in I\}$ or
- there exist functions $h_k : X_k \to \mathbb{R}$ on open sets $X_k \subset \mathbb{R}^{n_k}$, $k = 1, ..., \ell$, such that for all $\mathbf{x} = (x_1, ..., x_n) \in X$, there exist $n_0 \geq 0$ many $x_{0,1}, ..., x_{0,n_0} \in \{x_1, ..., x_n\}$ and, for $k = 1, ..., \ell - 1$, $l = 1, ..., n_k$, there exist n_k many $x_{k,l} \in \{x_1, ..., x_n\} \cap X_k$, with

$$\begin{aligned} &h(\mathbf{x}) \\ &= h_\ell(x_{0,1}, ..., x_{0,n_0}, h_1(x_{1,1}, ..., x_{1,n_1}), ..., h_{\ell-1}(x_{\ell-1,1}, ..., x_{\ell-1,n_{\ell-1}})), \end{aligned}$$

and $h_\ell \in \{\varphi_i \mid i \in I\}$ and, for $k = 1, ..., \ell-1$, each h_k is automatically differentiable.

(ii) We call a function $f : X \to \mathbb{R}^m$ with $f(\mathbf{x}) = \begin{pmatrix} f_1(\mathbf{x}) \\ \vdots \\ f_m(\mathbf{x}) \end{pmatrix}$ for all $\mathbf{x} \in X$ *automatically differentiable*, if each $f_j : X \to \mathbb{R}$ is automatically differentiable.

Example 2.2. The function $h : \mathbb{R}^2 \to \mathbb{R}$ given by

$$h(x_1, x_2) = \sin(x_2) + 5 * \cos(x_1 * x_1).$$

is automatically differentiable.

[5]For instance, it is impossible to write $x \mapsto \exp(x) + \sin(x)$ as a composition of $\exp, \sin$ and $+$.

From now on, without necessarily stating it explicitly, we will always assume that our function $f : X \to \mathbb{R}^m$ is automatically differentiable in the sense of Definition 2.1.

One may, rightfully, ask why we use an inductive description of automatically differentiable functions, instead of just describing them as compositions of suitable multi-variable, multi-dimensional mappings. However, from a computational point of view, one should note that an automatically differentiable $f : X \to \mathbb{R}^m$ will usually be given in the form of Definition 2.1, such that expressing f as a composition requires additional work.

Nevertheless, expressing f as a composition is indeed the basic step in an elementary description of Automatic Differentiation, which we describe in the next Section. We will describe other (equivalent) approaches which work directly with functions of the form of Definition 2.1 in later sections.

3 Forward AD—An elementary approach

In this approach, the function f is described as a composition of multi-variate and multi-dimensional mappings. Differentiating this composition to obtain $J_f(\mathbf{c}) \cdot \overrightarrow{\mathbf{x}}$, for given $\mathbf{c} \in X \subset \mathbb{R}^n$ and $\overrightarrow{\mathbf{x}} \in \mathbb{R}^n$, leads, by the chain rule, to a product of matrices. This method has, for example, been described in [21, Section 2] and, comprehensively, in the works of Griewank [8] and Griewank and Walther [9]. We follow mainly the notation of [8].

The simple idea is to express f as a composition of the form

$$f = P_Y \circ \Phi_\mu \circ \cdots \circ \Phi_1 \circ P_X.$$

Here, $P_X : X \to H$ is the (linear) natural embedding of the domain $X \subset \mathbb{R}^n$ into the so-called *state space* $H := \mathbb{R}^{n+\mu}$, where μ is the total number of (not necessarily distinct) elementary functions φ_i of which f consists. Each $\Phi_i : H \to H$, referred to as an *elementary transition*, corresponds to exactly one such elementary function. The mapping $P_Y : H \to \mathbb{R}^m$ is some suitable linear projection of H down into $\mathbb{R}^m$.

Determining now $J_f(\mathbf{c}) \cdot \overrightarrow{\mathbf{x}}$ for fixed $\mathbf{c} \in X$ and fixed $\overrightarrow{\mathbf{x}} \in \mathbb{R}^n$ becomes, by the chain rule, the evaluation of the matrix-vector product

$$J_f(\mathbf{c}) \cdot \overrightarrow{\mathbf{x}} = P_Y \cdot \Phi'_{\mu,\mathbf{c}} \ \cdots \ \Phi'_{1,\mathbf{c}} \cdot P_X \cdot \overrightarrow{\mathbf{x}}, \tag{3.1}$$

where $\Phi'_{i,\mathbf{c}}$ denotes the Jacobian of Φ_i at $(\Phi_{i-1} \circ \cdots \circ \Phi_1 \circ P_X)(\mathbf{c})$.

The process is now performed in a particular ordered fashion, which we describe in the following.

The evaluation of f at some point $\mathbf{x} = (x_1, ..., x_n)$ can be described by a so-called *evaluation trace* $\mathbf{v}^{[0]} = \mathbf{v}^{[0]}(\mathbf{x}), ..., \mathbf{v}^{[\mu]} = \mathbf{v}^{[\mu]}(\mathbf{x})$, where each $\mathbf{v}^{[i]} \in H$ is a so-called *state vector*, representing the state of the evaluation after i steps. More precisely, we set

$$\mathbf{v}^{[0]} := P_X(x_1, ..., x_n) = (x_1, ..., x_n, 0, ..., 0) \ \text{ and } \ \mathbf{v}^{[i]} = \Phi_i(\mathbf{v}^{[i-1]}), \quad i = 1, ..., \mu.$$

The elementary transitions Φ_i are now given by imposing some suitable ordering on the μ elementary functions φ_k of which f consists, such that φ_i is the i-th elementary function with respect to this order, and by setting

$$\Phi_i \begin{pmatrix} v_1 \\ \vdots \\ v_{n+\mu} \end{pmatrix} = \begin{pmatrix} v_1 \\ \vdots \\ v_{n+i-1} \\ \varphi_i(v_{i_1}, ..., v_{i_{n_i}}) \\ v_{n+i+1} \\ \vdots \\ v_{n+\mu} \end{pmatrix}, \quad \text{for all} \begin{pmatrix} v_1 \\ \vdots \\ v_{n+\mu} \end{pmatrix} \in H,$$

and $v_{i_1}, ..., v_{i_{n_i}} \in \{v_1, ..., v_{n+i-1}\} \cap U_i$ (where $U_i \subset \mathbb{R}^{n_i}$ is the open domain of φ_i). Note that this is not a definition in the strict sense, since we neither specify the ordering of the φ_i, nor the arguments $v_{i_1}, ..., v_{i_{n_i}}$ of each φ_i. These will depend on the actual functions f and φ_i. (Compare the example below.)

Therefore, we have

$$\mathbf{v}^{[i]}(\mathbf{x}) = \Phi_i(\mathbf{v}^{[i-1]}(\mathbf{x})) = \begin{pmatrix} \mathbf{v}_1^{[i-1]}(\mathbf{x}) = x_1 \\ \vdots \\ \mathbf{v}_n^{[i-1]}(\mathbf{x}) = x_n \\ \vdots \\ \mathbf{v}_{n+i-1}^{[i-1]}(\mathbf{x}) \\ \varphi_i(v_{i_1}(\mathbf{x}), ..., v_{i_{n_i}}(\mathbf{x})) \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} \mathbf{v}_1^{[i-1]} = x_1 \\ \vdots \\ \mathbf{v}_n^{[i-1]} = x_n \\ \vdots \\ \mathbf{v}_{n+i-1}^{[i-1]} \\ \varphi_i(v_{i_1}, ..., v_{i_{n_i}}) \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

for $v_{i_1} = v_{i_1}(\mathbf{x}), ..., v_{i_{n_i}} = v_{i_{n_i}}(\mathbf{x}) \in \{\mathbf{v}_1^{[i-1]}, ..., \mathbf{v}_{n+i-1}^{[i-1]}\} \cap U_i$.

It is clear that, for the above to make sense, the ordering imposed on the elementary functions φ_k must have the property that all arguments in $\varphi_i(v_{i_1}, ..., v_{i_{n_i}})$ have already been evaluated, before φ_i is applied.

The definition of the projection $P_Y : H \to \mathbb{R}^m$ depends on the ordering imposed on the φ_k. If this ordering is such that we have

$$f_1(x_1, ..., x_n) = \mathbf{v}_{n+\mu-m}^{[\mu]}, ..., \; f_m(x_1, ..., x_n) = \mathbf{v}_{n+\mu}^{[\mu]},$$

we can obviously choose $P_Y(v_1, ..., v_{n+\mu}) = (v_{n+\mu-m}, ..., v_{n+\mu})$.

Example 3.1. The following is a trivial modification of an example taken from [8, page 332].

Consider the function $f : \mathbb{R}^2 \to \mathbb{R}^2$ given by

$$f(x_1, x_2) = \begin{pmatrix} \exp(x_1) * \sin(x_1 + x_2) \\ x_2 \end{pmatrix}.$$

Choose $H = \mathbb{R}^7$ and $f = P_Y \circ \Phi_5 \circ \Phi_4 \circ \Phi_3 \circ \Phi_2 \circ \Phi_1 \circ P_X$ with

$$P_X : \mathbb{R}^2 \to \mathbb{R}^7, \quad \text{with} \quad P_X(x_1, x_2) = (x_1, x_2, 0, 0, 0, 0, 0),$$

$\Phi_i : \mathbb{R}^7 \to \mathbb{R}^7$, $i = 1, ..., 5$, with

$$\begin{aligned}
\Phi_1\,(v_1, v_2, v_3, v_4, v_5, v_6, v_7) &= (v_1, v_2, \exp(v_1), v_4, v_5, v_6, v_7),\\
\Phi_2\,(v_1, v_2, v_3, v_4, v_5, v_6, v_7) &= (v_1, v_2, v_3, v_1 + v_2, v_5, v_6, v_7),\\
\Phi_3\,(v_1, v_2, v_3, v_4, v_5, v_6, v_7) &= (v_1, v_2, v_3, v_4, \sin(v_4), v_6, v_7),\\
\Phi_4\,(v_1, v_2, v_3, v_4, v_5, v_6, v_7) &= (v_1, v_2, v_3, v_4, v_5, v_3 * v_5, v_7),\\
\Phi_5\,(v_1, v_2, v_3, v_4, v_5, v_6, v_7) &= (v_1, v_2, v_3, v_4, v_5, v_6, v_2)
\end{aligned}$$

and

$$P_Y : \mathbb{R}^7 \to \mathbb{R}^2, \quad \text{with} \quad P_Y(v_1, v_2, v_3, v_4, v_5, v_6, v_7) = (v_6, v_7).$$

Analogously to the evaluation of $f(\mathbf{x})$, the evaluation of the matrix-vector product (3.1) for some $\mathbf{c} \in \mathbb{R}^n$ and some $\overrightarrow{\mathbf{x}} = (x'_1, ..., x'_n) \in \mathbb{R}^n$ can be expressed as an evaluation trace $\mathbf{v}'^{[0]} = \mathbf{v}'^{[0]}(\mathbf{c}, \overrightarrow{\mathbf{x}}), ..., \mathbf{v}'^{[\mu]} = \mathbf{v}'^{[\mu]}(\mathbf{c}, \overrightarrow{\mathbf{x}})$, where

$$\mathbf{v}'^{[0]} := P_X \cdot \overrightarrow{\mathbf{x}} = (x'_1, ..., x'_m, 0, ..., 0) \quad \text{and} \quad \mathbf{v}'^{[i]} := \Phi'_{i,\mathbf{c}} \cdot \mathbf{v}'^{[i-1]}, \quad i = 1, ..., \mu.$$

By the nature of the elementary transformations Φ_i, each Jacobian
$\Phi'_{i,\mathbf{c}} := J_{\Phi_i}(\mathbf{v}^{[i-1]}(\mathbf{c}))$ will be of the form

$$\Phi'_{i,\mathbf{c}} = \begin{pmatrix}
1 & \cdots & 0 & 0 & \cdots & 0\\
\vdots & \ddots & \vdots & \vdots & \cdots & \vdots\\
0 & \cdots & 1 & 0 & \cdots & 0\\
\frac{\partial \varphi_i}{\partial v_1}(\cdots) & \cdots & \cdots & \cdots & \cdots & \frac{\partial \varphi_i}{\partial v_{n+\mu}}(\cdots)\\
0 & \cdots & 0 & 1 & \cdots & 0\\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots\\
0 & \cdots & 0 & 0 & \cdots & 1
\end{pmatrix} \leftarrow \ (n+i)\text{-th row,} \tag{3.2}$$

where $\frac{\partial \varphi_i}{\partial v_k}(\cdots) = \frac{\partial \varphi_i}{\partial v_k}(v_{i_1}(\mathbf{c}), ..., v_{i_{n_i}}(\mathbf{c}))$ is interpreted as 0 if φ_i does not depend on v_k.

Thus, each $\mathbf{v}'^{[i]}$ will be of the form

$$\mathbf{v}'^{[i]} = \begin{pmatrix} \mathbf{v}'^{[i-1]}_1 = x'_1 \\ \vdots \\ \mathbf{v}'^{[i-1]}_n = x'_n \\ \vdots \\ \mathbf{v}'^{[i-1]}_{n+i-1} \\ \nabla\varphi_i(v_{i_1}, ..., v_{i_{n_i}}) \cdot \begin{pmatrix} v'_{i_1} \\ \vdots \\ v'_{i_{n_i}} \end{pmatrix} \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

for $v'_{i_1} = v'_{i_1}(\mathbf{c}, \overset{\rightharpoonup}{\mathbf{x}}), ..., v'_{i_{n_i}} = v'_{i_{n_i}}(\mathbf{c}, \overset{\rightharpoonup}{\mathbf{x}}) \in \{\mathbf{v}'^{[i-1]}_1, ..., \mathbf{v}'^{[i-1]}_{n+i-1}\}$, where the $v'_{i_1}, ..., v'_{i_{n_i}}$ correspond exactly to the $v_{i_1}, ..., v_{i_{n_i}}$. That is, if $v_{i_j} = \mathbf{v}^{[i-1]}_l(\mathbf{c})$, then $v'_{i_j} = \mathbf{v}'^{[i-1]}_l(\mathbf{c}, \overset{\rightharpoonup}{\mathbf{x}})$.

The directional derivative of f at $\mathbf{c}$ in direction of $\overset{\rightharpoonup}{\mathbf{x}}$ is then simply

$$J_f(\mathbf{c}) \cdot \overset{\rightharpoonup}{\mathbf{x}} = P_Y \cdot \mathbf{v}'^{[\mu]}.$$

Example 3.2. Let $c_1, c_2, x'_1, x'_2 \in \mathbb{R}$. The computation of $J_f((c_1, c_2)) \cdot \begin{pmatrix} x'_1 \\ x'_2 \end{pmatrix}$ with $f : \mathbb{R}^2 \to \mathbb{R}^2$ given by

$$f(x_1, x_2) = \begin{pmatrix} \exp(x_1) * \sin(x_1 + x_2) \\ x_2 \end{pmatrix}$$

has five *evaluation trace pairs* $[\mathbf{v}^{[0]}, \mathbf{v}'^{[0]}], ..., [\mathbf{v}^{[5]}, \mathbf{v}'^{[5]}]$, where

$$\mathbf{v}^{[0]} = (c_1, c_2, 0, 0, 0, 0, 0) \text{ and } \mathbf{v}'^{[0]} = (x'_1, x'_2, 0, 0, 0, 0, 0)$$

and

$$\mathbf{v}^{[5]} = \begin{pmatrix} c_1 \\ c_2 \\ \exp(c_1) \\ c_1 + c_2 \\ \sin(c_1 + c_2) \\ \exp(c_1) * \sin(c_1 + c_2) \\ c_2 \end{pmatrix},$$

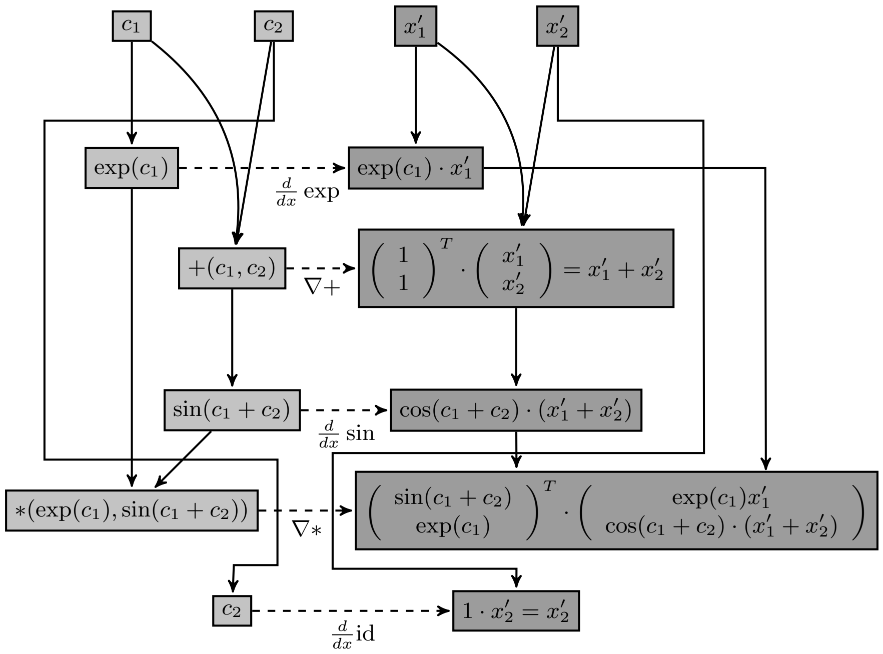

c_1
c_2
x'_1
x'_2
$\exp(c_1)$
$\exp(c_1)\cdot x'_1$
$\frac{d}{dx}\exp$
$+(c_1,c_2)$
$\begin{pmatrix}1\\1\end{pmatrix}^T\cdot\begin{pmatrix}x'_1\\x'_2\end{pmatrix}=x'_1+x'_2$
$\nabla+$
$\sin(c_1+c_2)$
$\cos(c_1+c_2)\cdot(x'_1+x'_2)$
$\frac{d}{dx}\sin$
$*(\exp(c_1),\sin(c_1+c_2))$
$\begin{pmatrix}\sin(c_1+c_2)\\\exp(c_1)\end{pmatrix}^T\cdot\begin{pmatrix}\exp(c_1)x'_1\\\cos(c_1+c_2)\cdot(x'_1+x'_2)\end{pmatrix}$
$\nabla*$
c_2
$1\cdot x'_2=x'_2$
$\frac{d}{dx}\mathrm{id}$

Figure 4: Computational graph for Example 3.2 with elements of $\mathbf{v}^{[5]}$ in blue and of $\mathbf{v}'^{[5]}$ in red.

$$\overline{\mathbf{v}'}^{[5]} = \begin{pmatrix} x'_1 \\ x'_2 \\ \exp(c_1)x'_1 \\ x'_1 + x'_2 \\ \cos(c_1+c_2)(x'_1+x'_2) \\ \sin(c_1+c_2)\exp(c_1)x'_1 + \exp(c_1)\cos(c_1+c_2)(x'_1+x'_2) \\ x'_2 \end{pmatrix}.$$

Then $J_f((c_1,c_2)) \cdot \begin{pmatrix} x'_1 \\ x'_2 \end{pmatrix} = P_Y \cdot \mathbf{v}'^{[5]}$, which is

$$\begin{pmatrix} \big(\sin(c_1+c_2)\exp(c_1) + \exp(c_1)\cos(c_1+c_2)\big)x'_1 + \exp(c_1)\cos(c_1+c_2)x'_2 \\ x'_2 \end{pmatrix}.$$

Note that in the evaluation process, given the Φ_i, each pair $[\mathbf{v}^{[i]}, \mathbf{v}'^{[i]}]$ depends only on the previous pair $[\mathbf{v}^{[i-1]}, \mathbf{v}'^{[i-1]}]$ and the given vectors $\mathbf{c}$, $\overrightarrow{\mathbf{x}}$. (Since $\mathbf{v}^{[i]} = \Phi_i(\mathbf{v}^{[i-1]})$ and $\mathbf{v}'^{[i]} = J_{\Phi_i}(\mathbf{v}^{[i-1]}(\mathbf{c})) \cdot \mathbf{v}'^{[i-1]}$.) Therefore, in an implementation, one can actually overwrite $[\mathbf{v}^{[i-1]}, \mathbf{v}'^{[i-1]}]$ by $[\mathbf{v}^{[i]}, \mathbf{v}'^{[i]}]$ in each step.

Note further that the $(n+i)$-th entry in each pair $[\mathbf{v}^{[i]}, \mathbf{v}'^{[i]}]$ is of the form

$$\left(\varphi_i(v_{i_1}, ..., v_{i_{n_i}}), \nabla\varphi_i(v_{i_1}, ..., v_{i_{n_i}}) \cdot \begin{pmatrix} v'_{i_1} \\ \vdots \\ v'_{i_{n_i}} \end{pmatrix}\right) \in \mathbb{R}^2, \tag{3.3}$$

i.e. consisting of a value of φ_i and a directional derivative of this elementary function. Since the previous $n+i-1$ entries are identical to the first $n+i-1$ entries of $[\mathbf{v}^{[i-1]}, \mathbf{v}'^{[i-1]}]$, the computation of $[\mathbf{v}^{[i]}, \mathbf{v}'^{[i]}]$ is effectively the computation of (3.3).

We summarize the discussion of this Section:

Theorem 3.3. *By the above, given $\mathbf{c} \in X \subset \mathbb{R}^n$ and $\vec{\mathbf{x}} \in \mathbb{R}^n$, the evaluation of $J_f(\mathbf{c})\cdot\vec{\mathbf{x}}$ of an automatically differentiable function $f : X \to \mathbb{R}^m$ can be achieved by computing the evaluation trace pairs $[\mathbf{v}^{[i]}, \mathbf{v}'^{[i]}]$. This process is equivalent to the computation of the pairs* (3.3).

The following Section is concerned with a method which uses this last fact directly from the start. The approach about to be described also provides a better understanding on how an Automatic Differentiation system could actually be implemented. A question which may not be quite clear from the discussion so far.

4 Forward AD—An approach using Dual Numbers

Many descriptions and implementation of Forward AD actually use a slightly different approach than the elementary one that we have just described. Instead of expressing the function whose derivative one wants to compute as a composition, the main idea in this 'alternative' approach[6] is to lift this function (and all elementary functions) to (a subset of) the algebra of *dual numbers $\mathcal{D}$*. This method has, for example, been described in [14], [20] and [23].

Dual numbers, introduced by Clifford [3], are defined as $\mathcal{D} := (\mathbb{R}^2, +, \cdot)$, where addition is defined component-wise, as usual, and multiplication is defined as

$$(x_1, y_1) \cdot (x_2, y_2) := (x_1x_2, x_1y_2 + y_1x_2), \quad \forall\ (x_1, y_1), (x_2, y_2) \in \mathbb{R}^2.$$

It is easy to verify that $\mathcal{D}$ with these operations is an associative and commutative algebra over $\mathbb{R}$ with multiplicative unit $(1, 0)$ and that the element $\varepsilon := (0, 1)$ is nilpotent of order two. [7]

[6] Indeed, we will see at the end of this Section, that Forward AD using dual numbers is completely equivalent to the method of expressing f as $P_Y \circ \Phi_\mu \circ \cdots \circ \Phi_1 \circ P_X$.

[7] ε is sometimes referred to as an infinitesimal in the literature. The correct interpretation of this is probably that one can replace dual numbers by elements from non-standard analysis in the context of AD. However, this approach is actually unnecessary and, given the complexity of non-standard analysis, we will not consider it here.

Analogously to a complex number, we write a dual number $z = (x, y)$ as $z = x + y\varepsilon$, where we identify each $x \in \mathbb{R}$ with $(x, 0)$. We will further use the notation (x, x') instead of (x, y), i.e. we write $z = x + x'\varepsilon$. The x' in this representation will be referred to as the *dual part* of z.

We now define an extension of a differentiable, real-valued function $h : X \to \mathbb{R}$, defined on open $X \subset \mathbb{R}^n$, to a function $\widehat{h} : \mathcal{D}^n \supset X \times \mathbb{R}^n \to \mathcal{D}$ defined on a subset of the dual numbers, by setting

$$\widehat{h}(x_1 + x_1'\varepsilon, ..., x_n + x_n'\varepsilon) := h(x_1, ..., x_n) + \left(\nabla h(x_1, ..., x_n) \cdot \begin{pmatrix} x_1' \\ \vdots \\ x_n' \end{pmatrix} \right) \cdot \varepsilon. \tag{4.1}$$

This definition easily extends to differentiable functions $f : X \to \mathbb{R}^m$, where $\widehat{f} : \mathcal{D}^n \supset X \times \mathbb{R}^n \to \mathcal{D}^m$ is defined via

$$\begin{aligned} \widehat{f}(x_1 + x_1'\varepsilon, ..., x_n + x_n'\varepsilon) &:= \begin{pmatrix} \widehat{f}_1(x_1 + x_1'\varepsilon, ..., x_n + x_n'\varepsilon) \\ \vdots \\ \widehat{f}_m(x_1 + x_1'\varepsilon, ..., x_n + x_n'\varepsilon) \end{pmatrix} \qquad (4.2) \\ &= f(x_1, ..., x_n) + \left(J_f(x_1, ..., x_n) \cdot \begin{pmatrix} x_1' \\ \vdots \\ x_n' \end{pmatrix} \right) \varepsilon. \end{aligned}$$

The following statement shows that definition (4.1) makes sense. I.e., that it is compatible with the natural extension of functions which are defined via usual arithmetic, i.e. polynomials, and analytic functions. That is:

Proposition 4.1. *Definition* (4.1) *is compatible with the "natural" extension of*

(i) real-valued constant functions

(ii) projections of the form $(x_1, .., x_n) \mapsto x_k$,

(iii) the arithmetic operations $+, * : \mathbb{R}^2 \to \mathbb{R}$ *and* $/ : \{(x_1, x_2) \in \mathbb{R}^2 \mid x_2 \neq 0\} \to \mathbb{R}$, *with*

$$+(x_1, x_2) := x_1 + x_2, \quad *(x_1, x_2) := x_1 \cdot x_2, \quad /(x_1, x_2) := \frac{x_1}{x_2} \ .$$

(iv) (multivariate) polynomials and rational functions

(v) (multivariate) real analytic functions

to subsets of $\mathcal{D}^2$ *or* $\mathcal{D}^n$, *respectively.*

Proof. (i) and (ii) follow easily from the definition.

(iii): We have

$$\begin{aligned}
\widehat{+}(x_1 + x_1'\varepsilon, x_2 + x_2'\varepsilon) &= +\,(x_1, x_2) + \left(\nabla + (x_1, x_2) \cdot \begin{pmatrix} x_1' \\ x_2' \end{pmatrix}\right)\varepsilon \\
&= (x_1 + x_2) + \left(\begin{pmatrix} 1 \\ 1 \end{pmatrix}^T \cdot \begin{pmatrix} x_1' \\ x_2' \end{pmatrix}\right)\varepsilon \\
&= (x_1 + x_2) + (x_1' + x_2')\varepsilon \\
&= (x_1 + x_1'\varepsilon) + (x_2 + x_2'\varepsilon)
\end{aligned}$$

and, since $\varepsilon^2 = 0$,

$$\begin{aligned}
\widehat{*}(x_1 + x_1'\varepsilon, x_2 + x_2'\varepsilon) &= *(x_1, x_2) + \left(\nabla * (x_1, x_2) \cdot \begin{pmatrix} x_1' \\ x_2' \end{pmatrix}\right)\varepsilon \\
&= (x_1 \cdot x_2) + \left(\begin{pmatrix} x_2 \\ x_1 \end{pmatrix}^T \cdot \begin{pmatrix} x_1' \\ x_2' \end{pmatrix}\right)\varepsilon \\
&= (x_1 \cdot x_2) + (x_2 x_1' + x_1 x_2')\varepsilon \\
&= (x_1 + x_1'\varepsilon) \cdot (x_2 + x_2'\varepsilon).
\end{aligned}$$

Finally, considering division, it is easy to see that the multiplicative inverse of a dual numbers $x + x'\varepsilon$ is defined if, and only if, $x \neq 0$ and given by $\frac{1}{x} + \left(-\frac{x'}{x^2}\right)\varepsilon$.

Then, for $x_2 \neq 0$, since $\varepsilon^2 = 0$,

$$\begin{aligned}
\widehat{/}(x_1 + x_1'\varepsilon, x_2 + x_2'\varepsilon) &= /(x_1, x_2) + \left(\nabla/(x_1, x_2) \cdot \begin{pmatrix} x_1' \\ x_2' \end{pmatrix}\right)\varepsilon \\
&= \frac{x_1}{x_2} + \left(\begin{pmatrix} \frac{1}{x_2} \\ -\frac{x_1}{x_2^2} \end{pmatrix}^T \cdot \begin{pmatrix} x_1' \\ x_2' \end{pmatrix}\right)\varepsilon \\
&= \frac{x_1}{x_2} + \left(\frac{x_1'}{x_2} - \frac{x_1 \cdot x_2'}{x_2^2}\right)\varepsilon \\
&= (x_1 + x_1'\varepsilon) \cdot \left(\frac{1}{x_2} + \left(-\frac{x_2'}{x_2^2}\right)\varepsilon\right) \\
&= \frac{x_1 + x_1'\varepsilon}{x_2 + x_2'\varepsilon}\;.
\end{aligned}$$

(iv): This will follow from Proposition 4.2 in connection with (iii).

(v): We will recall the definition of multi-variate Taylor series in Section 6. For the moment, let $T_k(h; \mathbf{c})$ denote the k-th degree (multi-variate) Taylor polynomial of $h : X \to \mathbb{R}$ about $\mathbf{c} \in X$. Since h is real analytic, we have

$$T_k(h; \mathbf{c})(x_1, ..., x_n) \to h(x_1, ..., x_n) \quad (k \to \infty)$$

for all $(x_1, ..., x_n) \in V$, where V is an open neighbourhood of $\mathbf{c}$. It is well-known, that then

$$\frac{\partial}{\partial x_j} T_k(h; \mathbf{c})(x_1, ..., x_n) \to \frac{\partial}{\partial x_j} h(x_1, ..., x_n) \quad (k \to \infty)$$

on V (see for example [13, Chapter II.1]). Since addition and multiplication are continuous, then also

$$\left(\nabla T_k(h;\mathbf{c})(x_1,...,x_n)\cdot\overrightarrow{\mathbf{x}}\right)\to\left(\nabla h(x_1,...,x_n)\cdot\overrightarrow{\mathbf{x}}\right)\quad(k\to\infty)$$

on V, for any fixed $\overrightarrow{\mathbf{x}}=(x_1',...,x_n')\in\mathbb{R}^n$. Consequently,

$$\widehat{T_k}(h;\mathbf{c})(x_1+x_1'\varepsilon,...,x_n+x_n'\varepsilon)\to\widehat{h}(x_1+x_1'\varepsilon,...,x_n+x_n'\varepsilon)\quad(k\to\infty),$$

for all $(x_1+x_1'\varepsilon,...,x_n+x_n'\varepsilon)\in V\times\mathbb{R}^n$. □

To use definition (4.1) for Automatic Differentiation, we need to show that it behaves well for automatically differentiable functions as defined in Definition 2.1. Basically, we need to show that (4.1) is compatible with the chain rule.

Proposition 4.2. *Let $h:X\to\mathbb{R}$ defined on open $X\subset\mathbb{R}^n$ be automatically differentiable, $h\notin\{\varphi_i\mid i\in I\}$. Then*

$$\begin{aligned}&\widehat{h}(\mathbf{x}+\overrightarrow{\mathbf{x}}\varepsilon)\\&=\widehat{h}_\ell(x_{0,1}+\varepsilon x_{0,1}',...,x_{0,n_0}+\varepsilon x_{0,n}',\widehat{h}_1(x_{1,1}+\varepsilon x_{1,1}',...,x_{1,n_1}+\varepsilon x_{1,n_1}'),\\&\quad...,\widehat{h}_{\ell-1}(x_{\ell-1,1}+\varepsilon x_{\ell-1,1}',...,x_{\ell-1,n_{\ell-1}}+\varepsilon x_{\ell-1,n_{\ell-1}}')),\end{aligned}\tag{4.3}$$

for all $\mathbf{x}+\varepsilon\overrightarrow{\mathbf{x}}:=(x_1+\varepsilon x_1',...,x_n+\varepsilon x_n')\in X\times\mathbb{R}^n$, with $x_{k,i}+x_{k,i}'\in\{x_1+\varepsilon x_1',...,x_n+\varepsilon x_n'\}\cap X_k$.

Proof. We prove this statement by direct computation.[8] In the following, denote

$$\mathbf{x_k}:=(x_{k,1},...,x_{k,n_k})\in X_k\subset\mathbb{R}^{n_k}\quad\text{and}\quad\overrightarrow{\mathbf{x_k}}:=(x_{k,1}',...x_{k,n_k}')\in X_k\times\mathbb{R}^{n_k}.$$

Then the right hand-side of equation (4.3) is equal to

$$\begin{aligned}&\widehat{h_\ell}\left(\mathbf{x_0},h_1(\mathbf{x_1})+\left(\nabla h_1(\mathbf{x_1})\cdot\overrightarrow{\mathbf{x_k}}\right)\varepsilon,\right.\\&\qquad\left.\ldots,h_{\ell-1}(\mathbf{x_{\ell-1}})+\left(\nabla h_{\ell-1}(\mathbf{x_{\ell-1}})\cdot\overrightarrow{\mathbf{x}}_{\ell-\mathbf{1}}\right)\varepsilon\right)\end{aligned}\tag{4.4}$$

$$=h_\ell\left(\mathbf{x_0},h_1(\mathbf{x_1}),...,h_{\ell-1}(\mathbf{x_{\ell-1}})\right)+\left(\nabla h_\ell\left(...\right)\cdot\begin{pmatrix}x_{0,1}'\\\vdots\\x_{0,n}'\\\nabla h_1(\mathbf{x_1})\cdot\overrightarrow{\mathbf{x_1}}\\\vdots\\\nabla h_{\ell-1}(\mathbf{x_{\ell-1}})\cdot\overrightarrow{\mathbf{x}}_{\ell-\mathbf{1}}\end{pmatrix}\right)\varepsilon,$$

[8]A more elegant proof can be given by writing h as a composition and using the fact that the push-forward operator (see Section 7) is a functor.

where

$$\begin{aligned}\nabla h_\ell\,(...) &= \nabla h_\ell\,(\mathbf{x_0}, h_1(\mathbf{x_1}), ..., h_{\ell-1}(\mathbf{x_{\ell-1}})) \\ &= \frac{dh_\ell}{d(\mathbf{x_0}, h_1(\mathbf{x_1}), ..., h_{\ell-1}(\mathbf{x_{\ell-1}}))}(\mathbf{x_0}, h_1(\mathbf{x_1}), ..., h_{\ell-1}(\mathbf{x_{\ell-1}})).\end{aligned}$$

The left hand side of (4.3) is obviously equal to

$$h(\mathbf{x}) + \left(\nabla h(\mathbf{x}) \cdot \overrightarrow{\mathbf{x}}\right) \cdot \varepsilon. \tag{4.5}$$

By assumption, $h(\mathbf{x}) = h_\ell(\mathbf{x_0}, h_1(\mathbf{x_1}), ..., h_{\ell-1}(\mathbf{x_{\ell-1}}))$. Further, by the chain rule,

$$\begin{aligned}\nabla h(\mathbf{x}) \cdot \overrightarrow{\mathbf{x}} &= \frac{dh_\ell}{d\mathbf{x}}(\mathbf{x_0}, h_1(\mathbf{x_1}), ..., h_{\ell-1}(\mathbf{x_{\ell-1}})) \cdot \overrightarrow{\mathbf{x}} \\ &= \nabla h_\ell\,(\mathbf{x_0}, h_1(\mathbf{x_1}), ..., h_{\ell-1}(\mathbf{x_{\ell-1}})) \\ &\quad \cdot \frac{d\,(\mathbf{x} \mapsto (\mathbf{x_0}, h_1(\mathbf{x_1}), ..., h_{\ell-1}(\mathbf{x_{\ell-1}}))}{d\mathbf{x}}(\mathbf{x}) \cdot \overrightarrow{\mathbf{x}}.\end{aligned}$$

Now, $\frac{d(\mathbf{x} \mapsto (\mathbf{x_0}, h_1(\mathbf{x_1}), ..., h_{\ell-1}(\mathbf{x_{\ell-1}}))}{d\mathbf{x}}(\mathbf{x})$ equals

$$\begin{pmatrix} \frac{\partial(\mathbf{x} \mapsto x_{0,1})}{\partial x_1}(\mathbf{x}) & \cdots & \frac{\partial(\mathbf{x} \mapsto x_{0,1})}{\partial x_n}(\mathbf{x}) \\ \vdots & \vdots & \vdots \\ \frac{\partial(\mathbf{x} \mapsto x_{0,n_0})}{\partial x_1}(\mathbf{x}) & \cdots & \frac{\partial(\mathbf{x} \mapsto x_{0,n_0})}{\partial x_n}(\mathbf{x}) \\ \frac{\partial(\mathbf{x} \mapsto h_1(\mathbf{x_1}))}{\partial x_1}(\mathbf{x}) & \cdots & \frac{\partial(\mathbf{x} \mapsto h_1(\mathbf{x_1}))}{\partial x_n}(\mathbf{x}) \\ \vdots & \vdots & \vdots \\ \frac{\partial(\mathbf{x} \mapsto h_{\ell-1}(\mathbf{x_{\ell-1}}))}{\partial x_1}(\mathbf{x}) & \cdots & \frac{\partial(\mathbf{x} \mapsto h_{\ell-1}(\mathbf{x_{\ell-1}}))}{\partial x_n}(\mathbf{x}) \end{pmatrix}$$

Hence,

$$\frac{d\,(\mathbf{x} \mapsto (\mathbf{x_0}, h_1(\mathbf{x_1}), ..., h_{\ell-1}(\mathbf{x_{\ell-1}}))}{d\mathbf{x}}(\mathbf{x}) \cdot \overrightarrow{\mathbf{x}} = \begin{pmatrix} x'_{0,1} \\ \vdots \\ x'_{0,n} \\ \nabla h_1(\overrightarrow{\mathbf{x_1}}) \cdot \overrightarrow{\mathbf{x_1}} \\ \vdots \\ \nabla h_{\ell-1}(\mathbf{x_{\ell-1}}) \cdot \overrightarrow{\mathbf{x}}_{\ell-1} \end{pmatrix}$$

Thus, (4.4) equals (4.5) and we are done. □

We can now automatically compute directional derivatives $J_f(\mathbf{c}) \cdot \overrightarrow{\mathbf{x}}$ of an automatically differentiable function $f : X \to \mathbb{R}^m$, on open $X \subset \mathbb{R}^n$, at fixed $\mathbf{c} = (c_1, ..., c_n) \in \mathbb{R}^n$ in direction of fixed $\overrightarrow{\mathbf{x}} = (x'_1, ..., x'_n) \in \mathbb{R}^n$ by computing the directional derivatives $\nabla f_j(\mathbf{c}) \cdot \overrightarrow{\mathbf{x}}$ in the following way:

Theorem 4.3. *Assume that definition 4.1 is implemented for all elementary functions in the set* $\{\varphi_i \mid i \in I\}$. *Then the directional derivative* $\nabla f_j(\mathbf{c}) \cdot \overrightarrow{\mathbf{x}}$ *of an automatically differentiable function* $f_j : X \to \mathbb{R}$ *can be computed 'automatically' through extending* f_j *to* $X \times \mathbb{R}^n \subset \mathcal{D}^n$, *and evaluating the dual part of* $\widehat{f_j}(c_1 + x'_1\varepsilon, ..., c_n + x'_n\varepsilon)$.

Proof. Each f_j is automatically differentiable. By assumption, the case $f_j \in \{\varphi_i \mid i \in I\}$ is clear: We simply obtain the pair $\widehat{f_j}(c_1 + x'_1\varepsilon, ..., c_n + x'_n\varepsilon)$ by calling f_j and all $\frac{\partial f_j}{\partial x_k}$ and computing the gradient-vector product. The sought directional derivative is the dual part (second entry) of that pair.

So assume that there exists real-valued $h_1, ..., h_\ell$ on open sets $X_k \subset \mathbb{R}^{n_k}$, such that for all $\mathbf{x} \in X$,

$$f_j(\mathbf{x}) = h_\ell(x_{0,1}, ..., x_{0,n_0}, h_1(x_{1,1}, ..., x_{1,n_1}), ..., h_{\ell-1}(x_{\ell-1,1}, ..., x_{\ell-1,n_{\ell-1}})),$$

for suitable $x_{k,j}$, with $h_\ell \in \{\varphi_i \mid i \in I\}$ and $h_1, ..., h_{\ell-1}$ automatically differentiable.

We proceed by induction on the depth of f_j.

Base case: Assume that each $h_1, ..., h_\ell \in \{\varphi_i \mid i \in I\}$. Extending f_j to $X \times \mathbb{R}^n \subset \mathcal{D}^n$ leads to the extension of $h_1, ..., h_\ell$ to sets $X_k \times \mathbb{R}^{n_k} \subset \mathcal{D}^{n_k}$. Since definition 4.1 is implemented for all functions in $\{\varphi_i \mid i \in I\}$,

$$\widehat{h}_k(c_{k,1} + x'_{k,1}\varepsilon, ..., c_{k,n_k} + x'_{k,n_k}\varepsilon)$$

is defined for all h_k and computed by calling h_k and all $\frac{\partial h_k}{\partial v_j}$ with suitable inputs and computing the gradient-vector product. By Proposition 4.2, the computation of

$$\widehat{h}_\ell(c_{0,1} + x'_{0,1}\varepsilon, ..., c_{0,n_0} + x'_{0,n_0}\varepsilon, \widehat{h}_1(\cdots), ..., \widehat{h}_{\ell-1}(\cdots)),$$

which is performed last, gives $\widehat{f_j}(c_1 + x'_1\varepsilon, ..., c_n + x'_n\varepsilon)$, whose dual part is $\nabla f_j(\mathbf{c}) \cdot \overrightarrow{\mathbf{x}}$.

Induction step: Assume that $h_{j_0} \in \{h_1, ..., h_{\ell-1}\}$ is not an elementary function. Again, we extend f_j to $X \times \mathbb{R}^n \subset \mathcal{D}^n$, which leads to the extension of $h_1, ..., h_\ell$ to sets $X_k \times \mathbb{R}^{n_k} \subset \mathcal{D}^{n_k}$. Since h_{j_0} is still automatically differentiable

$$\widehat{h}_{j_0}(c_{j_0,1} + x'_{j_0,1}\varepsilon, ..., c_{j_0,n_{j_0}} + x'_{j_0,n_{j_0}}\varepsilon)$$

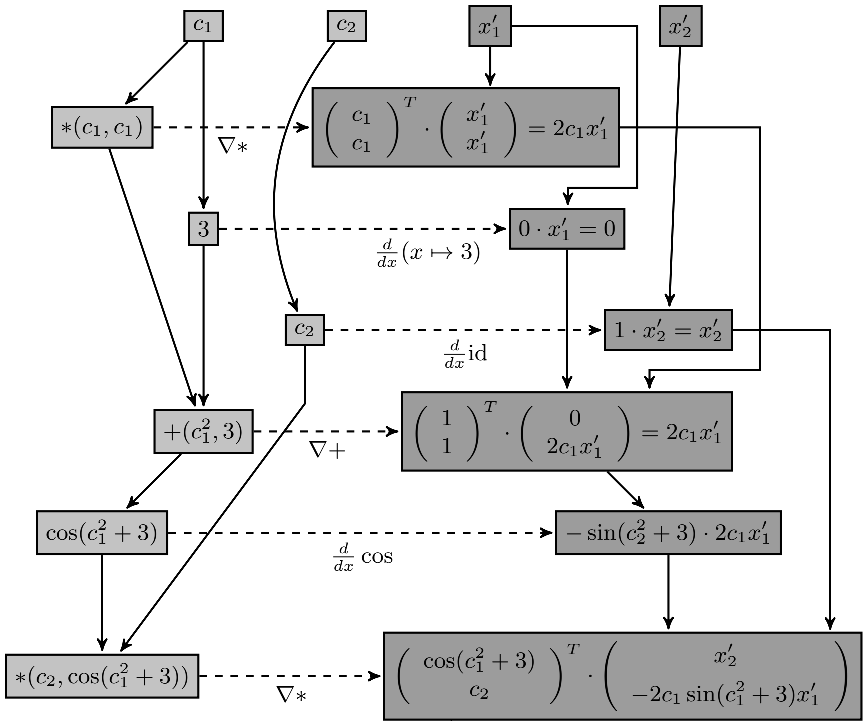
is computed by Induction Assumption. Then again, the computation of

$$\widehat{h}_\ell(c_{0,1} + x'_{0,1}\varepsilon, ..., c_{0,n_0} + x'_{0,n_0}\varepsilon, \widehat{h}_1(\cdots), ..., \widehat{h}_{\ell-1}(\cdots))$$

(note that h_ℓ is elementary) gives, by Proposition 4.2, the dual number $\widehat{f_j}(c_1 + x'_1\varepsilon, ..., c_n + x'_n\varepsilon)$. ☐

Example 4.4. Consider the function $f : \mathbb{R}^2 \to \mathbb{R}$ given by

$$f(x_1, x_2) = x_2 * \cos(x_1^2 + 3) = x_2 * \cos(x_1 * x_1 + 3).$$



c_1
c_2
x'_1
x'_2
$*(c_1, c_1)$
$\nabla *$
$\begin{pmatrix} c_1 \\ c_1 \end{pmatrix}^T \cdot \begin{pmatrix} x'_1 \\ x'_1 \end{pmatrix} = 2c_1 x'_1$
3
$\frac{d}{dx}(x \mapsto 3)$
$0 \cdot x'_1 = 0$
c_2
$\frac{d}{dx}\mathrm{id}$
$1 \cdot x'_2 = x'_2$
$+(c_1^2, 3)$
$\nabla +$
$\begin{pmatrix} 1 \\ 1 \end{pmatrix}^T \cdot \begin{pmatrix} 0 \\ 2c_1 x'_1 \end{pmatrix} = 2c_1 x'_1$
$\cos(c_1^2 + 3)$
$\frac{d}{dx}\cos$
$-\sin(c_2^2 + 3) \cdot 2c_1 x'_1$
$*(c_2, \cos(c_1^2 + 3))$
$\nabla *$
$\begin{pmatrix} \cos(c_1^2 + 3) \\ c_2 \end{pmatrix}^T \cdot \begin{pmatrix} x'_2 \\ -2c_1 \sin(c_1^2 + 3) x'_1 \end{pmatrix}$

Figure 5: Computational graph for Example 4.4 with primal parts in blue and dual parts in red.

Let $c_1, c_2, x'_1, x'_2 \in \mathbb{R}$. We evaluate the value of $\widehat{f}$ at $c_1 + x'_1\varepsilon$ and $c_2 + x'_2\varepsilon$. By definition (4.1), Proposition 4.1 and Proposition 4.2,

$$\begin{aligned}
&\widehat{f}(c_1 + x'_1\varepsilon, c_2 + x'_2\varepsilon) \\
&= (c_2 + x'_2\varepsilon) * \cos((c_1 + x'_1\varepsilon)^2 + 3) \\
&= (c_2 + x'_2\varepsilon) * \cos(c_1^2 + 2c_1x'_1\varepsilon + 3) \\
&= (c_2 + x'_2\varepsilon) * \cos((c_1^2 + 3) + 2c_1x'_1\varepsilon) \\
&= (c_2 + x'_2\varepsilon) * (\cos(c_1^2 + 3) - \sin(c_1^2 + 3)2c_1x'_1\varepsilon) \\
&= c_2\cos(c_1^2 + 3) + \left(-2c_1c_2\sin(c_1^2 + 3)x'_1 + \cos(c_1^2 + 3)x'_2\right)\varepsilon.
\end{aligned}$$

By Theorem 4.3, the dual part of this expression is $\nabla f(c_1, c_2) \cdot \begin{pmatrix} x'_1 \\ x'_2 \end{pmatrix}$. That is,

$$\nabla f(c_1, c_2) \cdot \begin{pmatrix} x'_1 \\ x'_2 \end{pmatrix} = -2c_1c_2\sin(c_1^2 + 3)x'_1 + \cos(c_1^2 + 3))x'_2.$$

Thus, a (basic) implementation of an Automatic Differentiation System can be realised by implementing (4.1) for all elementary functions. Usually, this

```
-- define Dual numbers
instance Num Dual where
  (Dual x x') + (Dual y y') = Dual (x + y) (x' + y')
  (Dual x x') * (Dual y y') = Dual (x * y) (x * y' + x' * y)

-- overload elementary functions
instance Floating Dual where
   sin (Dual x x') = Dual (sin x) (x' * (cos x))
   cos (Dual x x') = Dual (cos x) (x' * (-(sin x)))
   exp (Dual x x') = Dual (exp x) (x' * (exp x))

-- embed reals into duals
fromDouble x = Dual x 0

-- test case
let f x1 x2 = x2 * cos((x1 * x1) + 3.0) in f(Dual 5 1)(Dual 2 0)
```

Figure 6: Minimal Haskell example, showing the implementation of Forward AD for Example 4.4 and the test case $\frac{\partial f}{\partial x_1}(5, 2)$. Compare also the basically identical work in [17, Subsection 2.1].

is done by simply overloading elementary functions. Constant functions will usually be identified with real numbers, which themselves will be lifted to dual number with zero dual part (see Figure 6).

Note again that, at no time during the described process, any symbolic differentiation takes place. Instead, since each 'top-level' function h_ℓ of an automatically differentiable function is elementary, we are computing and passing on pairs

$$\left(\varphi_i(c_{i,1}, ..., c_{i,n_i}), \nabla\varphi_i(c_{i,1}, ..., c_{i,n_i}) \cdot \begin{pmatrix} x'_{i,1} \\ \vdots \\ x'_{i,n_i} \end{pmatrix} \right) \in \mathbb{R}^2, \tag{4.6}$$

which computes 'automatically' the directional derivatives $\nabla f_j(\mathbf{c}) \cdot \overrightarrow{\mathbf{x}}$ and, therefore, the directional derivative $J_f(\mathbf{c}) \cdot \overrightarrow{\mathbf{x}}$.

Note further that the pairs (4.6) are <u>exactly the same pairs</u> as the ones in (3.3). This means that the processes described in this and in the previous Section reduce to <u>exactly the same</u> computations.

Indeed, if we store the pairs (c_i, x'_i) and (4.6) in an array, we obtain the evaluation trace pairs $[\mathbf{v}^{[i]}, \mathbf{v}'^{[i]}]$. In summary:

Theorem 4.5. *By the above, given* $\mathbf{c} \in X \subset \mathbb{R}^n$ *and* $\overrightarrow{\mathbf{x}} \in \mathbb{R}^n$*, the evaluation of* $J_f(\mathbf{c}) \cdot \overrightarrow{\mathbf{x}}$ *of an automatically differentiable function* $f : X \to \mathbb{R}^m$ *can be achieved through the lifting of each* f_j *to a function* $\widehat{f}_j : X \times \mathbb{R}^n \to \mathcal{D}$ *as defined in* (4.1) *and by evaluating* $\widehat{f}_j(c_1 + x'_1\varepsilon, ..., c_n + x'_n\varepsilon)$*. This process is equivalent to the computation of the evaluation trace pairs* $[\mathbf{v}^{[i]}, \mathbf{v}'^{[i]}]$*, as described in the previous section, and to the computation of the pairs* (4.6) *in suitable order.*

5 Comparison with Symbolic Differentiation—Some Thoughts on Complexity

Before we move on to further descriptions of Forward AD and to a brief description of the Reverse Mode, we take a look at some example cases to demonstrate how Automatic Differentiation solves some complexity issues which appear in systems which use symbolic differentiation.

Assume that we want to obtain the derivative of a composition f of univariate elementary functions $\varphi_1, ..., \varphi_n : \mathbb{R} \to \mathbb{R}$, that is,

$$f = \varphi_n \circ \cdots \circ \varphi_1,$$

at a certain value $c \in \mathbb{R}$.

A symbolic differentiation system will first use the chain rule to determine the derivative function f', which is given by

$$f'(x) = \varphi'_n(\varphi_{n-1}(\cdots(\varphi_1(x)))) \cdot \varphi'_{n-1}(\varphi_{n-2}(\cdots(\varphi_1(x)))) \cdot \;\cdots\; \cdot \varphi'_2(\varphi_1(x)) \cdot \varphi'_1(x),$$

for all $x \in \mathbb{R}$, and then compute $f'(c)$ by substituting x by c. That is, each factor is computed and the results are multiplied. Hence, the system computes the following values:

$$\begin{aligned}
&\varphi'_1(c)\\
&\varphi_1(c),\ \varphi'_2(\varphi_1(c))\\
&\varphi_1(c),\ \varphi_2(\varphi_1(c)),\ \varphi'_3(\varphi_2(\varphi_1(c)))\\
&\varphi_1(c),\ \varphi_2(\varphi_1(c)),\ \varphi_3(\varphi_2(\varphi_1(c))),\ \varphi'_4(\varphi_3(\varphi_2(\varphi_1(c))))\\
&\vdots\\
&\varphi_1(c),\ \varphi_2(\varphi_1(c)),\ \varphi_3(\varphi_2(\varphi_1(c))), ...,\ \varphi'_n(\varphi_{n-1}(\cdots(\varphi_1(c))))
\end{aligned}$$

As we see many expressions will be computed multiple times (*loss of sharing*). If we ignore the time the system needs to determine f', as well as the time for performing multiplications, and set the cost for the computation of each value of φ_i, φ'_i as 1, then the total cost of computing $f'(c)$ is

$$1 + 2 + 3 + \cdots + n = \sum_{k=1}^{n} k = \frac{n(n+1)}{2} \in \mathcal{O}(n^2).$$

In comparison, a Forward Automatic Differentiation system will perform a computation of pairs starting with $\big(\varphi_1(c), \varphi'_1(c) \cdot 1\big)$, where the $(i+1)$-st pair for $i \geq 1$ looks like

$$\big(\varphi_{i+1}(v_i),\ \varphi'_{i+1}(v_i) \cdot v'_i\big),$$

for some $v_i, v'_i \in \mathbb{R}$. See the computational graph in Figure 1 in Subsection 2.1 for the case $n = 3$ (set $x' = 1$). If we again ignore costs for multiplications, the cost for evaluating each pair is 2. Hence, the total costs of evaluating $f'(c)$ via Automatic Differentiation is $2n$.

In [19] the example of a product of the form

$$f = \varphi_n * \cdots * \varphi_1 = *(\varphi_n,\ *(\varphi_{n-1}, *(\cdots(*(\varphi_2, \varphi_1)))))$$

is given. Here, the evaluation of the derivative of $f'(c)$ at some $c \in \mathbb{R}$ by a symbolic differentiation system will first use the product rule to compute f' given by

$$\begin{aligned} f'(x) =& \varphi_n'(x) \cdot (\varphi_{n-1}(x) \cdots \varphi_1(x)) + \varphi_{n-1}'(x) \cdot (\varphi_n(x) \cdot \varphi_{n-2}(x) \cdots \varphi_1(x)) \\ &+ \cdots + \varphi_1'(x) \cdot (\varphi_n(x) \cdots \varphi_2(x)), \end{aligned}$$

for all $x \in \mathbb{R}$, and then again substitute x by c. Again, many function values will be computed multiple times. Since we have n functions in each summand and n summands, ignoring cost for multiplications and addition, the computation of $f'(c)$ has a total cost of n^2.

In contrast, a Forward Automatic Differentiation system will compute pairs starting with

$$\big(\varphi_1(c),\ \varphi_1'(c) \cdot 1\big),\ \big(\varphi_2(c),\ \varphi_2'(c) \cdot 1\big),\ \left(\varphi_1(c) * \varphi_2(c),\ \begin{pmatrix} \varphi_2(c) \\ \varphi_1(c) \end{pmatrix}^T \cdot \begin{pmatrix} \varphi_1'(c) \\ \varphi_2'(c) \end{pmatrix}\right)$$

where the remaining pairs for $i \geq 2$ look like

$$\big(\varphi_{i+1}(c),\ \varphi_{i+1}'(c) \cdot 1\big), \qquad \left(\varphi_{i+1}(c) * v_i,\ \begin{pmatrix} v_i \\ \varphi_{i+1}(c) \end{pmatrix}^T \cdot \begin{pmatrix} \varphi_{i+1}'(c) \\ v_i' \end{pmatrix}\right)$$

for some $v_i, v_i' \in \mathbb{R}$. Since the 3rd, 5th, 7th etc. pairs are those which are created by lifting $*$ to the dual numbers, they contain only additions and multiplications of values which have already been computed. Hence, for simplicity, we may discard these pairs with regards to the costs of the evaluations of $f'(c)$. Thus, the total cost of computing $f'(c)$ is the cost of computing the n pairs $\big(\varphi_{i+1}(c),\ \varphi_{i+1}'(c) \cdot 1\big)$ which is $2n$.

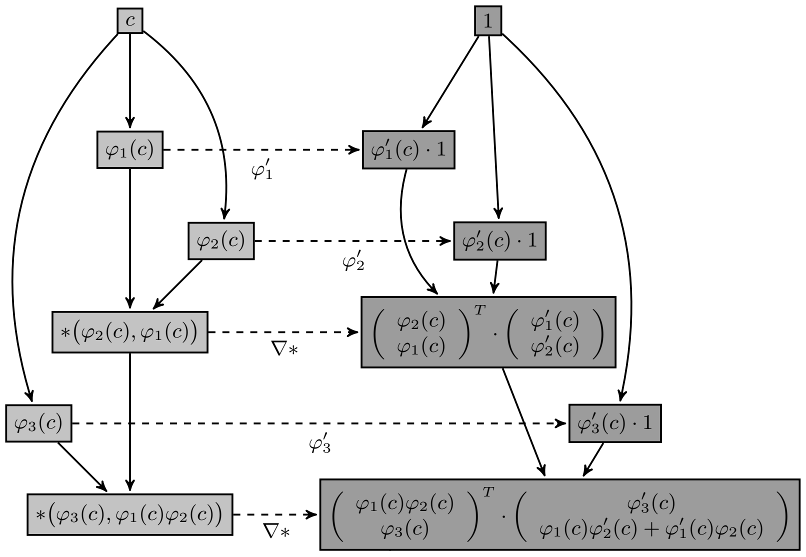
A further advantage of Forward AD, at least in many implementation, is the efficient handling of common intermediate expressions feeding into several subsequent intermediates. Consider for this the case of a sum of the n elementary uni-variate functions φ_i each composed with another univariate elementary function $\psi : \mathbb{R} \to \mathbb{R}$:

$$f = (\varphi_1 \circ \psi) + \cdots + (\varphi_n \circ \psi) = +(\varphi_n \circ \psi,\ +(\varphi_{n-1} \circ \psi, +(\cdots(+(\varphi_2 \circ \psi, \varphi_1 \circ \psi)))))$$

The derivative f' is in this case obviously given by

$$f'(x) = \psi'(x) \cdot \varphi_1'(\psi(x)) + \cdots + \psi'(x) \cdot \varphi_n'(\psi(x)),$$

for all $x \in \mathbb{R}$. To determine $f'(c)$, a symbolic differentiation system will evaluate $\psi(c)$ in each summand, that is n times (and possibly, if the expression $\psi'(x)$ is not factored out, $\psi'(c)$ as well n-times). Hence, the computational cost of



c
1
$\varphi_1(c)$
$\varphi_1'(c)\cdot 1$
φ_1'
$\varphi_2(c)$
$\varphi_2'(c)\cdot 1$
φ_2'
$*(\varphi_2(c),\varphi_1(c))$
$\nabla *$
$\begin{pmatrix}\varphi_2(c)\\ \varphi_1(c)\end{pmatrix}^T\cdot\begin{pmatrix}\varphi_1'(c)\\ \varphi_2'(c)\end{pmatrix}$
$\varphi_3(c)$
$\varphi_3'(c)\cdot 1$
φ_3'
$*(\varphi_3(c),\varphi_1(c)\varphi_2(c))$
$\nabla *$
$\begin{pmatrix}\varphi_1(c)\varphi_2(c)\\ \varphi_3(c)\end{pmatrix}^T\cdot\begin{pmatrix}\varphi_3'(c)\\ \varphi_1(c)\varphi_2'(c)+\varphi_1'(c)\varphi_2(c)\end{pmatrix}$

Figure 7: Computational graph for the computation of $f'(c) = \frac{d}{dx}\left(\varphi_n * \cdots * \varphi_1\right)(c)$ in the case $n = 3$.

evaluating $f'(c)$ is at least $2n+1$ (if $\psi'(x)$ is factored out, $3n$ otherwise), where we again ignore costs for additions and multiplications and for determining the derivative function f'. If we want to obtain the value $f(c)$, too, the cost increases to $4n+1$ (or $6n$, respectively).

In this particular case, some realisations of a Forward Automatic Differentiation system might evaluate $\psi'(c)$ n-times as well. However, in many implementations the value $\psi(c)$ will be assigned to a new variable z, such that

$$f(c) = \varphi_1(z) + \cdots + \varphi_n(z) \quad\text{and}\quad f'(c) = z'\cdot\varphi_1'(z) + \cdots + z'\cdot\varphi_n'(z),$$

where $z' := \psi'(c)$. The Forward AD system will then compute pairs starting with

$$\begin{aligned}&\big(z,\ z'\cdot 1\big),\ \ \big(\varphi_1(z),\ \varphi_1'(z)\cdot z'\big),\ \ \big(\varphi_2(z),\ \varphi_2'(z)\cdot z'\big),\\ &\big(\varphi_1(z)+\varphi_2(z),\ \varphi_1'(z)\cdot z' + \varphi_2'(z)\cdot z'\big)\ \text{ etc.}\end{aligned}$$

Clearly, in this process the values (numbers) $\psi(c) = z$ and $\psi'(c) = z'$ are computed only once. If we again ignore costs for additions and multiplications (including the cost for computing pairs created by lifting $+$), the total cost of determining both $f(c)$ and $f'(c)$ is the cost of computing the $n+1$ pairs $\big(z,\ z'\cdot 1\big)$, $\big(\varphi_i(z),\ \varphi_i'(z)\cdot z'\big)$, $i = 1,...,n$, which is $2n+2$.[9] Note that the substitution $z := \psi(x)$ in a symbolic differentiation system would not have the

[9]Admittedly, in this particular case, if one wants to determine only the derivative $f'(c)$, the symbolic evaluation appears to be slightly faster than FAD. However, we have chosen this ex-

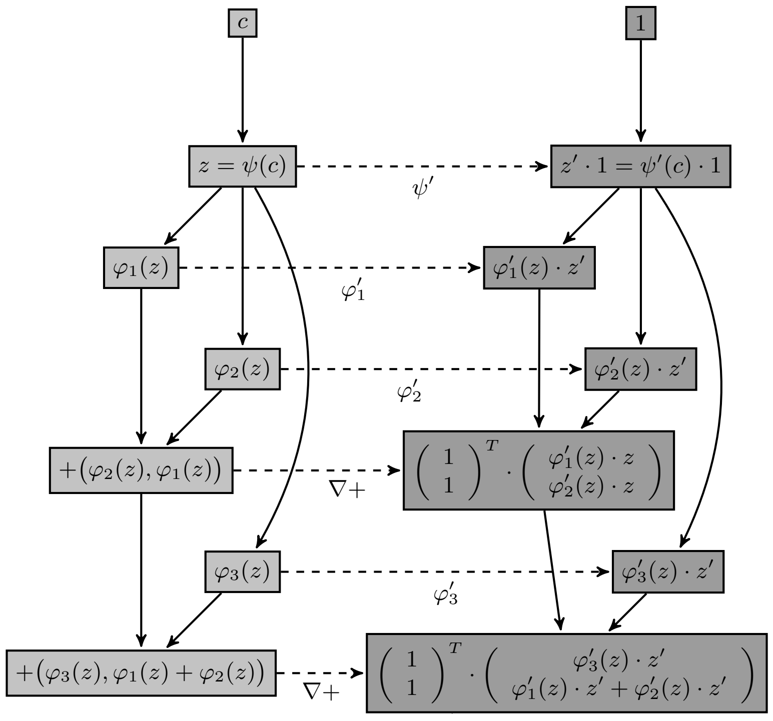

c
1
$z = \psi(c)$
$z' \cdot 1 = \psi'(c) \cdot 1$
ψ'
$\varphi_1(z)$
$\varphi_1'(z) \cdot z'$
φ_1'
$\varphi_2(z)$
$\varphi_2'(z) \cdot z'$
φ_2'
$+(\varphi_2(z), \varphi_1(z))$
$\nabla +$
$\begin{pmatrix} 1 \\ 1 \end{pmatrix}^T \cdot \begin{pmatrix} \varphi_1'(z) \cdot z \\ \varphi_2'(z) \cdot z \end{pmatrix}$
$\varphi_3(z)$
$\varphi_3'(z) \cdot z'$
φ_3'
$+(\varphi_3(z), \varphi_1(z) + \varphi_2(z))$
$\nabla +$
$\begin{pmatrix} 1 \\ 1 \end{pmatrix}^T \cdot \begin{pmatrix} \varphi_3'(z) \cdot z' \\ \varphi_1'(z) \cdot z' + \varphi_2'(z) \cdot z' \end{pmatrix}$

Figure 8: Computational graph for the computation of $f(c) = \varphi_1(z) + \cdots + \varphi_n(z)$ and $f'(c) = z' \cdot \varphi_1'(z) + \cdots + z' \cdot \varphi_n'(z)$ in the case $n = 3$, where $z := \psi(c)$ and $z' := \psi'(c)$.

same effect of avoiding redundant calculations. Since $\psi(x)$ is not a number, but an algebraic expression, to obtain $f'(c)$ the variable x would still have to be substituted by c in each instance of z in $f'(x) = z' \cdot \varphi_1'(z) + \cdots + z' \cdot \varphi_n'(z)$.

Of course, the situation is more difficult when the function f is more complicated or when multi-variate elementary functions other than $+$ or $*$ are involved. However, we hope to have demonstrated that, in general, Forward Automatic Differentiation avoids redundant computations of common sub-expressions and does not suffer from the same complexity issues as symbolic computation.

6 Forward AD and Taylor Series expansion

In the literature (see, for example, [20, Section 2]), definition (4.1) is sometimes described as being obtained by evaluating the Taylor series expansion of $\widehat{h}$ about $(x_1 + x_1'\varepsilon, ..., x_n + x_n'\varepsilon)$.

To understand this argument, recall that the Taylor series of an infinitely many times differentiable multivariate function $h : X \to \mathbb{R}$ on an open set $X \subset \mathbb{R}^n$ about some point $\mathbf{c} = (c_1, ..., c_n) \in X$ is given by

$$T(h; \mathbf{c})(\mathbf{x}) = \sum_{k_1+\cdots+k_n=0}^{\infty} \frac{(x_1 - c_1)^{k_1} \cdots (x_n - c_n)^{k_n}}{k_1! \cdots k_n!} \frac{\partial^{k_1+\cdots+k_n} h}{\partial x_1^{k_1} \cdots \partial x_n^{k_n}}(\mathbf{c}),$$

for all $\mathbf{x} = (x_1, ..., x_n) \in X$.

Let now $\tilde{h} : \mathcal{D}^n \supset X \times \mathbb{R}^n \to \mathcal{D}$ be an extension of h to the dual numbers (that is $\tilde{h}|_X = h$). We define the Taylor series of $\tilde{h}$ about some vector of dual numbers $(\mathbf{c}, \vec{\mathbf{c}}) := (c_1 + c_1'\varepsilon, ..., c_n + c_n'\varepsilon) \in X \times \mathbb{R}^n$ analogously to the real case. That is,

$$\begin{aligned} &T(\tilde{h}; (\mathbf{c}, \vec{\mathbf{c}}))((\mathbf{x}, \vec{\mathbf{x}})) \\ &\quad = \sum_{k_1+\cdots+k_n=0}^{\infty} \Bigg(\frac{(x_1 - c_1 + (x_1' - c_1')\varepsilon)^{k_1} \cdots (x_n - c_n + (x_n' - c_n')\varepsilon)^{k_n}}{k_1! \cdots k_n!} \\ &\qquad\qquad \cdot \frac{\partial^{k_1+\cdots+k_n} \tilde{h}}{\partial x_1^{k_1} \cdots \partial x_n^{k_n}}((\mathbf{c}, \vec{\mathbf{c}})) \Bigg), \end{aligned}$$

for all $(\mathbf{x}, \vec{\mathbf{x}}) := (x_1 + x_1'\varepsilon, ..., x_n + x_n'\varepsilon) \in X \times \mathbb{R}^n$.

Trivially, this series converges for $(\mathbf{x}, \vec{\mathbf{x}}) = (\mathbf{c}, \vec{\mathbf{c}})$. Further, due to $\varepsilon^2 = 0$, the Taylor series about any $(\mathbf{x}, \mathbf{0}) = (x_1 + 0\varepsilon, ..., x_n + 0\varepsilon) \in X \times \mathbb{R}^n$ converges for the arguments $(\mathbf{x}, \vec{\mathbf{x}}) = (x_1 + x_1'\varepsilon, ..., x_n + x_n'\varepsilon)$, for all $\vec{\mathbf{x}} \in \mathbb{R}^n$. We have,

ample mainly to demonstrate how the redundant computation of common sub-expressions can be avoided using Automatic Differentiation. Note further that we have disregarded the cost for determining the derivative function f' in a symbolic differentiation in our considerations.

identifying $\mathbf{x}$ with $(\mathbf{x}, \mathbf{0})$,

$$\begin{aligned}
T(\tilde{h}; \mathbf{x})(\mathbf{x}, \vec{\mathbf{x}}) &= \sum_{k_1+\cdots+k_n=0}^{\infty} \frac{(x'_1\varepsilon)^{k_1}\cdots(x'_n\varepsilon)^{k_n}}{k_1!\cdots k_n!} \frac{\partial^{k_1+\cdots+k_n}}{\partial x_1^{k_1}\cdots\partial x_n^{k_n}}\tilde{h}(\mathbf{x}) \\
&= \sum_{k_1+\cdots+k_n=0}^{1} \frac{(x'_1\varepsilon)^{k_1}\cdots(x'_n\varepsilon)^{k_n}}{k_1!\cdots k_n!} \frac{\partial^{k_1+\cdots+k_n}}{\partial x_1^{k_1}\cdots\partial x_n^{k_n}}\tilde{h}(\mathbf{x}) \qquad (6.1) \\
&= \tilde{h}(\mathbf{x}) + \sum_{j=1}^{n} \frac{\partial}{\partial x_j}\tilde{h}(\mathbf{x}) \cdot x'_j\varepsilon \\
&= \tilde{h}(\mathbf{x}) + \left(\nabla\tilde{h}(\mathbf{x}) \cdot \begin{pmatrix} x'_1 \\ \vdots \\ x'_n \end{pmatrix}\right)\varepsilon = h(\mathbf{x}) + (\nabla h(\mathbf{x}) \cdot \vec{\mathbf{x}})\varepsilon,
\end{aligned}$$

where $\nabla\tilde{h}(\mathbf{x}) := \left(\frac{\partial\tilde{h}}{\partial x_1}(\mathbf{x})\cdots\frac{\partial\tilde{h}}{\partial x_n}(\mathbf{x})\right) = \nabla h(\mathbf{x})$.

As we see, the right-hand side of the last equation is equal to $\widehat{h}(x_1+x'_1\varepsilon, ..., x_n+x'_n\varepsilon)$ in definition (4.1). Hence, if we choose $\tilde{h}$ as $\widehat{h}$, we obtain

$$\widehat{h}(x_1 + x'_1\varepsilon, ..., x_n + x'_n\varepsilon) = T(\widehat{h}; (x_1, ..., x_n))(x_1 + x'_1\varepsilon, ..., x_n + x'_n\varepsilon). \qquad (6.2)$$

That is:

Proposition 6.1. *The extension of an infinitely many times differentiable $h : X \to \mathbb{R}$, on open $X \subset \mathbb{R}^n$, to a set $X \times \mathbb{R}^n \subset \mathcal{D}^n$ as defined in* (4.1)*, is the (unique) function $\widehat{h}$, with the property that the images of any $(x_1 + x'_1\varepsilon, ..., x_n + x'_n\varepsilon) \in X \times \mathbb{R}^n$ under $\widehat{h}$ and $T(\widehat{h}; (x_1, ..., x_n))$ are equal.*

It is clear that the statement remains true if we replace 'infinitely many times differentiable' by 'differentiable' and $T(\widehat{h}; (x_1, ..., x_n))$ by the first-degree Taylor polynomial $T_1(\widehat{h}; (x_1, ..., x_n))$.[10]

Since we identify X with its natural embedding into $\mathcal{D}^n$, we can replace $\tilde{h}(\mathbf{x})$ by $h(\mathbf{x})$ in the right-hand side of (6.1). It is custom to do this in the left-hand side of (6.1) as well. That is, one usually writes $T(h; (x_1, ..., x_n))$ instead of $T(\tilde{h}; (x_1, ..., x_n))$ or $T(\widehat{h}; (x_1, ..., x_n))$.

By (6.2), it is obvious that one can describe the process of determining the directional derivatives $\nabla f_j(\mathbf{c}) \cdot \vec{\mathbf{x}}$ of each f_j in terms of Taylor series expansion, if f_j is infinitely many times differentiable, or its first-degree Taylor polynomial, otherwise.

Taylor series expansion or Taylor polynomials can also be used to compute higher-order partial derivatives which we discuss briefly in Section 8 (see also the work in [9, Chapter 13]).

[10]Note that we mean here by first-degree Taylor polynomial simply the polynomial in (6.1). That is, we make no statement about the existence of a remainder term, for which mere differentiability of h would not be sufficient.

7 Forward AD, Differential Geometry and Category Theory

In recent literature (see [19]) the extension of differentiable functions $h : X \to \mathbb{R}$ on open $X \subset \mathbb{R}^n$ to a function $\widehat{h} : \mathcal{D}^n \supset X \times \mathbb{R}^n \to \mathcal{D}$ is described in terms of the *push-forward* operator known from Differential Geometry. We shortly summarize the discussion provided in [19].

Let M, N be differentiable manifolds, TM, TN their tangent bundles and let $h : M \to N$ be a differentiable function. The push-forward (or *differential*) $T(h)$ of h can be defined[11] as

$$T(h) : TM \to TN \quad \text{with} \quad T(h)(\mathbf{x}, \overrightarrow{\mathbf{x}}) = (h(\mathbf{x}), d_{\mathbf{x}}h(\overrightarrow{\mathbf{x}})),$$

where $d_{\mathbf{x}}h(\overrightarrow{\mathbf{x}})$ is the *push-forward (or differential) of* h *at* $\mathbf{x}$ applied to $\overrightarrow{\mathbf{x}}$.

If now $f : X \to \mathbb{R}^m$ on open $X \subset \mathbb{R}^n$ is a differentiable function, this reads

$$T(f) : X \times \mathbb{R}^n \to \mathbb{R}^m \times \mathbb{R}^m \quad \text{with} \quad T(f)(\mathbf{x}, \overrightarrow{\mathbf{x}}) = \Big(f(\mathbf{x}),\ J_f(\mathbf{x}) \cdot \overrightarrow{\mathbf{x}}\Big).$$

Considering $X \times \mathbb{R}^n$ as a subset of $\mathcal{D}^n$ and identifying $\mathbb{R}^m \times \mathbb{R}^m$ with $\mathcal{D}^m$, in light of (4.2), this means nothing else than

$$T(f) = \widehat{f}.$$

Furthermore, in the special case of a real-valued and infinitely many times differentiable function $f_j : X \to \mathbb{R}$ on open $X \subset \mathbb{R}^n$ we also have, by equation (6.2),

$$T(f_j)(\mathbf{x}, \overrightarrow{\mathbf{x}}) = T(f_j; \mathbf{x})(\mathbf{x}, \overrightarrow{\mathbf{x}}), \quad \forall (\mathbf{x}, \overrightarrow{\mathbf{x}}) \in X \times \mathbb{R}^n,$$

which justifies using the letter T for both, the push-forward and the Taylor-series of f_j in this setting.

It is well-known that $T(id_M) = id_{TM}$ and that

$$T(h_2 \circ h_1) = T(h_2) \circ T(h_1),$$

for all $h_1 : M \to N$ and $h_2 : N \to L$, for differentiable manifolds M, N, L. That is, the mapping given by

$$\begin{aligned} M &\mapsto TM \\ h &\mapsto T(h) \end{aligned}$$

is a functor from the category of differentiable manifolds to the category of vector bundles (see, for example, [18, III, §2]). Furthermore, since $T\mathbb{R} = \mathbb{R}^2$, one can even consider the algebra of dual number $\mathcal{D}$ as the image of $\mathbb{R}$ under T, equipped with the push-forwards of addition and multiplication. I.e.,

$$(\mathcal{D}, +, \cdot) = (T\mathbb{R}, T(+), T(*)).$$

[11] The definition we are using here is the same as in [19]. Some authors define $T(h)$ via $T(h)(\mathbf{x}, \overrightarrow{\mathbf{x}}) := d_{\mathbf{x}}h(\overrightarrow{\mathbf{x}})$.

Extending this to higher dimensions, the lifting of a differentiable function $f : X \to \mathbb{R}^m$ on open $X \subset \mathbb{R}^n$ to a function $\widehat{f}$ on a set $X \times \mathbb{R}^n \subset \mathcal{D}^n$ may be considered as the application of the functor T to X, $\mathbb{R}^m$ and f. In other words,

$$\widehat{f} : X \times \mathbb{R}^n \to \mathcal{D}^m \quad = \quad T(f) : TX \to T\mathbb{R}^m \quad = \quad T\left(f : X \to \mathbb{R}^m\right).$$

In summary, the Forward Mode of AD may also be studied from viewpoints of Differential Geometry and Category Theory. This fact may be used to generalise the concept of Forward AD to functions operating on differentiable manifolds other than subsets of $\mathbb{R}^n$.

8 Higher-Order Partial Differentiation

8.1 Truncated Polynomial Algebras

The computation of higher-order partial derivatives of a sufficiently often and automatically differentiable function $f_j : U \to \mathbb{R}$ on open $U \subset \mathbb{R}^n$ can, for example, be achieved through the extension of f_j to a function defined on a truncated polynomial algebra.[12] This approach has, for instance, been described by Berz[13] in [1] with further elaborations to be found in [5] and the work in [17] and [20] extending the idea (for the two latter, see the following subsection).

Indeed, the extension of f_j to a function on dual numbers as given in definition (4.1) can already be considered in the context of truncated polynomials, since $\mathcal{D} \cong \mathbb{R}[X]/(X^2)$.

Let now $\mathbb{N} \ni n, N \geq 1$ and consider the algebra $\mathbb{R}[X_1, ..., X_n]/I_N$, where

$$I_N := \left(\left\{X_1^{k_1} \cdots X_n^{k_n} \mid (k_1, ..., k_n) \in \mathbb{N}^n \text{ with } k_1 + \cdots + k_n > N\right\}\right)$$

is the ideal generated by all monomials of order $N + 1$. Then

$$\mathbb{R}[X_1, ..., X_n]/I_N \cong \left\{ \sum_{k_1+\cdots+k_n=0}^{N} x_{(k_1,...,k_n)} X_1^{k_1} \cdots X_n^{k_n} \mid x_{(k_1,...,k_n)} \in \mathbb{R} \right\}$$

consists of all polynomials[14] in $X_1, ..., X_n$ of degree $\leq N$.

Denote

$$\mathfrak{f}_i := x_{i,(0,...,0)} + \sum_{k_1+\cdots+k_n=1}^{N} x_{i,(k_1,...,k_n)} X_1^{k_1} \cdots X_n^{k_n}$$

and, for simplicity, identify $x_i := x_{i,(0,...,0)}$. Let further $U \subset \mathbb{R}^n$ be open and define

$$\begin{aligned} &(\mathbb{R}[X_1, ..., X_n]/I_N)_U^n \\ &:= \left\{(\mathfrak{f}_1, ..., \mathfrak{f}_n) \in (\mathbb{R}[X_1, ..., X_n]/I_N)^n \mid (x_1, ..., x_n) \in U\right\}. \end{aligned}$$

[12] We denote the domain of definition by U here to avoid confusion with indeterminates which we denote by X or X_i.

[13] Berz actually used a slightly different, but equivalent approach than the one presented here (see the end of this subsection).

[14] Which, of course, can be identified with tuples consisting of their coefficients.

That is, $(\mathbb{R}[X_1,...,X_n]/I_N)_U^n$ consists of vectors of polynomials in $\mathbb{R}[X_1,...,X_n]/I_N$ with the property that the vector consisting of the trailing coefficients lies in U.

We now define an extension of an N-times differentiable, real-valued function $h : U \to \mathbb{R}$ to a function $\widehat{\widehat{h}} : (\mathbb{R}[X_1,...,X_n]/I_N)_U^n \to \mathbb{R}[X_1,...,X_n]/I_N$ via

$$\widehat{\widehat{h}}(\mathfrak{f}_1,...,\mathfrak{f}_n) := \sum_{k_1+\cdots+k_n=0}^{N} \frac{(\mathfrak{f}_1 - x_1)^{k_1} \cdots (\mathfrak{f}_n - x_n)^{k_n}}{k_1! \cdots k_n!} \frac{\partial^{k_1+\cdots+k_n} h}{\partial x_1^{k_1} \cdots \partial x_n^{k_n}}(x_1,...,x_n) \tag{8.1}$$

In other words, $\widehat{\widehat{h}}$ is the unique function with the property that the images of any $(\mathfrak{f}_1,...,\mathfrak{f}_n)$ under $\widehat{\widehat{h}}$ and its N-th degree Taylor polynomial $T_N(\widehat{\widehat{h}};(x_1,...,x_n))$ about $(x_1,...,x_n)$ are equal.

The reason for this definition becomes apparent when we apply $\widehat{\widehat{h}}$ to a vector of polynomials of the form $(x_1 + X_1,...,x_n + X_n)$. Then

$$\widehat{\widehat{h}}(x_1 + X_1,...,x_n + X_n) = h(\mathbf{x}) + \sum_{k_1+\cdots+k_n=1}^{N} \frac{1}{k_1! \cdots k_n!} X_1^{k_1} \cdots X_n^{k_n} \frac{\partial^{k_1+\cdots+k_n} h}{\partial x_1^{k_1} \cdots \partial x_n^{k_n}}(\mathbf{x}),$$

for $\mathbf{x} = (x_1,...,x_n)$.

One can now compute partial derivatives of order N of a sufficiently often and automatically differentiable function $f_j : U \to \mathbb{R}$ by implementing (8.1) for all elementary functions φ_i. This leads to the extension of f_j to $\widehat{\widehat{f_j}}$ and one obtains a partial derivative $\frac{\partial^{k_1+\cdots+k_n} f_j}{\partial x_1^{k_1} \cdots \partial x_n^{k_n}}(\mathbf{c})$ at a given $\mathbf{c} = (c_1,...,c_n) \in U$ as the $(k_1! \cdots k_n!)$-th multiple of the coefficient of $X_1^{k_1} \cdots X_n^{k_n}$ in $\widehat{\widehat{f_j}}(c_1 + X_1,...,c_n + X_n)$.

Of course, to show that this method actually works, one needs to prove an analogue of Proposition 4.2. However, we will omit the proof here.

Obviously, this method requires the computation of the N-th Taylor polynomial, or, equivalently, of the Taylor coefficients up to degree N, of each elementary function φ_i appearing in f_j. The complexity of this problem is discussed in detail in [9, pages 306–308]: If no restrictions on an elementary function φ_i is given, even in the uni-variate case, order-N^3 arithmetic operations may be required. However, in practice all elementary functions φ_i are solutions of linear ODEs which reduces the computational costs of their Taylor coefficients to $kN^2 + \mathcal{O}(N)$ for $k \in \{1,2,3\}$ (see [9, (13.7) and Proposition 13.1]).

We further remark that Berz in [1], instead of $\mathbb{R}[X_1,...,X_n]/I_N$, actually uses the algebra

$$\left\{ \sum_{k_1+\cdots+k_n=0}^{N} x_{(k_1,...,k_n)} \frac{X_1^{k_1}}{k_1!} \cdots \frac{X_n^{k_n}}{k_n!} \mid x_{(k_1,...,k_n)} \in \mathbb{R} \right\},$$

where multiplication is defined via

$$\left(\sum_{r_1+\cdots+r_n=0}^{N} x_{(r_1,...,r_n)} \frac{X_1^{r_1}}{r_1!} \cdots \frac{X_n^{r_n}}{r_n!} \right) \cdot \left(\sum_{s_1+\cdots+s_n=0}^{N} y_{(s_1,...,s_n)} \frac{X_1^{s_1}}{s_1!} \cdots \frac{X_n^{s_n}}{s_n!} \right) \tag{8.2}$$

$$= \sum_{k_1+\cdots+k_n=0}^{N} (k_1+\cdots+k_n)! \cdot z_{(k_1,...,k_n)} \frac{X_1^{k_1}}{k_1!} \cdots \frac{X_n^{k_n}}{k_n!}. \tag{8.3}$$

for $z_{(k_1,...,k_n)} := \left(\sum_{(r_1,...,r_n)+(s_1,...,s_n)=(k_1,...,k_n)} x_{(r_1,...,r_n)} y_{(s_1,...,s_n)} \right)$. This obviously makes no real difference to the theory, the main advantage is that after lifting a function h to this algebra, one can extract partial derivatives directly, without the need to multiply with $k_1! \cdots k_n!$.

8.2 Differential Algebra and Lazy Evaluation

Differential algebra is an area which has originally been developed to provide algebraic tools for the study of differential equations (see, for example, the original work by Ritt [24] or the introductory article [12]). In the context of Automatic Differentiation it was utilized in [17], [16].

A *differential algebra* is an algebra $\mathcal{A}$ with a mapping $\delta : \mathcal{A} \to \mathcal{A}$ called a *derivation*, which satisfies

$$\delta(a+b) = \delta(a) + \delta(b) \quad \text{and} \quad \delta(a \cdot b) = \delta(a) \cdot b + a \cdot \delta(b),$$

for all $a, b \in \mathcal{A}$. If $\mathcal{A}$ is a field, it is called a *differential field*. Examples for differential fields or algebras are the set of (real or complex) rational functions with any partial differential operator or polynomial algebras $\mathbb{R}[X_1, ..., X_n]$ with a formal partial derivative.Truncated polynomial algebras as described in the previous section can be made into differential algebras as well, however, as Garczynski in [5] points out, a formal (partial) derivative is not a derivation in that case. (For instance, the mapping $D \cdot X : \mathbb{R}[X]/(X^2) \to \mathbb{R}[X]/(X^2)$ with $(D \cdot X)(x + x'X) = x'X$ is a derivation, but the formal derivative D with $D(x + x'X) = x'$ is not.)

Karczmarczuk describes now in [17] a system in which, through a *lazy evaluation*, to each object $a \in \mathcal{A}$ in a differential field $\mathcal{A}$, the sequence

$$(\delta^n(a))_{n\in\mathbb{N}} = (a, \delta(a), \delta^2(a), \delta^3(a), ...)$$

is assigned. In the case of an infinitely many times differentiable univariate function $h : J \to \mathbb{R}$, on open $J \subset \mathbb{R}$, and the differential operator as derivation, this obviously gives $(h, h', h'', h''', ...)$.

The set $\{(\delta^n(a))_{n\in\mathbb{N}} \mid a \in \mathcal{A}\}$ now forms a differential algebra itself, where addition is defined entry-wise, multiplication is given by

$$\begin{aligned}
&(\delta^n(a))_{n\in\mathbb{N}} \cdot (\delta^n(b))_{n\in\mathbb{N}} := (\delta^n(a \cdot b))_{n\in\mathbb{N}} \\
&= \big(a \cdot b, a \cdot \delta(b) + \delta(a) \cdot b, a \cdot \delta^2(b) + 2(\delta(a) \cdot \delta(b)) + \delta^2(a) \cdot b, \\
&\quad a \cdot \delta^3(b) + 3(\delta^2(a) \cdot \delta(b)) + 3(\delta(a) \cdot \delta^2(b)) + \delta^3(a) \cdot b, ...\big) \\
&= \left(\sum_{k_a+k_b=n} \frac{n!}{k_a!k_b!} \delta^{k_a}(a) \cdot \delta^{k_b}(b) \right)_{n\in\mathbb{N}} \qquad (8.4)
\end{aligned}$$

and the derivation, denoted by df, is the right-shift operator, given by

$$df(a, \delta(a), \delta^2(a), \delta^3(a), ...) = (\delta(a), \delta^2(a), \delta^3(a), \delta^4(a), ...),$$

for all $a, b \in \mathcal{A}$. (These definition are given recursively in the original work; for implementation details, see [17, Subsections 3.2–3.3].)

To utilize these ideas for Automatic Differentiation, the assignment $\varphi_i \mapsto (\delta^n(\varphi_i))_{n\in\mathbb{N}}$ is implemented for all uni-variate infinitely many times differentiable elementary functions $\varphi_i : J_i \to \mathbb{R}$, defined on open $J_i \subset \mathbb{R}$, and the elementary functions $+$, $*$ and $/$ are replaced by addition, multiplication and division[15] on the $(\delta^n(\varphi_i))_{n\in\mathbb{N}}$. This then generates for any univariate automatically differentiable function $f : J \to \mathbb{R}$, on open $J \subset \mathbb{R}$, which is constructable by the φ_i, $+$, $*$ and $/$, the sequence $(f, f', f'', f''', ...)$. The N-the derivative of f is then the first entry (which is distinguished in the implementation in [17]) in $df^N(f, f', f'', f''', ...)$.

However, since a differential algebra/field is an abstract concept, this approach can, in principle, be applied to other objects than differentiable functions

We only remark that Karczmarczuk briefly describes in [15] a generalisation to the multi-variate case. Kalman in [14] also constructs a system which appears to be similar.

Comparing (8.4) with (8.2) shows that the multiplication on $\{(\delta^n(a))_{n\in\mathbb{N}} \mid a \in \mathcal{A}\}$ is identical to the multiplication on the algebra used by Berz. Hence, one can express the described system, at least in the case of $a = h$ being a function, in terms of polynomial algebras. That is, consider the truncated algebra $\left\{\sum_{k=0}^N x_k \frac{X^k}{k!} \mid x_k \in \mathbb{R}\right\}$ with multiplication as in (8.2) and define an extension of an $h : J \to \mathbb{R}$ to a mapping

$$\widehat{\widehat{h}} : \left\{ \mathfrak{f} = x + \sum_{k=1}^N x_k \frac{X^k}{k!} \mid x \in J, x_k \in \mathbb{R} \right\} \to \left\{ \sum_{k=0}^N x_k \frac{X^k}{k!} \mid x_k \in \mathbb{R} \right\}$$

via

$$\widehat{\widehat{h}}(\mathfrak{f}) := \sum_{k=0}^N (\mathfrak{f} - x)^k \, \frac{1}{k!} \frac{d^k h}{dx^k}(x),$$

[15]We omit the description of how division on $\{(\delta^n(a))_{n\in\mathbb{N}} \mid a \in \mathcal{A}\}$ is defined here, which can be found in the original publication.

for all $\mathfrak{f} \in \left\{\mathfrak{f} = x + \sum_{k=1}^{N} x_k \frac{X^k}{k!} \mid x \in J, x_k \in \mathbb{R}\right\}$. Then, for all $x \in J$,

$$\widehat{\widehat{h}}(x+X) = h(x) + \sum_{k=1}^{N} \frac{X^k}{k!} \frac{d^k h}{dx^k}(x),$$

which corresponds to the tuple $(h(x), h'(x), h''(x), ..., h^{(N)}(x))$. The lazy evaluation technique in [17] increases the degree of truncation N successively *ad infinitum* to compute any entry of the sequence $(h(x), h'(x), h''(x), h'''(x), ...)$ for any $x \in J$.

Similarly, Pearlmutter and Siskind describe in [20] the lifting of a multivariate function $h : U \to \mathbb{R}$ to a function on $(\mathbb{R}[X_1, ..., X_n]/I_N)_U^n$ as defined in (8.1), and then, through a lazy evaluation, increase the degree of truncation N successively.[16] This computes any entry of the sequence $\left(\frac{\partial^{k_1+\cdots+k_n} h}{\partial x_1^{k_1} \cdots \partial x_n^{k_n}}(\mathbf{c})\right)_{(k_1,...,k_n)\in\mathbb{N}^n}$ (given in some order), for any vector $\mathbf{c} \in U$.

9 The Reverse Mode of AD

Let, as before, $f : X \to \mathbb{R}^m$ on open $X \subset \mathbb{R}^n$ be automatically differentiable. As already mentioned, the Reverse Mode of Automatic Differentiation evaluates products of the Jacobian of f with row vectors. That is, it computes

$$\overleftharpoon{\mathbf{y}} \cdot J_f(\mathbf{c}), \qquad \text{for fixed } \mathbf{c} \in X \text{ and } \overleftharpoon{\mathbf{y}} \in \mathbb{R}^{1\times m}.$$

To our knowledge, there exists currently no method to achieve this computation, which resembles Forward AD using dual numbers. Instead, an elementary approach, similar to the Forward AD approach in Section 3 will have to suffice. The Reverse Mode is, for example, described in [8], [9] and [21]. We follow mainly the discussion in [8].

Express again f as the composition $P_Y \circ \Phi_\mu \circ \cdots \circ \Phi_1 \circ P_X$, with P_X, P_Y and the Φ_i as in Section 3. The computation of $\overleftharpoon{\mathbf{y}} \cdot J_f(\mathbf{c})$ is, by the chain rule, the evaluation of the product

$$\begin{aligned} \overleftharpoon{\mathbf{y}} \cdot J_f(\mathbf{c}) &= \overleftharpoon{\mathbf{y}} \cdot P_Y \cdot \Phi'_{\mu,\mathbf{c}} \ \cdots \ \Phi'_{1,\mathbf{c}} \cdot P_X \\ \Leftrightarrow \qquad J_f(\mathbf{c})^T \cdot \overleftharpoon{\mathbf{y}}^T &= P_X^T \cdot \Phi'^T_{1,\mathbf{c}} \ \cdots \ \Phi'^T_{\mu,\mathbf{c}} \cdot P_Y^T \cdot \overleftharpoon{\mathbf{y}}^T, \end{aligned} \tag{9.1}$$

where again $\Phi'_{i,\mathbf{c}}$ denotes the Jacobian of Φ_i at $(\Phi_{i-1} \circ \cdots \circ \Phi_1 \circ P_X)(\mathbf{c})$.

Obviously, the sequence of state vectors $\mathbf{v}^{[i]} \in H = \mathbb{R}^{n+\mu}$ is the same as in the Forward Mode case. (Here, μ is, of course, again the total number of elementary functions which make up the function f.) The difference lies in the computation of the evaluation trace of (9.1), which we denote by $\overline{\mathbf{v}}^{[\mu]} = \overline{\mathbf{v}}^{[\mu]}(\mathbf{c}, \overleftharpoon{\mathbf{y}}), ..., \overline{\mathbf{v}}^{[0]} = \overline{\mathbf{v}}^{[0]}(\mathbf{c}, \overleftharpoon{\mathbf{y}})$.

[16] Pearlmutter and Siskind actually seem to use the ideal $(X_1^{N+1}, ..., X_n^{N+1})$ instead of I_N, which makes no real difference.

For simplicity, assume $P_Y(v_1, ..., v_{n+\mu}) = (v_{n+\mu-m}, ..., v_{n+\mu})$, for all $(v_1, ..., v_{n+\mu}) \in H$ and denote $\overleftarrow{\mathbf{y}}^T = (y'_1, ..., y'_m)$. We define the evaluation trace of (9.1) as

$$\overline{\mathbf{v}}^{[\mu]} := P_Y^T \cdot \overleftarrow{\mathbf{y}}^T = (0, ..., 0, y'_1, ..., y'_m) \quad \text{and} \quad \overline{\mathbf{v}}^{[i-1]} := \Phi'^T_{i,\mathbf{c}} \cdot \overline{\mathbf{v}}^{[i]}.$$

By (3.2),

$$\Phi'^T_{i,\mathbf{c}} = \begin{array}{c} \quad\quad\quad (n+i)\text{-th column} \\ \quad\quad\quad \downarrow \\ \begin{pmatrix} 1 & \cdots & 0 & \frac{\partial\varphi_i}{\partial v_1}(\cdots) & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & & \vdots \\ 0 & \cdots & 1 & \vdots & 0 & \cdots & 0 \\ 0 & \cdots & 0 & \vdots & 1 & \cdots & 0 \\ \vdots & & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \frac{\partial\varphi_i}{\partial v_{n+\mu}}(\cdots) & 0 & \cdots & 1 \end{pmatrix} \end{array},$$

where $\frac{\partial\varphi_i}{\partial v_k}(\cdots) = \frac{\partial\varphi_i}{\partial v_k}(v_{i_1}(\mathbf{c}), ..., v_{i_{n_i}}(\mathbf{c}))$ is interpreted as 0 if φ_i does not depend on v_k, and the $v_{i_1}(\mathbf{c}), ..., v_{i_{n_i}}(\mathbf{c}) \in \{\mathbf{v}_1^{[i-1]}(\mathbf{c}), ..., \mathbf{v}_{n+i-1}^{[i-1]}(\mathbf{c})\}$.

Therefore, each $\overline{\mathbf{v}}^{[i-1]}$ is of the form

$$\overline{\mathbf{v}}^{[i-1]} = \begin{pmatrix} \overline{\mathbf{v}}_{n+i}^{[i]} \cdot \frac{\partial\varphi_i}{\partial v_1}(\cdots) + \overline{\mathbf{v}}_1^{[i]} \\ \vdots \\ \overline{\mathbf{v}}_{n+i}^{[i]} \cdot \frac{\partial\varphi_i}{\partial v_{n+i-1}}(\cdots) + \overline{\mathbf{v}}_{n+i-1}^{[i]} \\ \overline{\mathbf{v}}_{n+i}^{[i]} \cdot \frac{\partial\varphi_i}{\partial v_{n+i}}(\cdots) \\ \overline{\mathbf{v}}_{n+i}^{[i]} \cdot \frac{\partial\varphi_i}{\partial v_{n+i+1}}(\cdots) + \overline{\mathbf{v}}_{n+i+1}^{[i]} \\ \vdots \\ \overline{\mathbf{v}}_{n+i}^{[i]} \cdot \frac{\partial\varphi_i}{\partial v_{n+\mu}}(\cdots) + \overline{\mathbf{v}}_{n+\mu}^{[i]} \end{pmatrix}$$

$$= \overline{\mathbf{v}}_{n+i}^{[i]} \cdot \begin{pmatrix} \frac{\partial\varphi_i}{\partial v_1}(\cdots) \\ \vdots \\ \frac{\partial\varphi_i}{\partial v_{n+i-1}}(\cdots) \\ \frac{\partial\varphi_i}{\partial v_{n+i}}(\cdots) \\ \frac{\partial\varphi_i}{\partial v_{n+i+1}}(\cdots) \\ \vdots \\ \frac{\partial\varphi_i}{\partial v_{n+\mu}}(\cdots) \end{pmatrix} + \begin{pmatrix} \overline{\mathbf{v}}_1^{[i]} \\ \vdots \\ \overline{\mathbf{v}}_{n+i-1}^{[i]} \\ 0 \\ \overline{\mathbf{v}}_{n+i+1}^{[i]} \\ \vdots \\ \overline{\mathbf{v}}_{n+\mu}^{[i]} \end{pmatrix} \tag{9.2}$$

The value $\overleftarrow{\mathbf{y}} \cdot J_f(\mathbf{c})$ is then given by

$$\left(\overleftarrow{\mathbf{y}} \cdot J_f(\mathbf{c})\right)^T = P_X^T \cdot \overline{\mathbf{v}}^{[0]}.$$

If we let $\overline{\varphi}_i : H \to \mathbb{R}$ be an extension of $\varphi_i : U_i \to \mathbb{R}$ to H with $\frac{\partial \overline{\varphi}_i}{\partial v_k} = 0$ if φ_i does not depend on v_k, and define $\overline{\mathbf{v}}^{[i,*]} \in H$ by $\overline{\mathbf{v}}_k^{[i,*]} = \overline{\mathbf{v}}_k^{[i]}$, for $k \neq n+i$, and $\overline{\mathbf{v}}_{n+i}^{[i,*]} = 0$, then we can rewrite (9.2) as

$$\overline{\mathbf{v}}^{[i-1]} = \left(\overline{\mathbf{v}}_{n+i}^{[i]} \cdot \nabla \overline{\varphi}_i(v_1, ..., v_{n+\mu})\right)^T + \overline{\mathbf{v}}^{[i,*]}.$$

The expression on the right is the analogue of the term

$$\nabla \varphi_i(v_{i_1}, ..., v_{i_{n_i}}) \cdot \begin{pmatrix} v'_{i_1} \\ \vdots \\ v'_{i_{n_i}} \end{pmatrix},$$

which appears in the process of Forward AD, where the main difference is the appearance of the added vector $\overline{\mathbf{v}}^{[i,*]}$.

Note that, in contrast to Forward AD, the sequence of evaluation trace pairs $[\mathbf{v}^{[i]}, \overline{\mathbf{v}}^{[i]}]$ appears in reverse order (that is, $[\mathbf{v}^{[\mu]}, \overline{\mathbf{v}}^{[\mu]}], ..., [\mathbf{v}^{[1]}, \overline{\mathbf{v}}^{[1]}]$). In particular, unlike to Forward AD, it is not efficient to overwrite the previous pair in each computational step. Indeed, since the state vector $\mathbf{v}^{[i]}$ is needed to compute $\overline{\mathbf{v}}^{[i]}$, the pairs $[\mathbf{v}^{[i]}, \overline{\mathbf{v}}^{[i]}]$ are not computed (as pairs) at all. Instead, one first evaluates the evaluation trace $\mathbf{v}^{[1]}, ..., \mathbf{v}^{[\mu]}$, stores these values, and then uses them to compute the $\overline{\mathbf{v}}^{[\mu]}, ..., \overline{\mathbf{v}}^{[1]}$ afterwards.

Example 9.1. Consider the function

$$f : \mathbb{R} \to \mathbb{R}^2, \quad \text{with } \ f(x) = \begin{pmatrix} x \\ \exp(x) * \sin(x) \end{pmatrix}.$$

We want to determine $(y'_1 \quad y'_2) \cdot J_f(c)$ for fixed $\overleftarrow{\mathbf{y}} = (y'_1 \quad y'_2) \in \mathbb{R}^{1\times 2}$ and $\mathbf{c} = c \in \mathbb{R}$.

Set $H = \mathbb{R}^5$ and $f = P_Y \circ \Phi_4 \circ \Phi_3 \circ \Phi_2 \circ \Phi_1 \circ P_X$ with

$$P_X : \mathbb{R} \to \mathbb{R}^5, \quad \text{with } \ P_X(x) = (x, 0, 0, 0, 0),$$

$$\Phi_1 : \mathbb{R}^5 \to \mathbb{R}^5, \quad \text{with } \ \Phi_1(v_1, v_2, v_3, v_4, v_5) = (v_1, \exp(v_1), v_3, v_4,),$$
$$\Phi_2 : \mathbb{R}^5 \to \mathbb{R}^5, \quad \text{with } \ \Phi_2(v_1, v_2, v_3, v_4, v_5) = (v_1, v_2, \sin(v_1), v_4, v_5),$$

$$\Phi_3 : \mathbb{R}^5 \to \mathbb{R}^5, \quad \text{with } \ \Phi_3(v_1, v_2, v_3, v_4, v_5) = (v_1, v_2, v_3, v_1, v_5),$$

$$\Phi_4 : \mathbb{R}^5 \to \mathbb{R}^5, \quad \text{with } \ \Phi_4(v_1, v_2, v_3, v_4, v_5) = (v_1, v_2, v_3, v_4, v_2 * v_3),$$

$$P_Y : \mathbb{R}^5 \to \mathbb{R}, \quad \text{with } \ P_Y(v_1, v_2, v_3, v_4, v_5) = (v_4, v_5).$$

Clearly, we obtain the evaluation trace $\mathbf{v}^{[0]}(c), ..., \mathbf{v}^{[4]}(c)$ with

$$\mathbf{v}^{[0]}(c) = \begin{pmatrix} c \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, ..., \mathbf{v}^{[4]}(c) = \begin{pmatrix} c \\ \exp(c) \\ \sin(c) \\ c \\ \exp(c) * \sin(c) \end{pmatrix}.$$

The Reverse Mode of Automatic Differentiation produces now the vectors $\overline{\mathbf{v}}^{[4]}, ..., \overline{\mathbf{v}}^{[0]}$ with

$$\overline{\mathbf{v}}^{[4]} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ y_1' \\ y_2' \end{pmatrix}, \overline{\mathbf{v}}^{[3]} = \begin{pmatrix} 0 \\ y_2' \cdot \sin(c) \\ y_2' \cdot \exp(c) \\ y_1' \\ 0 \end{pmatrix}, \overline{\mathbf{v}}^{[2]} = \begin{pmatrix} y_1' \\ y_2' \sin(c) \\ y_2' \exp(c) \\ 0 \\ 0 \end{pmatrix},$$

$$\overline{\mathbf{v}}^{[1]} = \begin{pmatrix} y_2' \exp(c) \cdot \cos(c) + y_1' \\ y_2' \sin(c) \\ 0 \\ 0 \\ 0 \end{pmatrix} \text{ and finally}$$

$$\overline{\mathbf{v}}^{[0]} = \begin{pmatrix} y_2' \sin(c) \cdot \exp(c) + \left(y_2' \exp(c) \cos(c) + y_1'\right) \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Then

$$(y_1' \;\; y_2') \cdot J_f(c) = P_X^T \cdot \overline{\mathbf{v}}^0 = y_1' + y_2' \exp(c)(\sin(c) + \cos(c)).$$

We summarize:

Theorem 9.2. *By the above, given $\mathbf{c} \in X \subset \mathbb{R}^n$ and $\overleftarrow{\mathbf{y}} \in \mathbb{R}^{1 \times m}$, the evaluation of $\overleftarrow{\mathbf{y}} \cdot J_f(\mathbf{c})$ for an automatically differentiable function $f : X \to \mathbb{R}^m$ can be achieved by computing the vectors $\mathbf{v}^{[0]}, ..., \mathbf{v}^{[\mu]}$ and $\overline{\mathbf{v}}^{[\mu]}, ..., \overline{\mathbf{v}}^{[0]}$, where the computation of each $\overline{\mathbf{v}}^{[i-1]}$ is effectively the computation of the real numbers*

$$\overline{\mathbf{v}}_{n+i}^{[i]} \cdot \frac{\partial \varphi_i}{\partial v_k}(v_{i_1}(\mathbf{c}), ..., v_{i_{n_i}}(\mathbf{c})) + \overline{\mathbf{v}}_k^{[i]}, \quad k \neq n+i,$$

$$\textit{and} \qquad \overline{\mathbf{v}}_{n+i}^{[i]} \cdot \frac{\partial \varphi_i}{\partial v_{n+i}}(v_{i_1}(\mathbf{c}), ..., v_{i_{n_i}}(\mathbf{c})).$$

Comparing the complexity of Reverse AD with the one of symbolic differentiation for the examples given in Section 5 gives the same result as the comparison of Forward AD with symbolic differentiation. (See, for example, Figure 2 for the case of a composition of 3 uni-variate functions.)

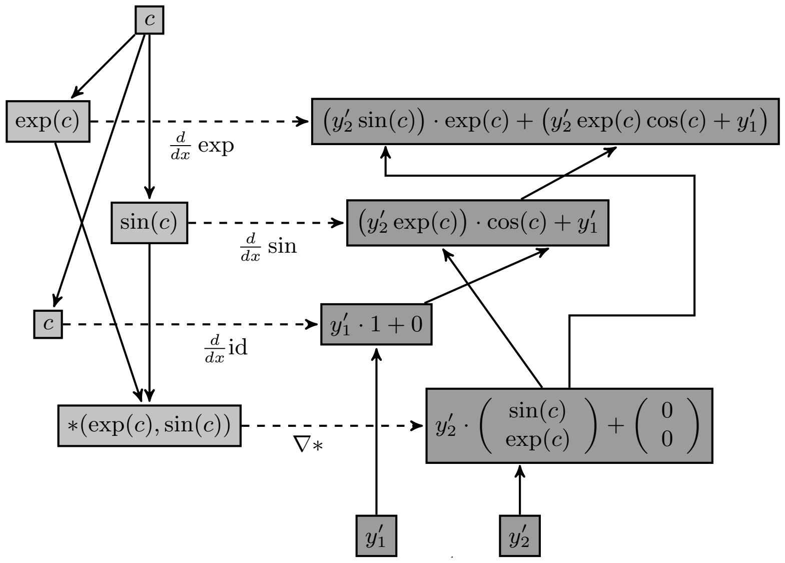

c
$\exp(c)$
$\sin(c)$
c
$*(\exp(c),\sin(c))$
$\frac{d}{dx}\exp$
$\frac{d}{dx}\sin$
$\frac{d}{dx}\mathrm{id}$
$\nabla *$
$\big(y_2'\sin(c)\big)\cdot\exp(c)+\big(y_2'\exp(c)\cos(c)+y_1'\big)$
$\big(y_2'\exp(c)\big)\cdot\cos(c)+y_1'$
$y_1'\cdot 1+0$
$y_2'\cdot\begin{pmatrix}\sin(c)\\ \exp(c)\end{pmatrix}+\begin{pmatrix}0\\0\end{pmatrix}$
y_1'
y_2'

Figure 9: Computational graph for Example 9.1 with elements of $\mathbf{v}^{[4]}$ in blue and the evaluation of the directional derivative in red.

Acknowledgements: I like to thank Felix Filozov for his advice and his help with the code in Figure 6, Atılım Güneş Baydin for his advice and his help with the code in Figure 10 and Barak Pearlmutter for general valuable advice. I am further very thankful to the anonymous referees for valuable criticism and suggestions which led to a significant improvement of this paper.

Funding: This work was supported by Science Foundation Ireland grant 09/IN.1/I2637.

```
// triple D holding entries of $\mathbf{v}^{[j]}(c)$, $\overline{\mathbf{v}}^{[j]}(c)$ and history
type D =
    // Primal (entries of $\mathbf{v}^{[j]}(c)$ )
    val P:float
    // 'Adjoint' (entries of $\overline{\mathbf{v}}^{[j]}(c)$ )
    val A:float ref
    // history of Primal
    val T:TraceOp
    new (p) = {P = p; A = ref 0.; T = NoOp}
    new (p, t) = {P = p; A = ref 0.; T = t}
    static member (*) (x:D, y:D) = D (x.P * y.P, MulOp
       (x, y))
    static member Sin (x:D) = D (sin x.P, SinOp(x))
    static member Exp (x:D) = D (exp x.P, ExpOp(x))

and TraceOp = | MulOp of D * D | ExpOp of D | SinOp of
    D | NoOp

// Helper functions
let primal (x:D) = x.P
let adjoint (x:D) = !x.A

// computation of $\overline{\mathbf{v}}^{[j]}(c)$
let rec reverse (v:D) a =
    v.A := !v.A + a // update entry of $\overline{\mathbf{v}}^{[j]}(c)$
    match v.T with
    | MulOp(u1, u2) ->
        reverse u2 (a * u1.P)
        reverse u1 (a * u2.P)
    | ExpOp(u) -> reverse u (a * exp u.P)
    | SinOp(u) -> reverse u (a * cos u.P)
    | NoOp -> ()

// Transposed Jacobian-vector product
let jacobianTv (f:D[]->D[]) (c:float[]) (y':float[]) =
    // Convert the input values from float into D
    let cd = Array.map D c
    // Run forward evaluation, compute output values
    let y = f cd
    // start reverse evaluation
    Array.iter2 reverse y y'
    // Return $\overline{\mathbf{v}}^{[0]}(c)$
    Array.map adjoint cd

// Test case
let f (x:D[]) = [|x.[0]; (exp x.[0]) * (sin x.[0])|]

let test = jacobianTv f ([|5.|]) ([|1.; 1.|])
```

Figure 10: Minimal (not optimal!) F# example, similar to work in the library DiffSharp [10], [11], showing the implementation of the Reverse Mode for Example 9.1 with the test case $c = 5$, $\overleftarrow{\mathbf{y}} = (1\ 1)$. (An optimal version would account for fan-out at each node.)

References

[1] M. Berz, Differential algebraic description of beam dynamics to very high orders, *Particle Accelerators* 24 (1989), 109–124.

[2] C. H. Bischof. On the Automatic Differentiation of Computer Programs and an Application to Multibody Systems, *IUTAM Symposium on Optimization of Mechanical Systems Solid Mechanics and its Applications* 43 (1996), 41–48.

[3] W. K. Clifford, Preliminary Sketch of Bi-quaternions, *Proceedings of the London Mathematical Society* 4 (1873), 381–395.

[4] L. Dixon, Automatic Differentiation: Calculation of the Hessian, In: Encyclopedia of Optimization, second edition, Springer Science+Business Media, LLC., 2009, 133–137.

[5] V. Garczynski, Remarks on differential algebraic approach to particle beam optics by M. Berz, *Nuclear Instruments and Methods in Physics Research Section A*, 334 (2-3) (1993), 294–298.

[6] R. M. Gower, M. P. Mello, A new framework for the computation of Hessians, *Optimization Methods and Software* 27 (2) (2012), 251–273.

[7] A. Griewank, Achieving Logarithmic Growth of Temporal and Spatial Complexity in Reverse Automatic Differentiation, *Optimization Methods and Software* 1 (1992), 35–54.

[8] A. Griewank, A Mathematical View of Automatic Differentiation, *Acta Numerica* 12 (2003), 321–398.

[9] A. Griewank and A. Walther, Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation, second edition, SIAM, Philadelphia, PA, 2008.

[10] A. G. Baydin, B. A. Pearlmutter, DiffSharp: Automatic Differentiation Library, version of 17th June 2015, `http://diffsharp.github.io/DiffSharp/`.

[11] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, J. M. Siskind, Automatic differentiation and machine learning: a survey. arXiv preprint. arXiv:1502.05767 (2015).

[12] J. H. Hubbard and B. E. Lundell, A First Look at Differential Algebra, *American Mathematical Monthly* 118 (3) (2011), 245–261.

[13] F. John, Partial Differential Equations, second edition, Springer-Verlag, New York-Heidelberg-Berlin, 1975.

[14] D. Kalman, Double Recursive Multivariate Automatic Differentiation, *Mathematics Magazine* 75 (3) (2002), 187–202.

[15] J. Karczmarczuk, Functional Coding of Differential Forms, *Proc First Scottish Workshop on Functional Programming, Stirling, Scotland, 1999.*

[16] J. Karczmarczuk, Functional Differentiation of Computer Programs, *Proc of the III ACM SIGPLAN International Conference on Functional Programming, Baltimore, MD, 1998*, 195–203.

[17] J. Karczmarczuk, Functional Differentiation of Computer Programs, *Higher-Order and Symbolic Computation* 14 (1) (2001), 35–57.

[18] S. Lang, Introduction to Differentiable Manifolds, second edition, Springer-Verlag, New York-Heidelberg-Berlin, 2002.

[19] O. Manzyuk, A Simply Typed λ-Calculus of Forward Automatic Differentiation, *Electronic Notes in Theoretical Computer Science* 286 (2012), 257–272.

[20] B. A. Pearlmutter and J. M. Siskind, Lazy Multivariate Higher-Order Forward-Mode AD, *Proc of the 2007 Symposium on Principles of Programming Languages, Nice, France, 2007*, 155–160.

[21] B. A. Pearlmutter and J. M. Siskind, Reverse-Mode AD in a Functional Framework: Lambda the Ultimate Backpropagator, *TOPLAS* 30 (2) (2008), 1–36.

[22] L. B. Rall, Differentiation and generation of Taylor coefficients in Pascal-SC, In: A New Approach to Scientific Computation, Academic Press, New York, 1983, 291–309.

[23] L. B. Rall, The Arithmetic of Differentiation, *Mathematics Magazine* 59, (1986), 275–282.

[24] J. Ritt, Differential Algebra, American Mathematical Society Colloquium Publications, vol. XXXIII, American Mathematical Society, New York, 1950.

[25] J. M. Siskind and B. A. Pearlmutter, Nesting Forward-Mode AD in a Functional Framework, *Higher-Order and Symbolic Computation* 21 (4) (2008),361–376.

[26] R. Wengert, A Simple Automatic Derivative Evaluation Program, *Communications of the ACM* 7 (8) (1964), 463–464.

Philipp Hoffmann
National University of Ireland, Maynooth
Department of Computer Science
Maynooth, County Kildare
Ireland
email: philipp.hoffmann@cs.nuim.ie
philip.hoffmann@maths.ucd.ie