COMPUTING OF B-SERIES BY AUTOMATIC DIFFERENTIATION

FERENC BARTHA AND HANS Z. MUNTHE-KAAS

1. Introduction

B-series, named after John C. Butcher, is a fundamental tool for the study of numerical integration methods [4, 14]. B-series are mainly used as a theoretical tool systematising the analysis of numerical integration schemes. In recent years B-series methods have also been used more directly as a computational tool. The method of modified vector fields [6] is based on the idea that the results of a given numerical integration scheme can be improved by modifying the vector field prior to feeding it into the numerical integrator. As an example, given a differential equation $y' = f(y)$ and the second order implicit midpoint rule, one may compute a modified vector field $\tilde{f}$ such that when $\tilde{f}$ is fed into the midpoint rule, the result is a higher order approximation to the original differential equation $y' = f(y)$. The modified vector field $\tilde{f}$ can be expressed as a B-series in f, and the computation of $\tilde{f}$ involves the computation of derivatives of f. A very special example is the method of modified vector fields applied to Eulers explicit method. In this case $\tilde{f}$ is given as a Taylor series expansion of f up to the desired order. More generally, Runge–Kutta like methods based on computing terms in the B-series are called *elementary differential Runge–Kutta methods* [17].

Previous work on modified vector fields have mainly been applied to special dynamical systems where the derivatives of f are analytically computable up to a certain order. For more general systems the analytical computation of higher order derivatives is intractable by analytical means, due to a combinatorial explosion in the number of terms. An important alternative to analytical differentiation is the use of Automatic Differentiation (AD) techniques [12]. Such techniques are applicable whenever there is a need to compute higher order derivatives in given points, and analytical expressions are not needed. In the special case of Taylor series integration methods, AD techniques have successfully been applied to compute Taylor series up to order 50 or more [21].

The main purpose of the present paper is to investigate AD techniques and software for the computation of general B-series, and hereby opening up the applicability of modified vector field methods and elementary differential Runge-Kutta methods for general classes of dynamical systems.

1.1. Pre-Lie algebras, trees and B-series.

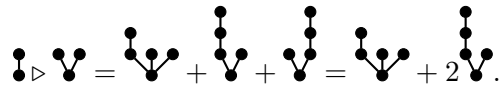
B-series can generally be defined in terms of a *pre-Lie algebra*. A pre-Lie algebra is a vector space $\mathcal{A}$ with a bilinear product $\triangleright\colon \mathcal{A}\times\mathcal{A} \to \mathcal{A}$. The product is neither commutative nor associative, but satisfies the *pre-Lie relation*

$$a(x, y, z) = a(y, x, z),$$

where $a(x,y,z) = (x \triangleright y) \triangleright z - x \triangleright (y \triangleright z)$ is the *associator* of the product. Our prime example of a pre-Lie algebra is the set of vector fields on $\mathbb{R}^n$, denoted $\Xi(\mathbb{R}^n)$. This consist of all smooth functions $f\colon \mathbb{R}^n \to \mathbb{R}^n$ with the pre-Lie product given for $f, g \in \Xi(\mathbb{R}^n)$ as

$$(f \triangleright g)(y) = \frac{\partial}{\partial s}\bigg|_{s=0} g(y + sf(y)). \tag{1.1}$$

An other important example of a pre-Lie algebra is the set of all rooted trees T, endowed with the product $\triangleright$ given by *grafting*. The grafting $t_1 \triangleright t_2$ is a sum of all possible ways of attaching the root of t_1 to a node of t_2, as in this example:



The pre-Lie algebra $\{T, \triangleright\}$ is called the *free pre-Lie algebra*, since it has a universality property as follows [5]. For any pre-Lie algebra $\mathcal{A}$ and any map $\bullet \mapsto f\colon T \to \mathcal{A}$ there exists a unique pre-Lie morphism $\mathcal{F}_f\colon T \to \mathcal{A}$, defined as a map such that $\mathcal{F}_f(\bullet) = f$ and $\mathcal{F}_f(t_1 \triangleright t_2) = \mathcal{F}_f(t_1) \triangleright \mathcal{F}_f(t_2)$ for all trees $t_1, t_2 \in T$. In numerical analysis $\mathcal{F}_f(t) \in \Xi(\mathbb{R}^n)$ are called *elementary differentials*. The evaluation of all elementary differentials can be done in different ways. Based on the homomorphism property of $\mathcal{F}_f$, we can build all the elementary differentials from the *monomial basis* for the free pre-Lie algebra. The first of these are $\bullet$, $\bullet \triangleright \bullet$, $(\bullet \triangleright \bullet) \triangleright \bullet$, $\bullet \triangleright (\bullet \triangleright \bullet)$, $(\bullet \triangleright \bullet) \triangleright (\bullet \triangleright \bullet)$. The monomial basis is indexed by binary trees. Rooted trees are obtained from binary trees by a recursion known as Knuth's rotation correspondence [9]. The first elementary differentials can be obtained from Knuth's correspondence as

$$\begin{aligned}
F_f(\bullet) &= f \\
F_f(\,\bullet\!\!-\!\!\bullet\,) &= F_f(\bullet \triangleright \bullet) = F_f(\bullet) \triangleright \mathcal{F}_f(\bullet) = f \triangleright f \\
F_f(\,\bullet\!\!-\!\!\bullet\!\!-\!\!\bullet\,) &= F_f(\,\bullet\!\!-\!\!\bullet \triangleright \bullet) = (f \triangleright f) \triangleright f \\
F_f(\,\bullet\!\!-\!\!\bullet\!\!<\!\!\bullet\,) &= F_f(\bullet \triangleright \,\bullet\!\!-\!\!\bullet - \bullet) = f \triangleright (f \triangleright f) - (f \triangleright f) \triangleright f.
\end{aligned}$$

A more conventional expression for the elementary differentials of rooted trees is directly in terms of higher order derivatives of f, as given in (2.5) below.

A B-series is traditionally defined as a formal finite or infinite sum

$$B_f(\beta) = \sum_{t \in T} \frac{h^{|t|}}{\sigma(t)} \beta(t) \mathcal{F}_f(t)$$

where $|t|$ denotes the number of nodes in t, $\sigma(t) \in \mathbb{Z}$ is the tree symmetry function, $h \in \mathbb{R}$ is a real parameter (e.g. step size of a numerical method) and $\beta\colon T \to \mathbb{R}$ is a given function. The parameter h can be absorbed by a re-scaling of the vector field $f \mapsto hf$ and σ can be absorbed into β. Therefore we adopt the simplified definition

$$B_f(\beta) = \sum_{t \in T} \beta(t) \mathcal{F}_f(t) \in \Xi(\mathbb{R}^n). \tag{1.2}$$

Our goal is efficient computation of finite B-series, i.e. series where $\beta(t) = 0$ for all $|t| > N$, using automatic differentiation.

2. An algorithm for computing B-series

In this section we will present a way to calculate B-series. This problem consists of multiple parts. The first is the generation of the rooted trees, for this we use the algorithm of Li and Ruskey [16]. Then in order to evaluate $\mathcal{F}_f$ over a tree we take advantage of Automatic Differentiation (AD, see in Griewank [12]) in our algorithm. One of the most important characteristics of AD is that we are not working with the formulas for derivatives, but the values at a given point. We compute higher order Taylor coefficients of univariate functions with automatic differentiation, and through interpolation from these we obtain the necessary derivatives using the formula by Griewank, Utke and Walther [13]. Finally one has to take care of the redundancy when evaluating $\mathcal{F}_f$ over all the trees up to a given degree. We handle this phenomenon by basing our calculation on weighted directed acyclic graphs instead of rooted trees.

2.1. **Generating rooted trees.**

We need an efficient way to generate all the rooted trees up to degree d, taking care about the isomorphic equivalences. We use the algorithm of Li and Ruskey [16], which we will review here.

There are multiple ways to represent a rooted tree, we will use the parent array concept. Suppose that we have a rooted tree τ whose vertices are labeled as $1, 2, \ldots, d$. Then the ith entry of the parrent array $par_\tau[i]$ is 0 if the vertex with the label i is the root, otherwise it is the label of the parent of the vertex labeled with i. In order to obtain a unique parent array associated with a given rooted tree τ, we label the vertices in the depth-first order starting from the root. See Figure 1 for a labeling of a rooted tree with 9 vertices.

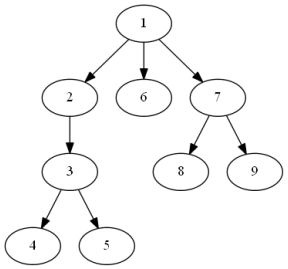


Figure 1. The depth-first labeling of τ gives $par_\tau = \langle 0, 1, 2, 3, 3, 1, 1, 7, 7 \rangle$

We need a representative element from each isomorphic equivalence class. We obtain these with the use of the parent array. The representative element from an equivalence class of the rooted trees is the rooted tree τ that has the lexicographically maximal parent array. This rooted tree is called canonic and we also say that the parent array is canonic. It is straightforward that removing the vertex with the highest label from a canonic tree with d vertices results in a canonic tree with $d - 1$ vertices. On Figure 2 we compare

the parent arrays of two isomorphic rooted trees τ_1 and τ_2. Since τ_2 has lexicographically bigger parent array, it is the canonic tree in the class consisting these two trees.

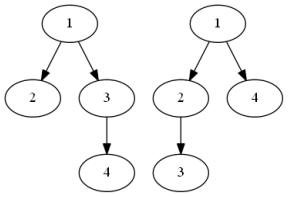


FIGURE 2. Two isomorphic trees with $par_{\tau_1} = \langle 0, 1, 1, 3\rangle < par_{\tau_2} = \langle 0, 1, 2, 1\rangle$

The algorithm from [16] generates canonic parent arrays up to a given size. Moreover it does so in constant time, amortized over all trees. This is the so-called CAT-property (Constant Amortized Time). The recursive procedure is based on extending a canonic tree τ of $d-1$ vertices into a canonic tree with d vertices.

The vertex with label d is a leaf by construction, and comes later than the vertex labeled with $d-1$ in the depth-first order. Therefore the parent of d is an ancestor of $d-1$ or the vertex $d-1$ itself.

Lemma 2.1. *Let par_τ be the parrent array of the canonic rooted tree τ with $d-1$ vertices. If the parrent array $\langle par_\tau[1], \ldots, par_\tau[d-1], \eta\rangle$ is canonic and $\eta \neq 1$, then the parent array $\langle par_\tau[1], \ldots, par_\tau[d-1], par_\tau[\eta]\rangle$ is canonic as well.*

Proof. Suppose that the parent array $\langle par_\tau[1], \ldots, par_\tau[d-1], par_\tau[\eta]\rangle$, and the respective tree τ', are not canonic. Let the canonic tree in the class of τ' be τ''. Consider the following operation on the tree τ''. Move the vertex that corresponds to d with respect to the labeling of τ' to be the child of the vertex that corresponds to η in the labeling of τ'. We obtain a lexicographically bigger parent array than $\langle par_\tau[1], \ldots, par_\tau[d-1], \eta\rangle$ and this is a contradiction. □

Lemma 2.1 suggests that we shall find the ancestor μ of $d-1$, for which $\langle par_\tau[1], \ldots, par_\tau[d-1], par_\tau[\mu]\rangle$ is canonic and μ is the lowest such index. This is exactly what Algorithm 1 does.

Algorithm 1 Generation of canonic rooted trees

```
1: procedure GEN(λ, ρ, S; par, T)   ▷ par[.] is a canonic parent array with λ − 1 entries
2:     if (λ > d) then                                              ▷ the size of par is d
3:         move par into T                                          ▷ save par into the list T
4:     else                                  ▷ copy the next vertex of the subtree rooted at ρ
5:         if L = 0 then
6:             par[λ] ← λ − 1                                       ▷ the first tree
7:         else if par[λ − S] < ρ then
8:             par[λ] ← par[ρ]                                      ▷ copying root
9:         else
10:            par[λ] ← S + par[λ − S]                              ▷ copying non-root
11:        end if
12:        Gen(λ + 1, ρ, S; par, T)                    ▷ continue the copying repeatedly
13:        while par[λ] > 1 do                  ▷ finding the next subtree to be copied
14:            ρ ← par[λ]
15:            par[λ] ← par[ρ]
16:            Gen(λ + 1, ρ, λ − ρ; par, T)          ▷ start the copying of this subtree
17:        end while
18:    end if
19: end procedure
```

For a rooted tree τ and its vertex labeled with q let $T_\tau(q)$ be the rooted subtree of τ rooted at q. In Algorithm 1 we copy repeatedly the subtree with the root ρ. We store the degree of this subtree in S. After Algorithm 1 is called, it produces all the canonic parent arrays of the form $\langle par[1], \dots, par[p-1], \cdot \rangle$. We start the generation by $\texttt{Gen}(1, 0, 0; par = \langle\rangle, T = \emptyset)$.

Assume now that τ is a rooted tree with $d-1$ vertices and let μ be the ancestor of $d-1$ that satisfies that the parent array of $T_\tau(\mu)$ is a prefix of the parent array of $T_\tau(\nu)$ where ν is the rightmost proper sibling of μ, and μ is the lowest such index. We call critical node the leftmost sibling ρ of ν that satisfies that $T_\tau(\rho) = T_\tau(\nu)$. On Figure 3 we have a tree with 16 vertices. We see that $\mu = 15$, $\nu = 12$ and $\rho = 6$.

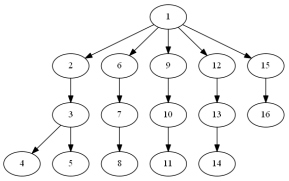


FIGURE 3. A tree with 16 vertices, the critical node is 6.

In order to see that Algorithm 1 is correct, let τ be a rooted tree with $d-1$ vertices and consider the lexicographically largest infinite canonic rooted tree agreeing with τ in the first $d-1$ positions of its parent array. This infinite tree may be formed by deleting $T_\tau(\mu)$,

and replacing it with an infinite sequence of rooted subtrees, all isomorphic to $T_\tau(\rho)$ and their root connected to $par_\tau(\rho)$ $(= par_\tau(\nu) = par_\tau(\mu))$. Truncating this infinite tree at any node $D \geq d$ gives the lexicographically largest extension of D nodes. We start a new copy of $T_\tau(\rho)$ in line 8 of Algorithm 1, while in line 10 we make a new isomorphic copy with offset S.

2.2. **Computing higher order derivatives.** Let $f\colon \mathbb{R}^n \to \mathbb{R}^n$ be sufficiently smooth function such that all appearing derivatives exist and are continuous. Our goal is to evaluate higher order derivatives of f. For this purpose one may use higher order automatic differentiation techniques for multivariate functions, the reader is referred to Berz [3], Danis [8]. We shall present here a different method described in Griewank, Utke and Walther [13].

Fix the integer $p \geq 1$ for the time being. We shall not denote explicitly that the dimension of a quantity is dependent on p, it is rather straightforward to use the formulas with different p-s later on.

Let $\mathbf{i} = (\mathbf{i}_1, \ldots, \mathbf{i}_p) \in \mathbb{N}_0^p$ be a *multi-index* with the norm $|\mathbf{i}|$ defined as $|\mathbf{i}| = \sum_{r=1}^p \mathbf{i}_r$. The multi-indices $\mathbf{i}$ and $\mathbf{j}$ satisfy $\mathbf{j} \leq \mathbf{i}$ if the relation is satisfied componentwise. Consequently $\mathbf{j} < \mathbf{i}$ is true if $\mathbf{j} \leq \mathbf{i}$ and $\mathbf{j} \neq \mathbf{i}$ stand. We denote by $\mathbf{0}$ and $\mathbf{1}$ the multi-indices that contain only zeros or ones respectively. Naturally a multi-index is a real vector in $\mathbb{R}^p$ as well, therefore we may use them in the standard algebraic operations.

Let $\mathbf{s_r} \in \mathbb{R}^n$ be real vectors for all $r = 1, \ldots, p$. The $\mathbf{S} \in \mathbb{R}^{n \times p}$ matrix that has the column vectors $\mathbf{s_j}$,

$$\mathbf{S} = \left[\ \mathbf{s_1}; \ldots; \mathbf{s_p} \ \right]$$

is called the *seed matrix*.

Our goal is to evaluate $\nabla_{\mathbf{S}}^d f(\mathbf{x})$, that is the d-th derivative tensor of $f(\mathbf{x} + \mathbf{S}\mathbf{z})$ with respect to $\mathbf{z}$ at $\mathbf{z} = \mathbf{0}$. This means that we have to obtain the partial derivatives of the form

$$f_{\mathbf{i}}(\mathbf{x}) = \left. \frac{\partial^{|\mathbf{i}|} f(\mathbf{x} + z_1 \mathbf{s_1} + \ldots + z_p \mathbf{s_p})}{\partial z_1^{\mathbf{i}_1} \ldots \partial z_p^{\mathbf{i}_p}} \right|_{\mathbf{z}=\mathbf{0}}, \tag{2.1}$$

where $\mathbf{i} \in \mathbb{N}_0^p$ is a multi-index with $|\mathbf{i}| = d$.
Let $F_k(\mathbf{x}; \mathbf{v})$ be the k-th Taylor-coefficient of the univariate function

$$f_{\mathbf{x};\mathbf{v}}\colon \mathbb{R} \to \mathbb{R}^n\colon t \mapsto f(\mathbf{x} + t\mathbf{v})$$

at $t = 0$. The quantities $\gamma(\mathbf{i}, \mathbf{j})$ where $\mathbf{i}$ and $\mathbf{j}$ are multi-indices are defined as follows

$$\gamma(\mathbf{i}, \mathbf{j}) = \sum_{\mathbf{0} < \mathbf{k} \leq \mathbf{i}} (-1)^{|\mathbf{i}-\mathbf{k}|} \binom{\mathbf{j}}{\mathbf{k}} \binom{|\mathbf{j}|\mathbf{k}/|\mathbf{k}|}{\mathbf{j}} \left(\frac{|\mathbf{k}|}{|\mathbf{j}|} \right)^{|\mathbf{i}|}. \tag{2.2}$$

Now we can state the main formula for higher derivative tensors from [12] and [13] that we used in our implementation. If $d \geq |\mathbf{i}| > 0$, then

$$\left. \frac{\partial^{|\mathbf{i}|} f(\mathbf{x} + z_1 \mathbf{s_1} + \ldots + z_p \mathbf{s_p})}{\partial z_1^{\mathbf{i}_1} \ldots \partial z_p^{\mathbf{i}_p}} \right|_{z=0} = \sum_{|j|=d} \gamma(\mathbf{i}, \mathbf{j}) F_{|\mathbf{i}|}(\mathbf{x}; \mathbf{S}\mathbf{j}). \tag{2.3}$$

According to the analysis done in [13], the operation count of this propagation of multiple univariate Taylor series in comparison with the operation count of the propagation of multivariate derivatives is essentially the same for moderate degrees and considerably fewer for higher degrees.

2.3. **Computing $\mathcal{F}_f$ over a rooted tree τ at $\mathbf{x}$.** The quantity $\mathcal{F}_f(\tau)$ is a vectorfield. Note that we want to obtain its value $\mathcal{F}_f(\tau)(\mathbf{x})$ at the point $\mathbf{x} \in \mathbb{R}^n$. For the tree with one vertex we have

$$\mathcal{F}_f(\bullet)(\mathbf{x}) = f(\mathbf{x}). \tag{2.4}$$

Assume that the rooted tree τ is a root connected to the subtrees: $\tau_1, \ldots, \tau_p$. In this situation the recursive formula for $\mathcal{F}_f(\tau)(\mathbf{x})$ is written as

$$\mathcal{F}_f(\tau)(\mathbf{x}) = \sum_{\mathbf{j} \in \{1,2,\ldots,n\}^p} \frac{\partial^p f}{\partial x_{\mathbf{j}_1} \ldots \partial x_{\mathbf{j}_p}}(\mathbf{x}) \mathcal{F}_f^{\mathbf{j}_1}(\tau_1)(\mathbf{x}) \ldots \mathcal{F}_f^{\mathbf{j}_p}(\tau_p)(\mathbf{x}). \tag{2.5}$$

Having isomporphic trees $\tau_1 \cong \tau_2$ results in $\mathcal{F}_f(\tau_1)(\mathbf{x}) = \mathcal{F}_f(\tau_2)(\mathbf{x})$.

We may consider formula (2.5) as a higher derivative tensor and obtain the closed expression

$$\mathcal{F}_f(\tau)(\mathbf{x}) = \left. \frac{\partial^p f\left(\mathbf{x} + z_1 F_f(\tau_1)(\mathbf{x}) + \ldots + z_p F_f(\tau_p)(\mathbf{x})\right)}{\partial z_1 \ldots \partial z_p} \right|_{z=0}. \tag{2.6}$$

We can convert this into the form of (2.3) by introducing $\mathbf{i} = (1, \ldots, 1) \in \mathbb{N}_+^p$ and $\mathbf{s_r} = F_f(\tau_r)(\mathbf{x})$ for $r \in \{1, \ldots, p\}$. However, if τ has symmetries on the first level, that is there are identical – up to isomorphism – trees between $\tau_1, \ldots, \tau_p$, exploiting this will result in a more efficient computation.

Assume now that the root of τ has the following subtrees $\tau_{1,1} \ldots \tau_{1,\mathbf{i}_1}; \ldots; \tau_{p,1} \ldots \tau_{p,\mathbf{i}_p}$ where $\tau_{r,1} \cong \ldots \cong \tau_{r,\mathbf{i}_r}$ for all $r \in \{1, \ldots, p\}$. Let τ_r be the canonic tree in the equivalency class that contains $\tau_{r,1}, \ldots, \tau_{r,\mathbf{i}_r}$. Using these, equation (2.6) now takes the somewhat familiar form

$$\mathcal{F}_f(\tau)(\mathbf{x}) = \left. \frac{\partial^{|\mathbf{i}|} f(\mathbf{x} + z_1 F_f(\tau_1)(\mathbf{x}) + \ldots + z_p F_f(\tau_p)(\mathbf{x}))}{\partial z_1^{\mathbf{i}_1} \ldots \partial z_p^{\mathbf{i}_p}} \right|_{z=0}. \tag{2.7}$$

The seed matrix in this case is

$$\mathbf{S} = \left[F_f(\tau_1)(\mathbf{x}); \ldots; F_f(\tau_p)(\mathbf{x})\right].$$

As the recursive definition shows, in order to evaluate $\mathcal{F}_f$ over the rooted tree τ, first we have to evaluate the children of the root and for that, their children etc.

Let us assume that τ has d vertices and those are $\mathcal{V}(\tau) = \{v_1, \ldots, v_d\}$. Let $\mathbf{i}$ be a permutation of $\{1, \ldots, d\}$. The ordering $\{v_{\mathbf{i}_1}, \ldots, v_{\mathbf{i}_d}\}$ of the vertices is called *post-ordering* if for every vertex $v_{\mathbf{i}_j}$ it is true that if $v_{\mathbf{i}_k}$ is a child of $v_{\mathbf{i}_j}$, then $\mathbf{i}_k < \mathbf{i}_j$. In order to have simpler notations let us assume that $\{v_1, \ldots, v_d\}$ is already in post-order. Note that this implies that v_1 is a leaf, and v_d is the root.

Recall that $T_\tau(q)$ denotes the rooted subtree of τ rooted at q. Consider the following sequence of rooted trees $T_\tau(v_1), \ldots, T_\tau(v_d)$. When computing $\mathcal{F}_f(T_\tau(v_j))(\mathbf{x})$, because of

the post-ordering of the vertices, the rows of the seed matrix are amongst the vectors $\mathcal{F}_f(T_\tau(v_k))(\mathbf{x})$ with $k < j$. Therefore we may evaluate the values

$$\mathcal{F}_f(T_\tau(v_1))(\mathbf{x}),\ \ldots\ ,\ \mathcal{F}_f(T_\tau(v_d))(\mathbf{x})$$

in this order, using equations (2.3) and (2.7). Observe that $T_\tau(v_d) = \tau$, thus we have a way to compute $\mathcal{F}_f(\tau)(\mathbf{x})$.

2.4. **Computing $\mathcal{F}_f$ over multiple rooted trees.** Our goal now is to evaluate $\mathcal{F}_f$ over all the rooted trees up to degree d. Consider that the list T of these rooted trees is ordered as follows. The rooted tree $\tau_1 \in T$ preceeds the rooted tree $\tau_2 \in T$ if the degree of τ_1 is smaller, or if they have the same degree, then if the parent array of τ_1 is lexicographically smaller. It is obvious that if τ' is a rooted subtree of τ, then τ' comes before τ in the ordering.

We wish to evaluate $\mathcal{F}_f$ over all the trees in the ordered list T. We shall progress from smaller order to higher order, thus we start with the tree with one vertex $\bullet$, obtaining $\mathcal{F}_f(\bullet)(\mathbf{x}) = f(\mathbf{x})$. In the rest of this section when we consider a rooted tree $\tau \in T$, we always assume that it has the following structure. The subtrees connecting to the root are $\tau_{1,1}\ldots\tau_{1,\mathbf{i}_1}; \ldots; \tau_{p,1}\ldots\tau_{p,\mathbf{i}_p}$ and $\tau_{r,1}, \ldots, \tau_{r,\mathbf{i}_r}$ are in the same equivalency class with the canonic rooted tree τ_r for all $r \in \{1, \ldots, p\}$. As we have seen with the post-ordering, we may evaluate $\mathcal{F}_f(\tau)(\mathbf{x})$ with applying equation (2.3) once, if $\mathcal{F}_f$ is known for the first level subtrees already. It is obvious that we don't have to repeat the evaluation recursively vertex by vertex, instead we may proceed through the ordered list T. If we have evaluated F_f over the first k trees already, then doing the computation for the $(k+1)$th tree is one application of the formula (2.3).

We may represent every rooted tree in T as a vertex in a weighted Directed Acylcic Graph (wDAG) in the following way. Let $\mathcal{G}$ be a wDAG with vertices $\mathcal{V}$ and edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. We denote the weight of an edge e by $\mathbf{w}(e)$. The weight of the vertex $v \in \mathcal{V}$ is the sum of the weights of the edges leaving v, that is

$$\mathbf{w}(v) = \sum_{u:(v,u)\in\mathcal{E}} \mathbf{w}((v,u)). \tag{2.8}$$

We say that $\mathcal{G}$ represents the ordered list T of rooted trees if there exists a bijective function $\iota : T \to \mathcal{V}$ such that if $\tau \in T$, then

$$\{v \in \mathcal{V} : (\iota(\tau), v) \in \mathcal{E}\} = \{\iota(\tau_r) : r \in \{1, \ldots, p\}\} \tag{2.9}$$

and $\mathbf{w}((\iota(\tau), \iota(\tau_r))) = \mathbf{i}_r$ for all $r \in \{1, \ldots, p\}$. On Figure 4 we added the wDAG generated by our program representing the rooted trees with degree less than 6.

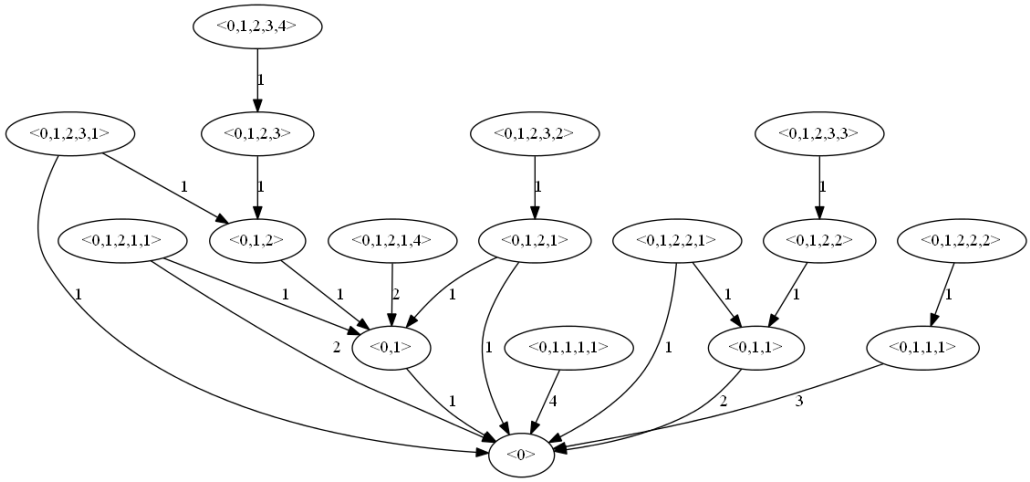


FIGURE 4. $\mathcal{G}$ representing the rooted trees with degree less than 6

It is evident that $\mathcal{G}$ has only one sink, that is only one vertex with no outgoing edge and this is $\iota(\bullet)$. Define the function $\Delta : \mathcal{V} \to \mathbb{R}^n$ as follows,

$$\Delta(v) = \begin{cases} f(\mathbf{x}) & \text{if } v = \iota(\bullet), \\ \left. \dfrac{\partial^{\mathbf{w}(v)} f\left(\mathbf{x} + \sum_{r=1}^{p} z_r \Delta(u_r)\right)}{\partial z_1^{\mathbf{w}((v,u_1))} \dots \partial z_p^{\mathbf{w}((v,u_p))}} \right|_{z=0} & \text{if } \{u \in \mathcal{V} : (v,u) \in \mathcal{E}\} = \{u_1, \dots, u_p\}. \end{cases} \tag{2.10}$$

We may progress with the evaluation of Δ over the vertices of $\mathcal{G}$ in a post-ordering of $\mathcal{V}$. It is obvious that Δ is computable and for any $\tau \in T$ we have

$$\mathcal{F}_f(\tau)(\mathbf{x}) = \Delta(\iota(\tau)). \tag{2.11}$$

2.5. **Computational cost.** An important aspect of our algorithm is that the dimension n of the vector field f enters *linearly* into the cost of the evaluation of the B-series, thus the algorithm is tractable also for high dimensional vector fields. The operation cost of evaluating a d'th order univariate Taylor-coefficients is of the order $\mathcal{O}(d^2)$ according to Griewank [12], thus the d'th order Taylor coefficient $\partial^d / \partial z^d f(\mathbf{x} + z\mathbf{s})$ is found in only $\mathcal{O}(nd^2)$ operations. This compares favorably with the exponential growth observed for the symbolic differentiation of a general function.

Griewank, Utke and Walther [13] analysed the formula (2.3) and found that the number of nonvanishing coefficients

$$0 \neq \gamma(\mathbf{i}, \mathbf{j}) \text{ for } \mathbf{i}, \mathbf{j} \in \mathbb{N}_0^p, 1 \leq |\mathbf{i}| \leq d, |\mathbf{j}| = d, \tag{2.12}$$

is less than or equal to

$$\#(d,p) = \sum_{m=1}^{d} \left(\begin{array}{c} p \\ m \end{array} \right) \left(\begin{array}{c} d \\ m \end{array} \right) \left(\begin{array}{c} m+d-1 \\ d \end{array} \right). \tag{2.13}$$

Given a seed matrix, this number bounds the number of operations needed to calculate all partial derivatives from the univariate Taylor series.

As seen in Plotkin and Rosenthal [19] and Finch [11], if T_d is the number of non-isomorphic rooted trees with n vertices, then

$$T_d \sim r^{-d} d^{3/2} \left(0.4399240125... + \frac{0.0441699018...}{d} + \frac{0.2216928059...}{d^2} + \frac{0.8676554908...}{d^3} + \dots \right), \tag{2.14}$$

where $r = 0.3383218568...$ is the unique positive root of the equation $F(x, 1) = 0$ for

$$F(x, y) = x \exp\left(y + \sum_{k=2}^{\infty} \frac{T(x^k)}{k} \right) - y, \tag{2.15}$$

and $T(x) = \sum_{j=1}^{\infty} T_j x^j$ is the generating function for $\{T_d\}$. Recall that a rooted tree is represented in the wDAG as a vertex with the same weight as the number (d) of the children of the root and with as many children as many (p) non-isomporphic children the root has.

We do not have a precise expression for the total cost of evaluating B-series up to order d with the present algorithm. However, assuming that we need at most T_d different seed matrices, we believe the cost is asymptotically bounded by $\mathcal{O}(nd^2T_d) < \mathcal{O}(nd^{5/2})$ operations.

2.6. **Implementation.** We have written our program in `C++` using the automatic differentiation library `FADBAD++` by Bendtsen and Stauning [2] and the `CAPD` library [7]. For the representation of the graphs we have used the `Boost Graph` library [20] and stored the graphs in Graphviz format [10]. The source code is available at [1].

3. Conclusion and future directions

We have presented a method to efficiently compute B-series using automatic differentiation. While repeatedly using formula (2.3), it becomes apparent, that during the evaluation of the wDAG described in Section 2.4, we have to calculate with the same seed-matrix or submatrix in certain cases. Note that it is also recommended to precompute the values $\gamma(\mathbf{i}, \mathbf{j})$ in advance, possibly with higher accuracy. These gives space for further improvements, that we plan to address in the future.

Whereas computation of d-order B-series for n dimensional vector fields is in general impossible for high d or high n using symbolic derivations, the present algorithm has a complexity which is only linear in n and polynomial in d and thus applicable in a range of practical situations, at least for moderately high order d.

However, an order d B-series is still substantially more expensive to compute than a order d Taylor expansion, using AD. The reason for this is that the Taylor expansion has a sparse representation in the monomial basis, represented by the B-series

$$f + f \triangleright f + \frac{1}{2} f \triangleright (f \triangleright f) + \frac{1}{6} f \triangleright (f \triangleright (f \triangleright f)) + \frac{1}{24} f \triangleright (f \triangleright (f \triangleright (f \triangleright f))) \dots .$$

In other words, the Taylor series is obtained by a repeated differentiation with repspect to t of the function $f(y(t))$, where $y'(t) = f(y(t))$. Thus, the B-series of Taylor expansions can be computed substantially more efficient than general B-series. An alternative to our computational approach, which may be more efficient also for other B-series being sparse in the monomial basis, might be to use (1.1) as a basis for an AD algorithm. The algorithmic details and possible computational advantages of such an approach is subject to future research.

References

[1] Bartha, F. Ad-trees software `http://hans.munthe-kaas.no/work/Projects.html`.

[2] Bendtsen, C., and Stauning, O. FADBAD, a flexible C++ package for automatic differentiation –- using the forward and backward methods. Tech. Rep. IMM-REP-1996-17, TU Denmark, DK-2800 Lyngby, Denmark, 1996.

[3] Berz, M. Algorithms for higher derivatives in many variables with applications to beam physics. In *Automatic differentiation of algorithms (Breckenridge, CO, 1991)*. SIAM, Philadelphia, PA, 1991, pp. 147–156.

[4] Butcher, J. An algebraic theory of integration methods. *Math. Comp 26*, 117 (1972), 79–106.

[5] Chapoton, F., and Livernet, M. Pre-lie algebras and the rooted trees operad. *International Mathematics Research Notices 2001*, 8 (2001), 395–408.

[6] Chartier, P., Hairer, E., Vilmart, G., et al. Numerical integrators based on modified differential equations. *Mathematics of computation 76*, 260 (2007), 1941–1954.

[7] Computer Assisted Proofs in Dynamics group. CAPD Library. http://capd.ii.uj.edu.pl. a C++ package for rigorous numerics.

[8] Danis, A. Thesis: Parameter estimation, set valued numerics – in preparation. *Uppsala University* (2012).

[9] Ebrahimi-Fard, K., and Manchon, D. The magnus expansion, trees and knuth's rotation correspondence. *arXiv preprint arXiv:1203.2878* (2012).

[10] Ellson, J., Gansner, E., Koutsofios, L., North, S., Woodhull, G., Description, S., and Technologies, L. Graphviz — open source graph drawing tools. In *Lecture Notes in Computer Science* (2001), Springer-Verlag, pp. 483–484.

[11] Finch, S. Two asymptotic series. *http://algo.inria.fr/bsolve/*.

[12] Griewank, A. *Evaluating derivatives*, vol. 19 of *Frontiers in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2000. Principles and techniques of algorithmic differentiation.

[13] Griewank, A., Utke, J., and Walther, A. Evaluating higher derivative tensors by forward propagation of univariate Taylor series. *Math. Comp. 69*, 231 (2000), 1117–1130.

[14] Hairer, E., Lubich, C., and Wanner, G. *Geometric numerical integration: structure-preserving algorithms for ordinary differential equations*, vol. 31. Springer, 2006.

[15] Jorba, À., and Zou, M. A software package for the numerical integration of ODEs by means of high-order Taylor methods. *Experiment. Math. 14*, 1 (2005), 99–117.

[16] Li, G., and Ruskey, F. The advantages of forward thinking in generating rooted and free trees (extended abstract). In *IN 10TH ANNUAL ACM-SIAM SYMPOSIUM ON DISCRETE ALGORITHMS (SODA* (1999), pp. 939–940.

[17] Moan, P., Murua, A., Quispel, G., Sofroniou, M., and Spaletta, G. Symplectic elementary differential runge–kutta methods. *preprint* (2004).

[18] Moore, R. E., Davidson, J. A., Jaske, H. R., and Shayer, S. DIFEQ integration routine—user's manual. Tech. Rep. LMSC 6–90–64–6, Lockheed Missiles and Space Co., Palo Alto, CA, 1964.

[19] Plotkin, J. M., and Rosenthal, J. W. How to obtain an asymptotic expansion of a sequence from an analytic identity satisfied by its generating function. *J. Austral. Math. Soc. Ser. A 56*, 1 (1994), 131–143.

[20] Siek, J. G., Lee, L.-Q., and Lumsdaine, A. *The Boost Graph Library User Guide and Reference Manual (With CD-ROM)*. Addison-Wesley Professional, 2001.

[21] Simó, C. On the analytical and numerical approximation of invariant manifolds. *Les Méthodes Modernes de la Mecánique Céleste* (1990), 285–329.

[22] Tucker, W. *Validated numerics*. Princeton University Press, Princeton, NJ, 2011. A short introduction to rigorous computations.