RWTH Aachen

Department of Computer Science
Technical Report

Algorithmic Differentiation of Numerical Methods: Second-Order Tangent and Adjoint Solvers for Systems of Parametrized Nonlinear Equations

Niloofar Safiran, Johannes Lotz, Uwe Naumann

ISSN 0935–3232 · Aachener Informatik-Berichte · AIB-2014-07

RWTH Aachen · Department of Computer Science · Nov. 2014 (revised version)

Algorithmic Differentiation of Numerical Methods: Second-Order Tangent and Adjoint Solvers for Systems of Parametrized Nonlinear Equations

Niloofar Safiran, Johannes Lotz, Uwe Naumann

LuFG Informatik 12: Software and Tools for Computational Engineering,
RWTH Aachen University, Germany.
Email: {`safiran, lotz, naumann`}`@stce.rwth-aachen.de`

Abstract. Forward and reverse modes of algorithmic differentiation (AD) transform implementations of multivariate vector functions $F : I\!R^n \rightarrow I\!R^m$ as computer programs into tangent and adjoint code, respectively. The reapplication of the same ideas yields higher derivative code. In particular, second derivatives play an important role in nonlinear programming. Second-order methods based on Newton's algorithm promise faster convergence in the neighbourhood of the minimum by taking into account second derivative information. The adjoint mode is of particular interest in large-scale gradient-based nonlinear optimization due to the independence of its computational cost on the number of free variables. Solvers for parametrized systems of n equations embedded into the evaluation of the objective function for a (without loss of generality) unconstrained nonlinear optimization problem require the Hessian of the objective with respect to the free variables implying the need for second derivatives of the nonlinear solver. The local computational overhead as well as the additional memory requirement for the computation of second-order tangents or second-order adjoints of the solution vector with respect to parameters by a fully algorithmic method (derived by AD) can quickly become prohibitive for large values of n. Both can be reduced significantly by the second-order symbolic approach to differentiation of the underlying numerical method to be discussed in this paper.

1 Introduction and Summary of the Results

This paper builds on [NLLT12], in which the first-order algorithmic differentiation (AD) [GW08,Nau12] of solvers for systems of nonlinear equations is discussed. In this paper we consider two alternative approaches for evaluating the second derivatives of numerical simulation programs which contain calls to solvers for parameterized systems of n nonlinear equations. The first approach is the *algorithmic version* (derived by AD) of computing second derivatives in which the local computational overhead as well as the additional memory requirement for the computation of second-order tangents or second-order adjoints of the solution vector with respect to the parameters can quickly become prohibitive for large values of n. The second approach is differentiation of the underlying mathematical formulation (*symbolic version*) which computes the derivatives of the solution under the assumption that the exact solution has been reached. Therefore the accuracy of the calculated derivatives depends on the accuracy of the solution, but it reduces the computational complexity by orders of magnitude.

With forward and reverse modes of AD as the two fundamental approaches to the computation of truncation-free first derivatives, there are 4 combinations yielding second derivatives, namely forward over forward (FoF), forward over reverse (FoR), reverse over forward (RoF) and reverse over reverse (RoR) [GW08,Nau12]. In this paper we focus on FoF and FoR for reasons laid out later in this work.

The run time and memory overhead for algorithmic and symbolic approaches to the differentiation of an iterative (e.g. Newton-type) method for the solution of nonlinear systems is shown in Table 1.

	Symbolic		Algorithmic	
	FoF	FoR	FoF	FoR
Memory	$O(n^2)$	$O(n^2)$	$O(n^2)$	$\nu \cdot O(n^3)$
Run Time	$O(n^3)$	$O(n^3)$	$\nu \cdot O(n^3)$	$\nu \cdot O(n^3)$

Table 1. Computational complexity and memory requirement of one projection with second-order algorithmic and symbolic tangent and adjoint modes of differentiation for ν (e.g. Newton) iterations applied to systems of n nonlinear equations.

Computing the second derivatives by a fully algorithmic method corresponds to a straight application of AD without taking into account any mathematical properties of the numerical method. It turns out to be the worst approach in terms of computational efficiency. The performance of the different approaches depends on the number of the iterations ν performed by the nonlinear solver (e.g. Newton steps) and on the problem size n. Any nonlinear solver with a direct linear solver called in each step, which approaches to solution of the nonlinear system, will do the same (e.g. SIMPLE). We do not really rely on Newton as the nonlinear solver. In this paper we refer to Newton's algorithm for parameterized systems of nonlinear equations as an example to show the complexities in algorithmic mode (see Sections 4.1 and 5.1).

2 Foundations

In this section we recall some aspects from [NLLT12]. We consider the computation of second-order tangents (directional derivatives) $\mathbf{x}^{(1,2)} \in I\!R^n$ as well as second-order adjoints $\boldsymbol{\lambda}_{(1)}^{(2)} \in I\!R^m$ for solvers of parametrized systems of nonlinear equations described by the residual

$$\mathbf{r} = F(\mathbf{x}, \boldsymbol{\lambda}) : I\!R^n \times I\!R^m \rightarrow I\!R^n. \tag{1}$$

For a given $\boldsymbol{\lambda} \in I\!R^m$, a vector $\mathbf{x} \in I\!R^n$ is sought such that $F(\mathbf{x}, \boldsymbol{\lambda}) = 0$.

Without loss of generality, the nonlinear solver is assumed to be embedded into the unconstrained convex nonlinear programming problem (NLP)

$$\min_{\mathbf{z} \in I\!R^q} f(\mathbf{z})$$

for a given objective function $f : I\!R^q \rightarrow I\!R$. In the context of second-order derivative-based methods (e.g. Newton) the gradient and the Hessian of $y =$

$f(\mathbf{z}) \in I\!R$ with respect to $\mathbf{z} \in I\!R^q$ need to be computed, which involves the second derivative of the nonlinear solver itself.

As in [NLLT12], f is decomposed as

$$y = f(\mathbf{z}) = p(S(\mathbf{x}^0, P(\mathbf{z}))), \tag{2}$$

where $P : I\!R^q \to I\!R^m$ denotes the part of the computation that precedes the nonlinear solver $S : I\!R^n \times I\!R^m \to I\!R^n$ and where $p : I\!R^n \to I\!R$ maps the result $\mathbf{x}$ onto the scalar objective y.

Most of our arguments will be based on the following algorithmic description of Equation (2)

$$\boldsymbol{\lambda} = P(\mathbf{z}) \tag{3}$$
$$\tilde{\mathbf{x}} = S(\mathbf{x}^0, \boldsymbol{\lambda}) \tag{4}$$
$$y = p(\tilde{\mathbf{x}}). \tag{5}$$

The parameters $\boldsymbol{\lambda} \in I\!R^m$ are computed as functions of $\mathbf{z} \in I\!R^q$ by the given implementation of P. They enter the nonlinear solver S as arguments as well as the given initial estimate $\mathbf{x}^0 \in I\!R^n$ of the solution $\mathbf{x} \in I\!R^n$. Finally, the computed approximation $\tilde{\mathbf{x}}$ of the solution $\mathbf{x}$ is reduced to a scalar objective value $y \in I\!R$ by the given implementation of p. In this paper, P is called the preprocessor, S is called the nonlinear solver and p is called the postprocessor.

As an example for a nonlinear solver we consider Newton's method. A basic version of Newton's algorithm for parameterized systems of nonlinear equations $F(\mathbf{x}, \boldsymbol{\lambda}) = 0$ yields

```
for $i = 0, \ldots, \nu$
    $A := F'(\mathbf{x}^i, \boldsymbol{\lambda}) \equiv \frac{\partial F}{\partial \mathbf{x}}(\mathbf{x}^i, \boldsymbol{\lambda})$        (6)
    $\mathbf{b} := -F(\mathbf{x}^i, \boldsymbol{\lambda})$
    $\mathbf{s} := \mathbf{L}(A, \mathbf{b}) \quad (\Rightarrow A \cdot \mathbf{s} = \mathbf{b})$        (7)
    $\mathbf{x}^{i+1} := \mathbf{x}^i + \mathbf{s}.$        (8)
```

While the symbolic approach does not rely on a specific method for the solution of the nonlinear system, the algorithmic version requires insight into the individual algorithmic steps performed by the nonlinear solver.

3 First- and Higher-Order Algorithmic Differentiation

We mention some significant elements of AD described in further detail in [GW08,Nau12]. Without loss of generality, the following discussion will be based on the residual function in Equation (1). In the following we use the notation from [Nau12] which is partially inspired by the notation used in [GW08]. Let therefore be

$$\mathbf{u} \equiv \begin{pmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{pmatrix} \in I\!R^h$$

and $h = n + m$. AD yields semantical transformations of the given implementation of $F : I\!R^h \rightarrow I\!R^n$ as a computer program into first and potentially also higher (k-th order) derivative code. For this purpose F is assumed to be k times symbolically differentiable for $k = 1, 2, \ldots$.

For AD to become applicable, the given implementation of F is assumed to decompose into a *single assignment code* (SAC) as follows

$$\begin{aligned} &\text{for } j = h, \ldots, h + q + n - 1 \\ &\quad v_j = \varphi_j(v_i)_{i \prec j}, \end{aligned}$$

where $i \prec j$ denotes a direct dependence of v_j on v_i. The result of each *intrinsic function*[1] φ_j is assigned to a unique auxiliary variable v_j. The h *independent inputs* $u_i = v_i$, for $i = 0, \ldots, h - 1$, are mapped onto n *dependent outputs* $r_j = v_{h+q+j}$, for $j = 0, \ldots, n - 1$. The values of q *intermediate variables* v_k are computed for $k = h, \ldots, h + q - 1$.

The SAC induces a directed acyclic graph (DAG) $G = (V, E)$ with integer vertices $V = \{0, \ldots, h + q + n - 1\}$ and edges $E = \{(i, j) | i \prec j\}$. The vertices are sorted topologically with respect to variable dependence inducing a partial order according to $\forall i, j \in V : (i, j) \in E \Rightarrow i < j$.

The intrinsic functions φ_j are assumed to posses jointly symbolic partial derivatives with respect to their arguments. Association of the local partial derivatives with their corresponding edges in the DAG yields a *linearized DAG*. The linearized DAG of our reference objective is shown in Fig. 1 (a) with (high-level) intrinsic functions P, S, and p.

By the chain rule of differential calculus, the entries of the Jacobian $A = (a_{i,j}) \equiv \nabla F(\mathbf{u})$ can be computed as

$$a_{i,j} = \sum_{\pi \in [i \rightarrow h+q+j]} \prod_{(k,l) \in \pi} c_{l,k}, \tag{9}$$

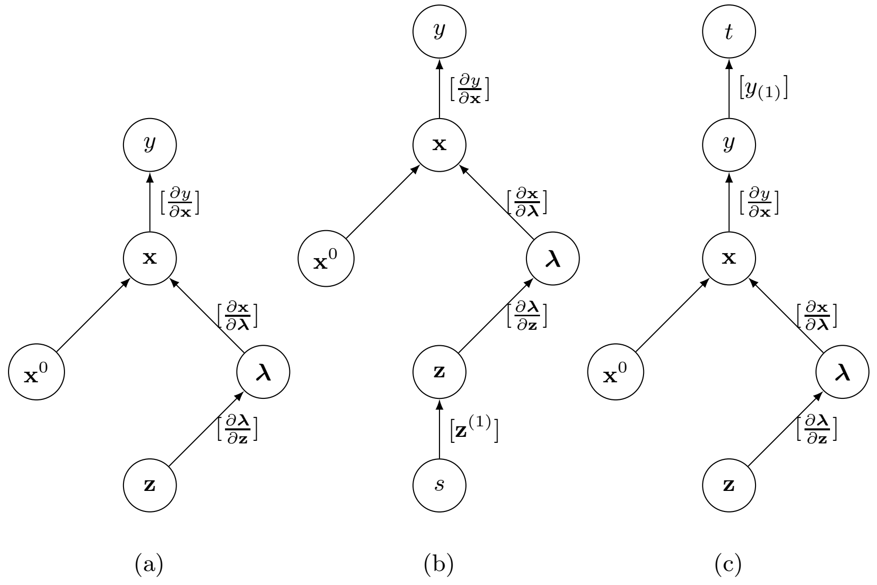
where

$$c_{l,k} \equiv \frac{\partial \varphi_l}{\partial v_k}(v_w)_{w \prec l}$$

and where $[i \rightarrow h+q+j]$ denotes the set of all paths that connect the independent vertex i with the dependent vertex $h + q + j$ [Bau74]. For example, according to Fig. 1 (a)

$$\frac{\partial f}{\partial \mathbf{z}} \equiv \frac{\partial y}{\partial \mathbf{z}} = \frac{\partial p}{\partial \mathbf{x}} \cdot \frac{\partial S}{\partial \boldsymbol{\lambda}} \cdot \frac{\partial P}{\partial \mathbf{z}} = \frac{\partial y}{\partial \mathbf{x}} \cdot \frac{\partial \mathbf{x}}{\partial \boldsymbol{\lambda}} \cdot \frac{\partial \boldsymbol{\lambda}}{\partial \mathbf{z}}.$$

[1] Intrinsic functions can range from fundamental arithmetic operations $(+, *, \ldots)$ and built-in (into the used programming language) functions $(\sin, \exp, \ldots)$ to potentially highly complex numerical algorithms such as routines for interpolation, numerical integration, or the solution of systems of linear or nonlinear equations. In its basic form, AD is defined for the arithmetic operators and built-in functions. A formal extension of this concept to higher-level intrinsics turns out to be reasonably straight forward. For a complex algorithm to become an intrinsic function all we require is the existence of and knowledge about the partial derivatives of its results with respect to its arguments.



y
$[\frac{\partial y}{\partial \mathbf{x}}]$
$\mathbf{x}$
$[\frac{\partial \mathbf{x}}{\partial \boldsymbol{\lambda}}]$
$\mathbf{x}^0$
$\boldsymbol{\lambda}$
$[\frac{\partial \boldsymbol{\lambda}}{\partial \mathbf{z}}]$
$\mathbf{z}$
(a)
y
$[\frac{\partial y}{\partial \mathbf{x}}]$
$\mathbf{x}$
$\mathbf{x}^0$
$[\frac{\partial \mathbf{x}}{\partial \boldsymbol{\lambda}}]$
$\boldsymbol{\lambda}$
$[\frac{\partial \boldsymbol{\lambda}}{\partial \mathbf{z}}]$
$\mathbf{z}$
$[\mathbf{z}^{(1)}]$
s
(b)
t
$[y_{(1)}]$
y
$[\frac{\partial y}{\partial \mathbf{x}}]$
$\mathbf{x}$
$[\frac{\partial \mathbf{x}}{\partial \boldsymbol{\lambda}}]$
$\mathbf{x}^0$
$\boldsymbol{\lambda}$
$[\frac{\partial \boldsymbol{\lambda}}{\partial \mathbf{z}}]$
$\mathbf{z}$
(c)

Fig. 1. Reference Problem: (a) Linearized DAG; (b) Tangent Extension; (c) Adjoint Extension

3.1 First-Order Tangent Model

The Jacobian $\nabla F = \nabla F(\mathbf{u})$ of a multivariate vector function $\mathbf{r} = F(\mathbf{u})$, $F : I\!R^h \to I\!R^n$, induces a linear mapping $I\!R^h \to I\!R^n$, where $h = n + m$, defined by

$$\mathbf{u}^{(1)} \mapsto < \nabla F, \mathbf{u}^{(1)} > \equiv \nabla F(\mathbf{u}) \cdot \mathbf{u}^{(1)} \quad .$$

A first-order tangent projection of ∇F in direction $\mathbf{u}^{(1)} \in I\!R^h$ is defined as the usual matrix-vector product $\nabla F(\mathbf{u}) \cdot \mathbf{u}^{(1)}$. Alternatively, we use the inner product notation $< \nabla F, \mathbf{u}^{(1)} >$ as introduced in [Nau12].

The function $F^{(1)} : I\!R^h \times I\!R^h \to I\!R^n$, defined as

$$\mathbf{r}^{(1)} = F^{(1)}(\mathbf{u}, \mathbf{u}^{(1)}) = < \nabla F, \mathbf{u}^{(1)} > \tag{10}$$

is referred to as the *tangent model* of F. Let $[\nabla F]_{k,j} = \frac{\partial [\mathbf{r}]_k}{\partial [\mathbf{u}]_j} \in I\!R^{n \times h}$ with $k = 0, ..., n-1$ and $j = 0, ..., h-1$ be a 2-tensor (a matrix). Hence, Equation (10) yields

$$[\mathbf{r}^{(1)}]_k = < [\nabla F]_{k,*}, \mathbf{u}^{(1)} > \equiv \sum_{j=0}^{h-1} [\nabla F]_{k,j} \cdot [\mathbf{u}^{(1)}]_j \quad ,$$

for $k = 0, ..., n-1$. The kth row of ∇F is denoted by $[\nabla F]_{k,*}$. The expression $< [\nabla F_{k,*}], \mathbf{u}^{(1)} >$ denotes the usual scalar product of two vectors in $I\!R^h$. This tensor notation will be required for the discussion of higher derivative models in Section 3.3 and the following.

The directional derivatives $\mathbf{r}^{(1)}$ can be regarded as the partial derivative of $\mathbf{r}$ with respect to an auxiliary scalar variable s, where initially

$$\mathbf{u}^{(1)} \equiv \frac{\partial \mathbf{u}}{\partial s} \quad .$$

Interpretation of chain rule on the corresponding linearized DAG (the tangent extension of the original linearized DAG) yields

$$\mathbf{r}^{(1)} \equiv \frac{\partial \mathbf{r}}{\partial s} = \frac{\partial \mathbf{r}}{\partial \mathbf{u}} \cdot \frac{\partial \mathbf{u}}{\partial s} =< \nabla F, \mathbf{u}^{(1)} > .$$

For example, in Figure 1(b) the tangent extension of the linearized DAG of our reference objective is shown. Equation (9) yields $y^{(1)} = \frac{\partial y}{\partial \mathbf{z}} \cdot \mathbf{z}^{(1)} =< \frac{\partial y}{\partial \mathbf{z}}, \mathbf{z}^{(1)} >$. Note that $\frac{\partial y}{\partial \mathbf{z}} \in I\!R^{1\times q}$.

Tangent (also: forward) mode software tools for AD transform a given implementation

```
F(u, r)
```

of Equation $\mathbf{r} = F(\mathbf{u})$ with input u$\hat{=}\mathbf{u}$ and output r $\hat{=}\mathbf{r}$ into the (algorithmic) tangent subroutine

```
t1_F(u, t1_u, r, t1_r)
```

where t1_u $\hat{=}\mathbf{u}^{(1)}$ and t1_r $\hat{=}\mathbf{r}^{(1)}$. A prefix *t*1_ marks 1st-order tangent mode. The Jacobian of the residual with respect to **u** can be accumulated by letting t1_u range over the Cartesian basis vectors in $I\!R^h$. The individual columns of the Jacobian are returned in t1_r while r contains the value of the residual. The complexity of this model for evaluating the whole Jacobian is $O(h) \cdot Cost(F)$.

3.2 First-Order Adjoint Model

The adjoint of a linear operator is its transpose [DS88], since

$$< \mathbf{u}^{(1)}, \mathbf{u}_{(1)} >=< \mathbf{u}^{(1)}, \nabla F^T(\mathbf{u}) \cdot \mathbf{r}_{(1)} >=< \mathbf{r}^{(1)}, \mathbf{r}_{(1)} >=< \nabla F(\mathbf{u}) \cdot \mathbf{u}^{(1)}, \mathbf{r}_{(1)} > .$$

Consequently, the transposed Jacobian $\nabla F^T = \nabla F(\mathbf{u})^T$ of a multivariate vector function $\mathbf{r} = F(\mathbf{u})$, $F : I\!R^h \to I\!R^n$, induces a linear mapping $I\!R^n \to I\!R^h$ defined by

$$\mathbf{r}_{(1)} \mapsto < \mathbf{r}_{(1)}, \nabla F(\mathbf{u}) > \equiv \nabla F(\mathbf{u})^T \cdot \mathbf{r}_{(1)} \quad .$$

A first-order adjoint projection of ∇F in direction $\mathbf{r}_{(1)}$ is defined as the usual matrix-vector product $\nabla F(\mathbf{u})^T \cdot \mathbf{r}_{(1)}$. Alternatively, we use the inner product notation $< \mathbf{r}_{(1)}, \nabla F(\mathbf{u}) >$ as introduced in [Nau12].

The function $F_{(1)} : I\!R^h \times I\!R^n \to I\!R^h$, defined as

$$\mathbf{u}_{(1)} = F_{(1)}(\mathbf{u}, \mathbf{r}_{(1)}) =< \mathbf{r}_{(1)}, \nabla F(\mathbf{u}) > \tag{11}$$

is referred to as the *adjoint model* of F. Let $\nabla F = [\nabla F]_{k,j} = \frac{\partial [\mathbf{r}]_k}{\partial [\mathbf{u}]_j} \in I\!R^{n\times h}$ with $k = 0, ..., n-1$ and $j = 0, ..., h-1$ be a 2-tensor (a matrix). Hence, Equation (11) yields

$$[\mathbf{u}_{(1)}]_j =< \mathbf{r}_{(1)}, [\nabla F]_{*,j} > \equiv \sum_{k=0}^{n-1} [\nabla F]_{k,j} \cdot [\mathbf{r}_{(1)}]_k \quad ,$$

for $j = 0, ..., h-1$. The jth column of ∇F is denoted by $[\nabla F]_{*,j}$. The expression $< \mathbf{r}_{(1)}, [\nabla F]_{*,j} >$ denotes the usual scalar product of two vectors in $I\!R^n$.

Adjoints can be regarded as partial derivatives of an auxiliary scalar variable t with respect to $\mathbf{r}$ and $\mathbf{u}$, where

$$\mathbf{r}_{(1)} \equiv \left(\frac{\partial t}{\partial \mathbf{r}}\right)^T \quad \text{and} \quad \mathbf{u}_{(1)} \equiv \left(\frac{\partial t}{\partial \mathbf{u}}\right)^T .$$

By the chain rule, we get

$$\mathbf{u}_{(1)} \equiv \left(\frac{\partial t}{\partial \mathbf{u}}\right)^T = \left(\frac{\partial \mathbf{r}}{\partial \mathbf{u}}\right)^T \cdot \left(\frac{\partial t}{\partial \mathbf{r}}\right)^T = \nabla F^T \cdot \mathbf{r}_{(1)} \quad .$$

For example, in Figure 1(c) the adjoint extension of the linearized DAG of our reference objective is shown. Equation (9) yields $\mathbf{z}_{(1)} = \frac{\partial y}{\partial \mathbf{z}}^T \cdot y_{(1)} =< y_{(1)}, \frac{\partial y}{\partial \mathbf{z}} >$.

Adjoint (also: reverse) mode software tools for AD transform a given implementation

```
F(u, r)
```

of Equation $\mathbf{r} = F(\mathbf{u})$ with u$\hat{=}\mathbf{u}$ and r $\hat{=}\mathbf{r}$ into the (algorithmic) adjoint subroutine

```
a1_F(u, a1_u, r, a1_r)
```

where a1_u $\hat{=}\mathbf{u}_{(1)}$ and a1_r $\hat{=}\mathbf{r}_{(1)}$. A prefix a1_ marks 1st-order adjoint mode. The Jacobian of the residual with respect to $\mathbf{u}$ can be accumulated by letting a1_r range over the Cartesian basis vectors in $I\!R^n$ while setting a1_u=0. The individual rows of the Jacobian are returned in a1_u. The output argument r contains the value of the residual. The complexity of this model for evaluating the whole Jacobian is $O(n) \cdot Cost(F)$.

3.3 Second-Order Tangent Model

The Hessian $\nabla^2 F = \nabla^2 F(\mathbf{u})$ of a multivariate vector function $\mathbf{r} = F(\mathbf{u})$, $F : I\!R^h \rightarrow I\!R^n$, induces a bilinear mapping $I\!R^h \times I\!R^h \rightarrow I\!R^n$ defined by

$$(\mathbf{u}^{(1)}, \mathbf{u}^{(2)}) \mapsto < \nabla^2 F, \mathbf{u}^{(1)}, \mathbf{u}^{(2)} > = << \nabla^2 F, \mathbf{u}^{(1)} >, \mathbf{u}^{(2)} > .$$

A second-order tangent projection $< \nabla^2 F, \mathbf{u}^{(1)}, \mathbf{u}^{(2)} >$ of a symmetric 3-tensor $\nabla^2 F$, where

$$\nabla^2 F = [\nabla^2 F]_{k,i,j} = \frac{\partial [r]_k}{\partial [\mathbf{u}]_i \partial [\mathbf{u}]_j}$$

for $k = 0, ..., n-1$ and $i, j = 0, ..., h-1$ with $[\nabla^2 F]_{k,i,j} = [\nabla^2 F]_{k,j,i}$ for $i, j = 0, ..., h-1$, in directions $\mathbf{u}^{(1)}, \mathbf{u}^{(2)} \in I\!R^h$ is a first-order tangent projection in direction $\mathbf{u}^{(2)}$ of the first-order tangent projection of $\nabla^2 F$ in direction $\mathbf{u}^{(1)}$, which is $<< \nabla^2 F, \mathbf{u}^{(1)} >, \mathbf{u}^{(2)} >$.

The function $F^{(1,2)} : I\!R^h \times I\!R^h \times I\!R^h \rightarrow I\!R^n$, which is defined as

$$\mathbf{r}^{(1,2)} = F^{(1,2)}(\mathbf{u}, \mathbf{u}^{(1)}, \mathbf{u}^{(2)}) \equiv < \nabla^2 F, \mathbf{u}^{(1)}, \mathbf{u}^{(2)} > \quad (12)$$

is referred to as the *second-order tangent model* of F. The Hessian tensor $(\nabla^2 F)$ is projected along its two domain dimensions (of size h) in directions $\mathbf{u}^{(1)}$ and $\mathbf{u}^{(2)}$.

Let $\nabla^2 F$ be a symmetric 3-tensor as defined above and

$$B =< \nabla^2 F, \mathbf{u}^{(1)} >\in I\!R^{n\times h} \quad ,$$
$$\mathbf{r}^{(1,2)} =< B, \mathbf{u}^{(2)} >=< \nabla^2 F, \mathbf{u}^{(1)}, \mathbf{u}^{(2)} >\in I\!R^{n} \quad .$$

Then,

$$b_{k,j} = \sum_{i=0}^{h-1} [\nabla^2 F]_{k,j,i} \cdot [\mathbf{u}^{(1)}]_i$$

for $j = 0, ..., h-1$ and $k = 0, ..., n-1$. Hence, Equation (12) yields

$$[\mathbf{r}^{(1,2)}]_k = \sum_{j=0}^{h-1} b_{k,j} \cdot [\mathbf{u}^{(2)}]_j = \sum_{j=0}^{h-1}\sum_{i=0}^{h-1} [\nabla^2 F]_{k,j,i} \cdot [\mathbf{u}^{(1)}]_i \cdot [\mathbf{u}^{(2)}]_j \quad ,$$

for $k = 0, ..., n-1$.

Application of tangent mode to the tangent model

$$\mathbf{r}^{(1)} = F^{(1)}(\mathbf{u}, \mathbf{u}^{(1)}) \equiv< \nabla^2 F, \mathbf{u}^{(1)} >$$

yields

$$\mathbf{r}^{(1,2)} =< \nabla F, \mathbf{u}^{(1,2)} > + < \nabla^2 F, \mathbf{u}^{(1)}, \mathbf{u}^{(2)} > \quad ,$$

where $\mathbf{u}^{(2)} \equiv \frac{\partial \mathbf{u}}{\partial s}$ and $\mathbf{u}^{(1,2)} \equiv \frac{\partial \mathbf{u}^{(1)}}{\partial s}$. Thus, for $\mathbf{u}^{(1,2)} = 0$, Equation (12) follows.

Second-order tangent (also: forward-over-forward) mode software tools for AD transform a given implementation

```
F(u, r)
```

of Equation $\mathbf{r} = F(\mathbf{u})$ with u $\hat{=}\mathbf{u}$ and r $\hat{=}\mathbf{r}$ into the (algorithmic) second-order tangent subroutine

```
t2_t1_F (u, t2_u, t1_u, t2_t_1_u, r, t2_r, t1_r, t2_t1_r)
```

where additionally t1_u $\hat{=}\mathbf{u}^{(1)}$, t2_u $\hat{=}\mathbf{u}^{(2)}$, t2_t1_u $\hat{=}\mathbf{u}^{(1,2)}$, t1_r $\hat{=}\mathbf{r}^{(1)}$, t2_r $\hat{=}\mathbf{r}^{(2)}$, and t2_t1_r $\hat{=}\mathbf{r}^{(1,2)}$. The prefix tm_ marks tangent versions of program variables (and of F itself) generated by the mth application of forward mode AD. The Hessian at point $\mathbf{u}$ can be accumulated by setting $\mathbf{u}^{(1,2)} = 0$ initially and by letting $\mathbf{u}^{(1)}$ and $\mathbf{u}^{(2)}$ range independently over Cartesian basis vectors in $I\!R^h$. The individual columns of the Hessian are returned in t2_t1_r while t1_r and t2_r contain the individual columns of the Jacobian and r contains the value of the residual. This model yields a computational complexity of $O(h^2) \cdot Cost(F)$ for the accumulation of the whole Hessian.

3.4 Second-Order Adjoint Model

With tangent and adjoint as the two basic modes of AD there are three combinations remaining, each of them involving at least one application of adjoint mode. In [Nau12] we show the mathematical equivalence of the various incarnations of second-order adjoint mode (that is, forward-over-reverse, reverse-over-forward, and reverse-over-reverse) due to symmetry within the Hessian of twice symbolically differentiable multivariate vector functions. All three variants compute projections of the Hessian tensor in the image dimension (of size n) and the domain dimension (of size h) with potentially varying computational costs due to implementation issues; see [Nau12]. In the following forward-over-reverse mode AD is explained.

The Hessian $\nabla^2 F = \nabla^2 F(\mathbf{u})$ of a multivariate vector function $\mathbf{r} = F(\mathbf{u})$, $F : I\!R^h \to I\!R^n$, induces a bilinear mapping $I\!R^n \times I\!R^h \to I\!R^h$ defined by

$$(\mathbf{r}_{(1)}, \mathbf{u}^{(2)}) \mapsto < \mathbf{r}_{(1)}, \nabla^2 F, \mathbf{u}^{(2)} > = << \mathbf{r}_{(1)}, \nabla^2 F >, \mathbf{u}^{(2)} > \quad .$$

A second-order adjoint projection $< \mathbf{r}_{(1)}, \nabla^2 F, \mathbf{u}^{(2)} >$ of a symmetric 3-tensor $\nabla^2 F$, where

$$\nabla^2 F = [\nabla^2 F]_{k,i,j} = \tfrac{\partial [r]_k}{\partial [\mathbf{u}]_i \partial [\mathbf{u}]_j}$$

for $k = 0, ..., n-1$ and $i, j = 0, ..., h-1$ with $[\nabla^2 F]_{k,i,j} = [\nabla^2 F]_{k,j,i}$ for $i, j = 0, ..., h-1$, in directions $\mathbf{r}_{(1)} \in I\!R^n$ and $\mathbf{u}^{(2)} \in I\!R^h$ is a first-order tangent projection in direction $\mathbf{u}^{(2)}$ of the first-order adjoint projection of $\nabla^2 F$ in direction $\mathbf{r}_{(1)}$, which is $<< \mathbf{r}_{(1)}, \nabla^2 F >, \mathbf{u}^{(2)} >$.

The function $F_{(1)}^{(2)} : I\!R^h \times I\!R^n \times I\!R^h \to I\!R^h$, which is

$$\mathbf{u}_{(1)}^{(2)} = F_{(1)}^{(2)}(\mathbf{u}, \mathbf{r}_{(1)}, \mathbf{u}^{(2)}) \equiv < \mathbf{r}_{(1)}, \nabla^2 F(\mathbf{x}), \mathbf{u}^{(2)} > \qquad (13)$$

is referred to as the *second-order adjoint model* of F. The Hessian tensor $(\nabla^2 F)$ is projected in directions $\mathbf{r}_{(1)} \in I\!R^n$ and $\mathbf{u}^{(2)} \in I\!R^h$.

Let $\nabla^2 F$ be a symmetric 3-tensor as defined above and

$$\begin{aligned} B &= < \mathbf{r}_{(1)}, \nabla^2 F > \in I\!R^{h \times h} \quad , \\ \mathbf{u}_{(1)}^{(2)} &= < B, \mathbf{u}^{(2)} > = < \mathbf{r}_{(1)}, \nabla^2 F, \mathbf{u}^{(2)} > \in I\!R^h \quad . \end{aligned}$$

Then,

$$b_{i,j} = \textstyle\sum_{k=0}^{n-1} [\mathbf{r}_{(1)}]_k \cdot [\nabla^2 F]_{k,i,j}$$

for $i, j = 0, ..., h-1$. Hence, Equation (13) yields

$$[\mathbf{u}_{(1)}^{(2)}]_i = \sum_{j=0}^{h-1} b_{i,j} \cdot [\mathbf{u}^{(2)}]_j = \sum_{j=0}^{h-1} \sum_{k=0}^{n-1} [\mathbf{r}_{(1)}]_k \cdot [\nabla^2 F]_{k,i,j} \cdot [\mathbf{u}^{(2)}]_j \quad ,$$

for $i = 0, ..., h-1$.

Application of tangent mode to the adjoint model

$$\mathbf{u}_{(1)} = F_{(1)}(\mathbf{u}, \mathbf{r}_{(1)}) =< \mathbf{r}_{(1)}, \nabla F(\mathbf{u}) >$$

yields

$$\mathbf{u}_{(1)}^{(2)} =< \mathbf{r}_{(1)}^{(2)}, \nabla F(\mathbf{x}) > + < \mathbf{r}_{(1)}, \nabla^2 F(\mathbf{x}), \mathbf{u}^{(2)} > \quad ,$$

where $\mathbf{u}^{(2)} \equiv \frac{\partial \mathbf{u}}{\partial s}$ and $\mathbf{r}_{(1)}^{(2)} \equiv \frac{\partial \mathbf{r}_{(1)}}{\partial s}$. Thus, for $\mathbf{r}_{(1)}^{(2)} = 0$ Equation (13).

Second-order adjoint (also: forward-over-reverse) mode software tools for AD transform a given implementation

```
F(u, r)
```

of Equation $\mathbf{r} = F(\mathbf{u})$ with u $\hat{=}\mathbf{u}$ and r $\hat{=}\mathbf{r}$ into the (algorithmic) second-order adjoint subroutine

```
t2_a1_F (u, t2_u, a1_u, t2_a1_u, r, t2_r, a1_r, t2_a1_r)
```

Subscripts of second-order adjoint subroutine and variable names are replaced with the prefixes a1_ and t2_; for example, a1_u $\hat{=}\mathbf{u}_{(1)}$, t2_u $\hat{=}\mathbf{u}^{(2)}$, t2_a1_u $\hat{=}\mathbf{u}_{(1)}^{(2)}$, a1_r $\hat{=}\mathbf{r}_{(1)}$, t2_r $\hat{=}\mathbf{r}^{(2)}$, and t2_a1_r $\hat{=}\mathbf{r}_{(1)}^{(2)}$. The computation of a projection of the Hessian in directions $\mathbf{u}^{(2)}$ and $\mathbf{r}_{(1)}$ requires $\mathbf{r}_{(1)}^{(2)} = 0$ initially. The entire Hessian can be accumulated by letting $\mathbf{u}^{(2)}$ and $\mathbf{r}_{(1)}$ range over Cartesian basis vectors in $I\!R^h$ and $I\!R^n$ respectively. The individual rows of the Hessian are returned in t2_a1_u while a1_u contains the individual rows of the Jacobian and r contains the value of the residual. This model yields the computational complexity of $O(h \cdot n) \cdot Cost(F)$ for the accumulation of the whole Hessian.

3.5 Third-Order Tangent Model

The third derivative tensor $\nabla^3 F = \nabla^3 F(\mathbf{u}) \in I\!R^n \times I\!R^h \times I\!R^h \times I\!R^h$ of a multivariate vector function $\mathbf{r} = F(\mathbf{u}), F : I\!R^h \mapsto I\!R^n$ induces a trilinear mapping $I\!R^h \times I\!R^h \times I\!R^h \rightarrow I\!R^n$ defined by

$$(\mathbf{u}^{(1)}, \mathbf{u}^{(2)}, \mathbf{u}^{(3)}) \rightarrow < \nabla^3 F, \mathbf{u}^{(1)}, \mathbf{u}^{(2)}, \mathbf{u}^{(3)} > = <<< \nabla^3 F, \mathbf{u}^{(1)} >, \mathbf{u}^{(2)} >, \mathbf{u}^{(3)} > \quad .$$

A third-order tangent projection $< \nabla^3 F, \mathbf{u}^{(1)}, \mathbf{u}^{(2)}, \mathbf{u}^{(3)} >$ of a symmetric 4-tensor $\nabla^3 F$, where

$$\nabla^3 F = [\nabla^3 F]_{k,i,j,l} = \frac{\partial [\mathbf{r}]_k}{[\partial \mathbf{u}]_i [\partial \mathbf{u}]_j [\partial \mathbf{u}]_l}$$

for $k = 0, ..., n-1$ and $i, j, l = 0, ..., h-1$ with $[\nabla^3 F]_{k,i,j,l} = [\nabla^3 F]_{k,\pi(i,j,l)}$ for any permutation π of i, j, l, in directions $\mathbf{u}^{(1)}, \mathbf{u}^{(2)}, \mathbf{u}^{(3)} \in I\!R^h$ is a first-order tangent projection in direction $\mathbf{u}^{(3)}$ of a second-order tangent projection of $\nabla^3 F$ in directions $\mathbf{u}^{(1)}$ and $\mathbf{u}^{(2)}$, which is $<< \nabla^3 F, \mathbf{u}^{(1)}, \mathbf{u}^{(2)} >, \mathbf{u}^{(3)} >$. It is a first-order tangent projection in direction $\mathbf{u}^{(3)}$ of a first-order tangent projection in direction $\mathbf{u}^{(2)}$ of a first-order tangent projection of $\nabla^3 F$ in direction $\mathbf{u}^{(1)}$, i.e. $<<< \nabla^3 F, \mathbf{u}^{(1)} >, \mathbf{u}^{(2)} >, \mathbf{u}^{(3)} >$.

The function $F^{(1,2,3)} : I\!R^h \times I\!R^h \times I\!R^h \times I\!R^h \rightarrow I\!R^n$, defined as

$$\mathbf{r}^{(1,2,3)} = F^{(1,2,3)}(\mathbf{u}, \mathbf{u}^{(1)}, \mathbf{u}^{(2)}, \mathbf{u}^{(3)}) \equiv < \nabla^3 F(\mathbf{u}), \mathbf{u}^{(1)}, \mathbf{u}^{(2)}, \mathbf{u}^{(3)} > \qquad (14)$$

is referred to as the *third-order tangent model* of F. The 4-tensor $\nabla^3 F$ is projected along its three domain dimensions (of size h) in directions $\mathbf{u}^{(1)}, \mathbf{u}^{(2)}$ and $\mathbf{u}^{(3)}$.

Let $\nabla^3 F$ be a symmetric 4-tensor as defined above and

$$B =< \nabla^3 F, \mathbf{u}^{(1)} >\in I\!R^n \times I\!R^h \times I\!R^h \quad , \text{ that is}$$
$$b_{k,i,j} = \sum_{l=0}^{h-1} [\nabla^3 F]_{k,i,j,l} \cdot [\mathbf{u}^{(1)}]_l \quad ,$$

for $k = 0, ..., n-1$ and $i, j = 0, ..., h-1$. Moreover,

$$C =< B, \mathbf{u}^{(2)} >=<< \nabla^3 F, \mathbf{u}^{(1)} >, \mathbf{u}^{(2)} >\in I\!R^n \times I\!R^h \quad , \text{ that is}$$
$$c_{k,i} = \sum_{j=0}^{h-1} b_{k,i,j} \cdot [\mathbf{u}^{(2)}]_j = \sum_{j=0}^{h-1}\sum_{l=0}^{h-1} [\nabla^3 F]_{k,i,j,l} \cdot [\mathbf{u}^{(1)}]_l \cdot [\mathbf{u}^{(2)}]_j \quad ,$$

for $k = 0, ..., n-1$ and $i = 0, ..., h-1$. Then we have

$$\mathbf{r}^{(1,2,3)} =< C, \mathbf{u}^{(3)} >=<<< \nabla^3 F, \mathbf{u}^{(1)} >, \mathbf{u}^{(2)} >, \mathbf{u}^{(3)} >\in I\!R^n \quad , \text{ that is}$$
$$[\mathbf{r}^{(1,2,3)}]_k = \sum_{i=0}^{h-1} c_{k,i} \cdot [\mathbf{u}^{(3)}]_i = \sum_{i=0}^{h-1}\sum_{j=0}^{h-1}\sum_{l=0}^{h-1} [\nabla^3 F]_{k,i,j,l} \cdot [\mathbf{u}^{(1)}]_l \cdot [\mathbf{u}^{(2)}]_j \cdot [\mathbf{u}^{(3)}]_i \quad ,$$

for $k = 0, ..., n-1$.

Application of tangent mode to the second-order tangent model

$$\mathbf{r}^{(1,2)} =< \nabla^2 F, \mathbf{u}^{(1)}, \mathbf{u}^{(2)} >$$

yields

$$\mathbf{r}^{(1,2,3)} =< \nabla^2 F(\mathbf{u}), \mathbf{u}^{(1,3)}, \mathbf{u}^{(2)} > + < \nabla^2 F(\mathbf{u}), \mathbf{u}^{(1)}, \mathbf{u}^{(2,3)} > + < \nabla^3 F(\mathbf{u}), \mathbf{u}^{(1)}, \mathbf{u}^{(2)}, \mathbf{u}^{(3)} >,$$

where $\frac{\partial \mathbf{u}}{\partial s} = \mathbf{u}^{(3)}$, $\frac{\partial \mathbf{u}^{(1)}}{\partial s} = \mathbf{u}^{(1,3)}$ and $\frac{\partial \mathbf{u}^{(2)}}{\partial s} = \mathbf{u}^{(2,3)}$. Thus, for $\mathbf{u}^{(1,3)} = \mathbf{u}^{(2,3)} = 0$, Equation (14) follows.

Third-order tangent (also: forward-over-forward-over-forward) mode software tools for AD transform a given implementation

```
F(u, r)
```

of Equation $\mathbf{r} = F(\mathbf{u})$ with u $\hat{=}\mathbf{u}$ and r $\hat{=}\mathbf{r}$ into the (algorithmic) second-order tangent subroutine

```
t3_t2_t1_F(u, t3_u, t2_u, t3_t2_u, t1_u, t3_t1_u, t2_t1_u, t3_t2_t1_u,
           r, t3_r, t2_r, t3_t2_r, t1_r, t3_t1_r, t2_t1_r, t3_t2_t1_r)
```

where t1_u $\hat{=}\mathbf{u}^{(1)}$, t2_u $\hat{=}\mathbf{u}^{(2)}$,t3_u $\hat{=}\mathbf{u}^{(3)}$, t2_t1_u $\hat{=}\mathbf{u}^{(1,2)}$, t3_t1_u $\hat{=}\mathbf{u}^{(1,3)}$, t3_t2_u $\hat{=}\mathbf{u}^{(2,3)}$, t3_t2_t1_u $\hat{=}\mathbf{u}^{(1,2,3)}$, t1_r $\hat{=}\mathbf{r}^{(1)}$,t2_r $\hat{=}\mathbf{r}^{(2)}$,t3_r $\hat{=}\mathbf{r}^{(3)}$, t2_t1_r $\hat{=}\mathbf{r}^{(1,2)}$, t3_t1_r $\hat{=}\mathbf{r}^{(1,3)}$, t3_t2_r $\hat{=}\mathbf{r}^{(2,3)}$ and t3_t2_t1_r $\hat{=}\mathbf{r}^{(1,2,3)}$. First-order projection of $\nabla^3 F$ in directions $\mathbf{u}^{(1)}, \mathbf{u}^{(2)}$ and $\mathbf{u}^{(3)}$ are returned in t1_r, t2_r and t3_r by letting $\mathbf{u}^{(1)}, \mathbf{u}^{(2)}$ and $\mathbf{u}^{(3)}$ range independently over Cartesian basis vector in $I\!R^h$. Corresponding second-order projections are returned in t2_t1_r, t3_t2_r and t3_t1_r. The third derivatives are returned in t3_t2_t1_r. This model yields the computational complexity of $O(h^3) \cdot Cost(F)$ for evaluating the whole third derivative tensor.

3.6 Third-Order Adjoint Model

Due to issues in implementation, the preferred approach to the computation of higher derivatives of multivariate scalar functions is the repeated application of forward mode AD to the first-order adjoint code.

The third derivative tensor $\nabla^3 F = \nabla^3 F(\mathbf{u}) \in I\!R^n \times I\!R^h \times I\!R^h \times I\!R^h$ of a multivariate vector function $\mathbf{r} = F(\mathbf{u}), F : I\!R^h \mapsto I\!R^n$ induces a trilinear mapping $I\!R^n \times I\!R^h \times I\!R^h \to I\!R^h$ defined by

$$(\mathbf{r}_{(1)}, \mathbf{u}^{(2)}, \mathbf{u}^{(3)}) \to < \mathbf{r}_{(1)}, \nabla^3 F, \mathbf{u}^{(2)}, \mathbf{u}^{(3)} > = <<< \mathbf{r}_{(1)}, \nabla^3 F >, \mathbf{u}^{(2)} >, \mathbf{u}^{(3)} > \quad .$$

A third-order adjoint projection $< \mathbf{r}_{(1)}, \nabla^3 F, \mathbf{u}^{(2)}, \mathbf{u}^{(3)} >$ of a symmetric 4-tensor $\nabla^3 F$, where

$$\nabla^3 F = [\nabla^3 F]_{k,i,j,l} = \frac{\partial [\mathbf{r}]_k}{[\partial \mathbf{u}]_i [\partial \mathbf{u}]_j [\partial \mathbf{u}]_l}$$

for $k = 0, ..., n-1$ and $i, j, l = 0, ..., h-1$ with $[\nabla^3 F]_{k,i,j,l} = [\nabla^3 F]_{k,\pi(i,j,l)}$ for any permutation π of i, j, l, in directions $\mathbf{r}_{(1)} \in I\!R^n$ and $\mathbf{u}^{(2)}, \mathbf{u}^{(3)} \in I\!R^h$ is a first-order tangent projection in direction $\mathbf{u}^{(3)}$ of the second-order adjoint projection of $\nabla^3 F$ in directions $\mathbf{r}_{(1)}$ and $\mathbf{u}^{(2)}$, that is $<< \mathbf{r}_{(1)}, \nabla^3 F, \mathbf{u}^{(2)} >, \mathbf{u}^{(3)} >$. It is a first-order tangent projection in direction $\mathbf{u}^{(3)}$ of the first-order tangent projection in direction $\mathbf{u}^{(2)}$ of the first-order adjoint projection of $\nabla^3 F$ in direction $\mathbf{r}_{(1)}$, i.e. $<< \mathbf{r}_{(1)}, \nabla^3 F >, \mathbf{u}^{(2)} >, \mathbf{u}^{(3)} >$.

The function $F_{(1)}^{(2,3)} : I\!R^h \times I\!R^n \times I\!R^h \times I\!R^h \to I\!R^h$, defined as

$$\mathbf{u}_{(1)}^{(2,3)} = F_{(1)}^{(2,3)}(\mathbf{u}, \mathbf{r}_{(1)}, \mathbf{u}^{(2)}, \mathbf{u}^{(3)}) \equiv < \mathbf{r}_{(1)}, \nabla^3 F(\mathbf{u}), \mathbf{u}^{(2)}, \mathbf{u}^{(3)} > \qquad (15)$$

is referred to as the *third-order adjoint model* of F. The 4-tensor $\nabla^3 F$ is projected in directions $\mathbf{r}_{(1)} \in I\!R^n$ and $\mathbf{u}^{(2)}, \mathbf{u}^{(3)} \in I\!R^h$.

Let $\nabla^3 F$ be a symmetric 4-tensor as defined above and

$$B = < \mathbf{r}_{(1)}, \nabla^3 F > \in I\!R^h \times I\!R^h \times I\!R^h \quad , \text{ that is}$$

$$b_{i,j,l} = \sum_{k=0}^{n-1} [\mathbf{r}_{(1)}]_k \cdot [\nabla^3 F]_{k,i,j,l} \quad ,$$

for $i, j, l = 0, ..., h-1$. Moreover,

$$C =< B, \mathbf{u}^{(2)} >=<< \mathbf{r}_{(1)}, \nabla^3 F >, \mathbf{u}^{(2)} >\in I\!R^h \times I\!R^h \quad \text{, that is}$$

$$c_{i,j} = \sum_{l=0}^{h-1} b_{i,j,l} \cdot [\mathbf{u}^{(2)}]_l = \sum_{l=0}^{h-1} \sum_{k=0}^{n-1} [\mathbf{r}_{(1)}]_k \cdot [\nabla^3 F]_{k,i,j,l} \cdot [\mathbf{u}^{(2)}]_l \quad ,$$

for $i, j = 0, ..., h-1$. Then we have

$$\mathbf{u}_{(1)}^{(2,3)} =< C, \mathbf{u}^{(3)} >=<<< \mathbf{r}_{(1)}, \nabla^3 F >, \mathbf{u}^{(2)} >, \mathbf{u}^{(3)} >\in I\!R^h \quad \text{, that is}$$

$$[\mathbf{u}_{(1)}^{(2,3)}]_i = \sum_{j=0}^{h-1} c_{i,j} \cdot [\mathbf{u}^{(3)}]_j = \sum_{j=0}^{h-1} \sum_{l=0}^{h-1} \sum_{k=0}^{n-1} [\mathbf{r}_{(1)}]_k \cdot [\nabla^3 F]_{k,i,j,l} \cdot [\mathbf{u}^{(2)}]_l \cdot [\mathbf{u}^{(3)}]_j \quad ,$$

for $i = 0, ..., h-1$.

Application of tangent mode to the second-order adjoint model

$$\mathbf{u}_{(1)}^{(2)} =< \mathbf{r}_{(1)}, \nabla^2 F(\mathbf{u}), \mathbf{u}^{(2)} >$$

yields

$$\mathbf{u}_{(1)}^{(2,3)} =< \mathbf{r}_{(1)}^{(3)}, \nabla^2 F(\mathbf{u}), \mathbf{u}^{(2)} >< \mathbf{r}_{(1)}, \nabla^2 F(\mathbf{u}), \mathbf{u}^{(2,3)} > + < \mathbf{r}_{(1)}, \nabla^3 F(\mathbf{u}), \mathbf{u}^{(2)}, \mathbf{u}^{(3)} >,$$

where $\frac{\partial \mathbf{u}}{\partial s} = \mathbf{u}^{(3)}$, $\frac{\partial \mathbf{r}_{(1)}}{\partial s} = \mathbf{r}_{(1)}^{(3)}$ and $\frac{\partial \mathbf{u}^{(2)}}{\partial s} = \mathbf{u}^{(2,3)}$. Thus, for $\mathbf{r}_{(1)}^{(3)} = \mathbf{u}^{(2,3)} = 0$, Equation (15) follows.

Third-order adjoint (also: forward-over-forward-over-reverse) mode software tools for AD transform a given implementation

```
F(u, r)
```

of Equation $\mathbf{r} = F(\mathbf{u})$ with u $\hat{=}\mathbf{u}$ and r $\hat{=}\mathbf{r}$ into the (algorithmic) second-order tangent subroutine

```
t3_t2_a1_F(u, t3_u, t2_u, t3_t2_u, a1_u, t3_a1_u, t2_a1_u, t3_t2_a1_u,
           r, t3_r, t2_r, t3_t2_r, a1_r, t3_a1_r, t2_a1_r, t3_t2_a1_r)
```

where a1_u $\hat{=}\mathbf{u}_{(1)}$, t2_u $\hat{=}\mathbf{u}^{(2)}$, t3_u $\hat{=}\mathbf{u}^{(3)}$, t2_a1_u $\hat{=}\mathbf{u}_{(1)}^{(2)}$, t3_a1_u $\hat{=}\mathbf{u}_{(1)}^{(3)}$,t3_t2_u $\hat{=}\mathbf{u}^{(2,3)}$, t3_t2_t1_u $\hat{=}\mathbf{u}_{(1)}^{(2,3)}$, a1_r $\hat{=}\mathbf{r}_{(1)}$, t2_r $\hat{=}\mathbf{r}^{(2)}$, t3_r $\hat{=}\mathbf{r}^{(3)}$, t2_a1_r $\hat{=}\mathbf{r}_{(1)}^{(2)}$, t3_a1_r $\hat{=}\mathbf{r}_{(1)}^{(3)}$, t3_t2_r $\hat{=}\mathbf{r}^{(2,3)}$ and t3_t2_a1_r $\hat{=}\mathbf{r}_{(1)}^{(2,3)}$. The computation of the projection of tensor $\nabla^3 F$ in directions $\mathbf{u}^{(2)}, \mathbf{u}^{(3)}$ and $\mathbf{r}_{(1)}$ requires $\mathbf{u}^{(2,3)} = 0$ and $\mathbf{r}_{(1)}^{(3)} = 0$ initially. The entire tensor can be accumulated by letting $\mathbf{u}^{(2)}, \mathbf{u}^{(3)}$ and $\mathbf{r}_{(1)}$ range over Cartesian basis vectors in $I\!R^h, I\!R^h$ and $I\!R^n$ respectively. The third partial derivatives are returned in t3_t2_a1_u. This model yields the computational complexity of $O(n \cdot h^2) \cdot Cost(F)$ for evaluating the whole third derivative tensor.

The application of forward or reverse mode AD to any of the third derivative models yields fourth derivative information and so forth.

The derivative code that is generated by AD can compute projections of derivative tensors of arbitrary order, for example, (transposed) Jacobian-vector products in the first-order case, Hessian-vector products in the scalar second-order case, and so forth. Sums of the projections of tensors of various orders are returned by higher derivative code. AD users need to understand the effects of choosing certain directions for these projections (the *seeding* of the derivative code) in order to be able to retrieve (*harvest*) the desired results. For more information refer to [GW08,Nau12].

4 Second-Order Tangent Nonlinear Solver

We distinguish between two alternative approaches to the generation of second-order tangent solvers for systems of nonlinear equations. A *algorithmic* second-order tangent version of the solver computes second-order directional derivatives of the approximation of the solution, which is actually computed by the algorithm. Second-order AD is applied to the individual statements of the given implementation yielding an increase of roughly four in memory requirement as well as operations count.

A second-order *symbolic* tangent version of the solver computes the second directional derivatives of the solution under the assumption that the exact solution $\mathbf{x}^*$ has been reached. The nonlinear system $F(\mathbf{x}, \boldsymbol{\lambda}) = 0$ can be differentiated symbolically in this case. In symbolic tangent mode, the computation of second-order directional derivatives amounts to the solution of a linear system based on the Jacobian of F with respect to $\mathbf{x}^*$, which results in a significant reduction of the computational overhead in comparison with the algorithmic tangent version. The discrepancies in the results computed by second-order algorithmic and symbolic tangent nonlinear solvers depend on the accuracy of the approximation of the *primal* solution.

4.1 Algorithmic Mode

As an example for a nonlinear solver we solve the nonlinear system in Equation (1) with Newton's algorithm. The latter uses the Jacobian of the nonlinear system $(\nabla F(\mathbf{x}^i))$ at the current iterate $\mathbf{x}^i$ to determine the next Newton step.

A first-order tangent version of Newton's algorithm requires second directional derivatives of the residual. Consequently, second-order tangent version of the Newton's algorithm requires third directional derivatives of the given implementation of F.

As shown in [NLLT12], the first-order algorithmic tangent version of the given objective with Newton's algorithm used for the solution of the embedded parametrized system of nonlinear equations results from the straight application of tangent mode AD to Equations (6)–(8) as follows

For $i = 0, \ldots, \nu$:

$$A = \frac{\partial F}{\partial \mathbf{x}}(\mathbf{x}^i, \boldsymbol{\lambda}) \tag{16}$$

$$
\begin{aligned}
A^{(1)} &= < \frac{\partial^2 F}{\partial \mathbf{x} \partial(\mathbf{x},\boldsymbol{\lambda})}(\mathbf{x}^i,\boldsymbol{\lambda}), \begin{pmatrix} \mathbf{x}^{i\ (1)} \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix} > \\
\mathbf{b} &= -F(\mathbf{x}^i,\boldsymbol{\lambda}) \\
\mathbf{b}^{(1)} &= - < \frac{\partial F}{\partial(\mathbf{x},\boldsymbol{\lambda})}(\mathbf{x}^i,\boldsymbol{\lambda}), \begin{pmatrix} \mathbf{x}^{i\ (1)} \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix} > \\
\mathbf{s} &= \mathbf{L}(A,\mathbf{b}) \\
\mathbf{s}^{(1)} &= < \frac{\partial \mathbf{L}}{\partial(A,\mathbf{b})}(A,\mathbf{b}), \begin{pmatrix} A^{(1)} \\ \mathbf{b}^{(1)} \end{pmatrix} > \\
\mathbf{x}^{i+1} &= \mathbf{x}^i + \mathbf{s} \\
\mathbf{x}^{i+1\ (1)} &= \mathbf{x}^{i\ (1)} + \mathbf{s}^{(1)}
\end{aligned} \tag{17}
$$

The linear solver ($\mathbf{L}$) is augmented at the statement-level with local tangent models, thus roughly duplicating the required memory as well as the number of operations performed.

Reapplication of tangent AD to Equations (16)–(17) yields

for $i = 0, \ldots, \nu$:

$$
\begin{aligned}
A &= \frac{\partial F}{\partial \mathbf{x}}(\mathbf{x}^i,\boldsymbol{\lambda}) = < \frac{\partial F}{\partial \mathbf{x}}, I_n > = < \frac{\partial F}{\partial(\mathbf{x},\boldsymbol{\lambda})}, \overset{(\in \mathbb{R}^{(n+m)\times n})}{\begin{pmatrix} I_n \\ \mathbf{0}_m \end{pmatrix}} > \qquad (18) \\
A^{(2)} &= < \frac{\partial^2 F}{\partial \mathbf{x} \partial(\mathbf{x},\boldsymbol{\lambda})}, \begin{pmatrix} \mathbf{x}^{i\ (2)} \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > = < \frac{\partial^2 F}{\partial(\mathbf{x},\boldsymbol{\lambda})^2}, \begin{pmatrix} I_n \\ \mathbf{0}_m \end{pmatrix}, \begin{pmatrix} \mathbf{x}^{i\ (2)} \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > \\
A^{(1)} &= < \frac{\partial^2 F}{\partial \mathbf{x} \partial(\mathbf{x},\boldsymbol{\lambda})}, \begin{pmatrix} \mathbf{x}^{i\ (1)} \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix} > = < \frac{\partial^2 F}{\partial(\mathbf{x},\boldsymbol{\lambda})^2}, \begin{pmatrix} I_n \\ \mathbf{0}_m \end{pmatrix}, \begin{pmatrix} \mathbf{x}^{i\ (1)} \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix} > \\
A^{(1,2)} &= < \frac{\partial^2 F}{\partial \mathbf{x} \partial(\mathbf{x},\boldsymbol{\lambda})}, \begin{pmatrix} \mathbf{x}^{i\ (1,2)} \\ \boldsymbol{\lambda}^{(1,2)} \end{pmatrix} > + < \frac{\partial^3 F}{\partial \mathbf{x} \partial(\mathbf{x},\boldsymbol{\lambda})^2}, \begin{pmatrix} \mathbf{x}^{i\ (1)} \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix}, \begin{pmatrix} \mathbf{x}^{i\ (2)} \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > \\
&= < \frac{\partial^2 F}{\partial \mathbf{x} \partial(\mathbf{x},\boldsymbol{\lambda})}, I_n, \begin{pmatrix} \mathbf{x}^{i\ (1,2)} \\ \boldsymbol{\lambda}^{(1,2)} \end{pmatrix} > + < \frac{\partial^3 F}{\partial \mathbf{x} \partial(\mathbf{x},\boldsymbol{\lambda})^2}, I_n, \begin{pmatrix} \mathbf{x}^{i\ (1)} \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix}, \begin{pmatrix} \mathbf{x}^{i\ (2)} \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > \\
&= < \frac{\partial^2 F}{\partial(\mathbf{x},\boldsymbol{\lambda})^2}, \begin{pmatrix} I_n \\ \mathbf{0}_m \end{pmatrix}, \begin{pmatrix} \mathbf{x}^{i\ (1,2)} \\ \boldsymbol{\lambda}^{(1,2)} \end{pmatrix} > + < \frac{\partial^3 F}{\partial(\mathbf{x},\boldsymbol{\lambda})^3}, \begin{pmatrix} I_n \\ \mathbf{0}_m \end{pmatrix}, \begin{pmatrix} \mathbf{x}^{i\ (1)} \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix}, \begin{pmatrix} \mathbf{x}^{i\ (2)} \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > \\
\mathbf{b} &= -F(\mathbf{x}^i,\boldsymbol{\lambda}) \\
\mathbf{b}^{(2)} &= - < \frac{\partial F}{\partial(\mathbf{x},\boldsymbol{\lambda})}, \begin{pmatrix} \mathbf{x}^{i\ (2)} \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > \\
\mathbf{b}^{(1)} &= - < \frac{\partial F}{\partial(\mathbf{x},\boldsymbol{\lambda})}, \begin{pmatrix} \mathbf{x}^{i\ (1)} \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix} > \\
\mathbf{b}^{(1,2)} &= - < \frac{\partial F}{\partial(\mathbf{x},\boldsymbol{\lambda})}, \begin{pmatrix} \mathbf{x}^{i\ (1,2)} \\ \boldsymbol{\lambda}^{(1,2)} \end{pmatrix} > - < \frac{\partial^2 F}{\partial(\mathbf{x},\boldsymbol{\lambda})^2}, \begin{pmatrix} \mathbf{x}^{i\ (1)} \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix}, \begin{pmatrix} \mathbf{x}^{i\ (2)} \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > \\
\mathbf{s} &= \mathbf{L}(A,\mathbf{b}) \\
\mathbf{s}^{(2)} &= < \frac{\partial \mathbf{L}}{\partial(A,\mathbf{b})}(A,\mathbf{b}), \begin{pmatrix} A^{(2)} \\ \mathbf{b}^{(2)} \end{pmatrix} >
\end{aligned}
$$

$$\mathbf{s}^{(1)} =< \frac{\partial \mathbf{L}}{\partial (A,\mathbf{b})}(A,\mathbf{b}), \begin{pmatrix} A^{(1)} \\ \mathbf{b}^{(1)} \end{pmatrix} >$$
$$\mathbf{s}^{(1,2)} =< \frac{\partial \mathbf{L}}{\partial (A,\mathbf{b})}(A,\mathbf{b}), \begin{pmatrix} A^{(1,2)} \\ \mathbf{b}^{(1,2)} \end{pmatrix} > + < \frac{\partial^2 \mathbf{L}}{\partial (A,\mathbf{b})^2}(A,\mathbf{b}), \begin{pmatrix} A^{(1)} \\ \mathbf{b}^{(1)} \end{pmatrix}, \begin{pmatrix} A^{(2)} \\ \mathbf{b}^{(2)} \end{pmatrix} >$$

$$\begin{aligned} \mathbf{x}^{i+1} &= \mathbf{x}^i + \mathbf{s} \\ \mathbf{x}^{i+1\ (2)} &= \mathbf{x}^{i\ (2)} + \mathbf{s}^{(2)} \\ \mathbf{x}^{i+1\ (1)} &= \mathbf{x}^{i\ (1)} + \mathbf{s}^{(1)} \\ \mathbf{x}^{i+1\ (1,2)} &= \mathbf{x}^{i\ (1,2)} + \mathbf{s}^{(1,2)} \quad , \end{aligned}$$

where all derivatives of F, e.g. $\frac{\partial F}{\partial (\mathbf{x},\boldsymbol{\lambda})}$ are evaluated at point $(\mathbf{x}^i, \boldsymbol{\lambda})$.

In Equation (18), $I_n \in I\!R^{n\times n}$ is the identity matrix, filled with m zero rows yielding $\begin{pmatrix} I_n \\ \mathbf{0}_m \end{pmatrix} \in I\!R^{(n+m)\times n}$. Furthermore, the differentiation of $\mathbf{L}(A,\mathbf{b}) \in I\!R^n$ with respect to $(A,\mathbf{b})$ is done through serialization of $(A,\mathbf{b})$, meaning that $A \in I\!R^{n\times n}$ and $\mathbf{b} \in I\!R^n$, $(A,\mathbf{b})$ is considered as a vector of size n^2+n. Consequently, $\frac{\partial \mathbf{L}}{\partial (A,\mathbf{b})}(A,\mathbf{b}) \in I\!R^{n\times(n^2+n)}$ and $\frac{\partial^2 \mathbf{L}}{\partial (A,b)^2}(A,b) \in I\!R^{n\times(n^2+n)\times(n^2+n)}$. Similarly we get $\begin{pmatrix} A^k \\ \mathbf{b}^k \end{pmatrix} \in I\!R^{n^2+n}$ for $k = (1), (2)$ or $(1,2)$.

In the above equations, first, second and third derivatives of F with respect to $(\mathbf{x}^i, \boldsymbol{\lambda})$ are required when computing third-order tangents of F with respect to $(\mathbf{x}^i, \boldsymbol{\lambda})$ using AD software tools, the function value as well as derivatives up to third order are evaluated.

In this case, the required memory is four times the memory (MEM) required by the nonlinear solver itself, i.e. $MEM(\mathbf{L}) \sim O(n^2)$ and the number of operations is four times the operations (OPS) performed by the nonlinear solver itself, i.e., $OPS(\mathbf{L}) \sim \nu \cdot O(n^3)$.

4.2 Symbolic Mode

Lemma 1 (Differentiation of a Matrix-Vector Product).
Let $\mathbf{G}(\mathbf{c}) =< \mathbf{A}(\mathbf{c}), \mathbf{b}(\mathbf{c}) >$ *be a symbolic bilinear map in which* $A(\mathbf{c}) \in \mathbf{R^{m\times n}}, \mathbf{b}(\mathbf{c}) \in \mathbf{R^n}$ *and* $\mathbf{G} : \mathbf{R^{m\times n}} \times \mathbf{R^n} \to \mathbf{R^m}$ *are differentiable functions. Differentiation of* $\mathbf{G}$ *with respect to* $\mathbf{c}$ *yields*

$$\mathbf{G^{(1)}}(\mathbf{c}) =< \mathbf{A^{(1)}}(\mathbf{c}), \mathbf{b}(\mathbf{c}) > + < \mathbf{A}(\mathbf{c}), \mathbf{b^{(1)}}(\mathbf{c}) > \quad .$$

Proof. Let $\mathbf{G^{(1)}} = \frac{\partial}{\partial \mathbf{c}} \mathbf{G}(\mathbf{c})$ and $\mathbf{G^{(1)}} \in \mathbf{R^m}$. Then we have

$$\begin{aligned}
\mathbf{G}^{(1)} &= \frac{\partial}{\partial \mathbf{c}} < A(\mathbf{c}), \mathbf{b}(\mathbf{c}) > \\
\mathbf{g}_i^{(1)} &= \frac{\partial}{\partial \mathbf{c}} \left(\sum_{j=1}^{n} a_{i,j}(\mathbf{c}) \cdot \mathbf{b}_j(\mathbf{c}) \right) = \sum_{j=1}^{n} \frac{\partial}{\partial \mathbf{c}} (a_{i,j}(\mathbf{c}) \cdot \mathbf{b}_j(\mathbf{c})) \\
&= \sum_{j=1}^{n} (\frac{\partial a_{i,j}(\mathbf{c})}{\partial \mathbf{c}} \cdot \mathbf{b}_j(\mathbf{c}) + a_{i,j}(\mathbf{c}) \cdot \frac{\partial \mathbf{b}_j(\mathbf{c})}{\partial \mathbf{c}}) \\
&= \sum_{j=1}^{n} \frac{\partial a_{i,j}(\mathbf{c})}{\partial \mathbf{c}} \cdot \mathbf{b}_j(\mathbf{c}) + \sum_{j=1}^{n} a_{i,j}(\mathbf{c}) \cdot \frac{\partial \mathbf{b}_j(\mathbf{c})}{\partial \mathbf{c}}) \quad ,
\end{aligned}$$

for $i = 1, \ldots, m$. Consequently,

$$\mathbf{G}^{(1)} = \frac{\partial A(\mathbf{c})}{\partial \mathbf{c}} \cdot \mathbf{b}(\mathbf{c}) + A(\mathbf{c}) \cdot \frac{\partial \mathbf{b}(\mathbf{c})}{\partial \mathbf{c}} =< A^{(1)}(\mathbf{c}), \mathbf{b}(\mathbf{c}) > + < A(\mathbf{c}), \mathbf{b}^{(1)}(\mathbf{c}) > \quad .$$

For further information refer to [Gil08].

Lemma 2. *Let* $T \in I\!R^{n \times (n+m) \times (n+m)}, \mathbf{x}^{(1)}, \mathbf{x}^{(2)} \in I\!R^n$ *and* $\boldsymbol{\lambda}^{(1)}, \boldsymbol{\lambda}^{(2)} \in I\!R^m$. *Then we have*

$$\begin{aligned}
< T, \begin{pmatrix} \mathbf{x}^{(1)} \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix}, \begin{pmatrix} \mathbf{x}^{(2)} \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} >= < T, \begin{pmatrix} \mathbf{x}^{(1)} \\ \mathbf{0}_m \end{pmatrix}, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > + < T, \begin{pmatrix} \mathbf{x}^{(1)} \\ \mathbf{0}_m \end{pmatrix}, \begin{pmatrix} \mathbf{x}^{(2)} \\ \mathbf{0}_m \end{pmatrix} > \\
+ < T, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix}, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > + < T, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix}, \begin{pmatrix} \mathbf{x}^{(2)} \\ \mathbf{0}_m \end{pmatrix} >
\end{aligned}$$

Proof. Let $\mathbf{a} = \begin{pmatrix} \mathbf{x}^{(1)} \\ \mathbf{0}_m \end{pmatrix}, \mathbf{b} = \begin{pmatrix} \mathbf{x}^{(2)} \\ \mathbf{0}_m \end{pmatrix}, \mathbf{c} = \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix}, \mathbf{d} = \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} \in \mathbf{R^{n+m}}$. Therefore, we have

$$
\begin{aligned}
&< T, \begin{pmatrix}\mathbf{x}^{(1)}\\ \mathbf{0}_m\end{pmatrix}, \begin{pmatrix}\mathbf{0}_n\\ \boldsymbol{\lambda}^{(2)}\end{pmatrix} > + < T, \begin{pmatrix}\mathbf{x}^{(1)}\\ \mathbf{0}_m\end{pmatrix}, \begin{pmatrix}\mathbf{x}^{(2)}\\ \mathbf{0}_m\end{pmatrix} > \\
&+ < T, \begin{pmatrix}\mathbf{0}_n\\ \boldsymbol{\lambda}^{(1)}\end{pmatrix}, \begin{pmatrix}\mathbf{0}_n\\ \boldsymbol{\lambda}^{(2)}\end{pmatrix} > + < T, \begin{pmatrix}\mathbf{0}_n\\ \boldsymbol{\lambda}^{(1)}\end{pmatrix}, \begin{pmatrix}\mathbf{x}^{(2)}\\ \mathbf{0}_m\end{pmatrix} > \\
&= \sum_{j=0}^{n+m}\sum_{k=0}^{n+m} T_{ijk}\cdot a_k \cdot d_j + \sum_{j=0}^{n+m}\sum_{k=0}^{n+m} T_{ijk}\cdot a_k \cdot b_j \\
&+ \sum_{j=0}^{n+m}\sum_{k=0}^{n+m} T_{ijk}\cdot c_k \cdot d_j + \sum_{j=0}^{n+m}\sum_{k=0}^{n+m} T_{ijk}\cdot c_k \cdot b_j \\
&= \sum_{j=0}^{n+m}\sum_{k=0}^{n+m} T_{ijk}\cdot (\begin{pmatrix}\mathbf{x}^{(1)}\\ \mathbf{0}_m\end{pmatrix}\cdot \begin{pmatrix}\mathbf{0}_n\\ \boldsymbol{\lambda}^{(2)}\end{pmatrix} + \begin{pmatrix}\mathbf{x}^{(1)}\\ \mathbf{0}_m\end{pmatrix}\cdot \begin{pmatrix}\mathbf{x}^{(2)}\\ \mathbf{0}_m\end{pmatrix} \\
&+ \begin{pmatrix}\mathbf{0}_n\\ \boldsymbol{\lambda}^{(1)}\end{pmatrix}\cdot \begin{pmatrix}\mathbf{0}_n\\ \boldsymbol{\lambda}^{(2)}\end{pmatrix} + \begin{pmatrix}\mathbf{0}_n\\ \boldsymbol{\lambda}^{(1)}\end{pmatrix}\cdot \begin{pmatrix}\mathbf{x}^{(2)}\\ \mathbf{0}_m\end{pmatrix}) \\
&= \sum_{j=0}^{n+m}\sum_{k=0}^{n+m} T_{ijk}\cdot (\begin{pmatrix}\mathbf{x}^{(1)}\\ \mathbf{0}_m\end{pmatrix}\cdot \left(\begin{pmatrix}\mathbf{0}_n\\ \boldsymbol{\lambda}^{(2)}\end{pmatrix} + \begin{pmatrix}\mathbf{x}^{(2)}\\ \mathbf{0}_m\end{pmatrix}\right) + \begin{pmatrix}\mathbf{0}_n\\ \boldsymbol{\lambda}^{(1)}\end{pmatrix}\cdot \left(\begin{pmatrix}\mathbf{0}_n\\ \boldsymbol{\lambda}^{(2)}\end{pmatrix} + \begin{pmatrix}\mathbf{x}^{(2)}\\ \mathbf{0}_m\end{pmatrix}\right)) \\
&= \sum_{j=0}^{n+m}\sum_{k=0}^{n+m} T_{ijk}\cdot (\begin{pmatrix}\mathbf{x}^{(1)}\\ \mathbf{0}_m\end{pmatrix}\cdot \left(\begin{pmatrix}\mathbf{x}^{(2)}\\ \boldsymbol{\lambda}^{(2)}\end{pmatrix}\right) + \begin{pmatrix}\mathbf{0}_n\\ \boldsymbol{\lambda}^{(1)}\end{pmatrix}\cdot \left(\begin{pmatrix}\mathbf{x}^{(2)}\\ \boldsymbol{\lambda}^{(2)}\end{pmatrix}\right)) \\
&= \sum_{j=0}^{n+m}\sum_{k=0}^{n+m} T_{ijk}\cdot (\begin{pmatrix}\mathbf{x}^{(2)}\\ \boldsymbol{\lambda}^{(2)}\end{pmatrix}\cdot \left(\begin{pmatrix}\mathbf{x}^{(1)}\\ \mathbf{0}_m\end{pmatrix} + \begin{pmatrix}\mathbf{0}_n\\ \boldsymbol{\lambda}^{(1)}\end{pmatrix}\right)) \\
&= \sum_{j=0}^{n+m}\sum_{k=0}^{n+m} T_{ijk}\cdot (\begin{pmatrix}\mathbf{x}^{(2)}\\ \boldsymbol{\lambda}^{(2)}\end{pmatrix}\cdot \left(\begin{pmatrix}\mathbf{x}^{(1)}\\ \mathbf{0}_m\end{pmatrix} + \begin{pmatrix}\mathbf{0}_n\\ \boldsymbol{\lambda}^{(1)}\end{pmatrix}\right)) \\
&= \sum_{j=0}^{n+m}\sum_{k=0}^{n+m} T_{ijk}\cdot \begin{pmatrix}\mathbf{x}^{(1)}\\ \boldsymbol{\lambda}^{(1)}\end{pmatrix}\cdot \begin{pmatrix}\mathbf{x}^{(2)}\\ \boldsymbol{\lambda}^{(2)}\end{pmatrix} \quad \text{for} \quad i = 0, ..., n \\
&= < T, \begin{pmatrix}\mathbf{x}^{(1)}\\ \boldsymbol{\lambda}^{(1)}\end{pmatrix}, \begin{pmatrix}\mathbf{x}^{(2)}\\ \boldsymbol{\lambda}^{(2)}\end{pmatrix} >
\end{aligned}
$$

Theorem 1. Symbolic Second-Order Tangent Solvers of Nonlinear Equation: *Let $\mathbf{r} = F(\mathbf{x}(\boldsymbol{\lambda}), \boldsymbol{\lambda}) : I\!R^n \times I\!R^m \rightarrow I\!R^n$ for a given $\boldsymbol{\lambda} \in I\!R^m$, a vector $\mathbf{x} \in R^n$ is sought such that $F(\mathbf{x}(\boldsymbol{\lambda}), \boldsymbol{\lambda}) = 0$. Second-order tangent differentiation of $F(\mathbf{x}, \boldsymbol{\lambda}) = 0$ at the solution $\mathbf{x} = \mathbf{x}^*$ with respect to $\boldsymbol{\lambda}$, i.e., computation of $\mathbf{x}^{(1,2)} \in I\!R^n$, amounts to the solution of the linear system*

$$
\frac{\partial F}{\partial \mathbf{x}} \cdot \mathbf{x}^{(1,2)} =< \frac{\partial F}{\partial \mathbf{x}}, \mathbf{x}^{(1,2)} >= - < \nabla^2 F, \begin{pmatrix}\mathbf{x}^{(1)}\\ \boldsymbol{\lambda}^{(1)}\end{pmatrix}, \begin{pmatrix}\mathbf{x}^{(2)}\\ \boldsymbol{\lambda}^{(2)}\end{pmatrix} > \quad . \qquad (19)
$$

Proof (Version 1).

As shown in [NLLT12], first-order symbolic tangent differentiation of $F(\mathbf{x}(\boldsymbol{\lambda}), \boldsymbol{\lambda}) = 0$ at the solution $\mathbf{x} = \mathbf{x}^*$ with respect to $\boldsymbol{\lambda}$, i.e., computation of $\mathbf{x}^{(1)}$, yields

$$\frac{\partial F}{\partial \mathbf{x}}(\mathbf{x},\boldsymbol{\lambda})\cdot \mathbf{x}^{(1)} = -\frac{\partial F}{\partial \boldsymbol{\lambda}}(\mathbf{x},\boldsymbol{\lambda})\cdot \boldsymbol{\lambda}^{(1)} \tag{20}$$

which is a linear system and $\frac{\partial F}{\partial \mathbf{x}}$ is the Jacobian matrix. This equation can be written as

$$\frac{\partial F}{\partial \mathbf{x}}(\mathbf{x},\boldsymbol{\lambda})\cdot \mathbf{x}^{(1)} + \frac{\partial F}{\partial \boldsymbol{\lambda}}(\mathbf{x},\boldsymbol{\lambda})\cdot \boldsymbol{\lambda}^{(1)} =< \frac{\partial F}{\partial \mathbf{x}}(\mathbf{x},\boldsymbol{\lambda}), \mathbf{x}^{(1)} > + < \frac{\partial F}{\partial \boldsymbol{\lambda}}(\mathbf{x},\boldsymbol{\lambda}), \boldsymbol{\lambda}^{(1)} > . \tag{21}$$

An alternative for evaluating the second directional derivatives of the solution $\mathbf{x} = \mathbf{x}^*$ with respect to $\boldsymbol{\lambda}$ in $F(\mathbf{x},\boldsymbol{\lambda}) = 0$ is to differentiate Equation (21) with respect to $\boldsymbol{\lambda}$

$$\begin{aligned} &\frac{d}{d\boldsymbol{\lambda}}\left(< \frac{\partial F}{\partial \mathbf{x}}(\mathbf{x},\boldsymbol{\lambda}), \mathbf{x}^{(1)} > + < \frac{\partial F}{\partial \boldsymbol{\lambda}}(\mathbf{x},\boldsymbol{\lambda}), \boldsymbol{\lambda}^{(1)} >\right)\cdot \boldsymbol{\lambda}^{(2)} \\ &= < \frac{d < \frac{\partial F}{\partial \mathbf{x}}(\mathbf{x},\boldsymbol{\lambda}), \mathbf{x}^{(1)} >}{d\boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} > + < \frac{d < \frac{\partial F}{\partial \boldsymbol{\lambda}}(\mathbf{x},\boldsymbol{\lambda}), \boldsymbol{\lambda}^{(1)} >}{d\boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} > \quad , \end{aligned} \tag{22}$$

where $< \frac{d< \frac{\partial F}{\partial \mathbf{x}}(\mathbf{x},\boldsymbol{\lambda}),\mathbf{x}^{(1)}>}{d\boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} >$ and $< \frac{d< \frac{\partial F}{\partial \boldsymbol{\lambda}}(\mathbf{x},\boldsymbol{\lambda}),\boldsymbol{\lambda}^{(1)}>}{d\boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} >$ are the total derivatives of $g_1 =< \frac{\partial F}{\partial \mathbf{x}}(\mathbf{x},\boldsymbol{\lambda}), \mathbf{x}^{(1)} >$ and $g_2 =< \frac{\partial F}{\partial \boldsymbol{\lambda}}(\mathbf{x},\boldsymbol{\lambda}), \boldsymbol{\lambda}^{(1)} >$ with respect to $\boldsymbol{\lambda}$ respectively. Because g_1 and g_2 depend on $\mathbf{x}$ as well as $\boldsymbol{\lambda}$, we have

$$\begin{aligned} < \frac{dg_1}{d\boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} > &=< \frac{\partial g_1}{\partial \boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} > + < \frac{\partial g_1}{\partial \mathbf{x}}, \underbrace{< \frac{\partial \mathbf{x}}{\partial \boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} >}_{=\mathbf{x}^{(2)}} > \\ &=< \frac{\partial < \frac{\partial F}{\partial \mathbf{x}}(\mathbf{x},\boldsymbol{\lambda}), \mathbf{x}^{(1)} >}{\partial \boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} > + < \frac{\partial < \frac{\partial F}{\partial \mathbf{x}}(\mathbf{x},\boldsymbol{\lambda}), \mathbf{x}^{(1)} >}{\partial \mathbf{x}}, \mathbf{x}^{(2)} > \quad , \end{aligned} \tag{23}$$

$$\begin{aligned} < \frac{dg_2}{d\boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} > &=< \frac{\partial g_2}{\partial \boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} > + < \frac{\partial g_2}{\partial \mathbf{x}}, \underbrace{< \frac{\partial \mathbf{x}}{\partial \boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} >}_{=\mathbf{x}^{(2)}} > \\ &=< \frac{\partial < \frac{\partial F}{\partial \boldsymbol{\lambda}}(\mathbf{x},\boldsymbol{\lambda}), \boldsymbol{\lambda}^{(1)} >}{\partial \boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} > + < \frac{\partial < \frac{\partial F}{\partial \boldsymbol{\lambda}}(\mathbf{x},\boldsymbol{\lambda}), \boldsymbol{\lambda}^{(1)} >}{\partial \mathbf{x}}, \mathbf{x}^{(2)} > . \end{aligned} \tag{24}$$

According to Theorem 1, the first term on the right hand side of Equation (23) yields

$$\begin{aligned} < \frac{\partial < \frac{\partial F}{\partial \mathbf{x}}, \mathbf{x}^{(1)} >}{\partial \boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} > &= < \frac{\partial^2 F}{\partial \mathbf{x}\partial \boldsymbol{\lambda}}, \mathbf{x}^{(1)}, \boldsymbol{\lambda}^{(2)} > + < \frac{\partial F}{\partial \mathbf{x}}, < \frac{\partial \mathbf{x}^{(1)}}{\partial \boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} >> \\ &= < \frac{\partial^2 F}{\partial (\mathbf{x},\boldsymbol{\lambda})^2}, \begin{pmatrix} \mathbf{x}^{(1)} \\ \mathbf{0}_m \end{pmatrix}, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > \\ &\quad + < \frac{\partial F}{\partial \mathbf{x}}, < \frac{\partial \mathbf{x}^{(1)}}{\partial \boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} >> \\ &= < \nabla^2 F, \begin{pmatrix} \mathbf{x}^{(1)} \\ \mathbf{0}_m \end{pmatrix}, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > + < \frac{\partial F}{\partial \mathbf{x}}, \mathbf{x}^{(1,2)} > . \end{aligned}$$

Similarly, the second term on the right hand side of Equation (23) becomes

$$\begin{aligned} < \frac{\partial < \frac{\partial F}{\partial \mathbf{x}}, \mathbf{x}^{(1)} >}{\partial \mathbf{x}}, \mathbf{x}^{(2)} > &=< \frac{\partial^2 F}{\partial \mathbf{x}^2}, \mathbf{x}^{(1)}, \mathbf{x}^{(2)} > + < \frac{\partial F}{\partial \mathbf{x}}, < \underbrace{\frac{\partial \mathbf{x}^{(1)}}{\partial \mathbf{x}}}_{0}, \mathbf{x}^{(2)} >> \\ &=< \underbrace{\frac{\partial^2 F}{\partial (\mathbf{x}, \boldsymbol{\lambda})^2}}_{\nabla^2 F}, \begin{pmatrix} \mathbf{x}^{(1)} \\ \mathbf{0}_m \end{pmatrix}, \begin{pmatrix} \mathbf{x}^{(2)} \\ \mathbf{0}_m \end{pmatrix} > . \end{aligned}$$

Applying Theorem 1, the first term on the right hand side of Equation (24) yields

$$\begin{aligned} < \frac{\partial < \frac{\partial F}{\partial \boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(1)} >}{\partial \boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} > &=< \frac{\partial^2 F}{\partial \boldsymbol{\lambda}^2}, \boldsymbol{\lambda}^{(1)}, \boldsymbol{\lambda}^{(2)} > + < \frac{\partial F}{\partial \boldsymbol{\lambda}}, < \underbrace{\frac{\partial \boldsymbol{\lambda}^{(1)}}{\partial \boldsymbol{\lambda}}}_{0}, \boldsymbol{\lambda}^{(2)} >> \\ &=< \underbrace{\frac{\partial^2 F}{\partial (\mathbf{x}, \boldsymbol{\lambda})^2}}_{\nabla^2 F}, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix}, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > . \end{aligned}$$

Similarly, the second term on the right hand side of Equation (24) becomes

$$\begin{aligned} < \frac{\partial < \frac{\partial F}{\partial \boldsymbol{\lambda}}(\mathbf{x}, \boldsymbol{\lambda}), \boldsymbol{\lambda}^{(1)} >}{\partial \mathbf{x}}, \mathbf{x}^{(2)} > &=< \frac{\partial^2 F}{\partial \boldsymbol{\lambda} \partial \mathbf{x}}, \boldsymbol{\lambda}^{(1)}, \mathbf{x}^{(2)} > + < \frac{\partial F}{\partial \boldsymbol{\lambda}}, < \underbrace{\frac{\partial \boldsymbol{\lambda}^{(1)}}{\partial \mathbf{x}}}_{0}, \mathbf{x}^{(2)} >> \\ &=< \underbrace{\frac{\partial^2 F}{\partial (\mathbf{x}, \boldsymbol{\lambda})^2}}_{\nabla^2 F}, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix}, \begin{pmatrix} \mathbf{x}^{(2)} \\ \mathbf{0}_m \end{pmatrix} > . \end{aligned}$$

Consequently, Equation (22) yields

$$\begin{aligned} &< \nabla^2 F, \begin{pmatrix} \mathbf{x}^{(1)} \\ \mathbf{0}_m \end{pmatrix}, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > + < \frac{\partial F}{\partial \mathbf{x}}, \mathbf{x}^{(1,2)} > + < \nabla^2 F, \begin{pmatrix} \mathbf{x}^{(1)} \\ \mathbf{0}_m \end{pmatrix}, \begin{pmatrix} \mathbf{x}^{(2)} \\ \mathbf{0}_m \end{pmatrix} > \\ &+ < \nabla^2 F, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix}, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > + < \nabla^2 F, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix}, \begin{pmatrix} \mathbf{x}^{(2)} \\ \mathbf{0}_m \end{pmatrix} > = 0 \end{aligned}$$

$$\begin{aligned} < \frac{\partial F}{\partial \mathbf{x}}, \mathbf{x}^{(1,2)} > &= - < \nabla^2 F, \begin{pmatrix} \mathbf{x}^{(1)} \\ \mathbf{0}_m \end{pmatrix}, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > - < \nabla^2 F, \begin{pmatrix} \mathbf{x}^{(1)} \\ \mathbf{0}_m \end{pmatrix}, \begin{pmatrix} \mathbf{x}^{(2)} \\ \mathbf{0}_m \end{pmatrix} > \\ &- < \nabla^2 F, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix}, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > - < \nabla^2 F, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix}, \begin{pmatrix} \mathbf{x}^{(2)} \\ \mathbf{0}_m \end{pmatrix} > \end{aligned}$$

Therefore, according to Theorem 2,

$$< \frac{\partial F}{\partial \mathbf{x}}, \mathbf{x}^{(1,2)} > = \frac{\partial F}{\partial \mathbf{x}} \cdot \mathbf{x}^{(1,2)} = - < \nabla^2 F, \begin{pmatrix} \mathbf{x}^{(1)} \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix}, \begin{pmatrix} \mathbf{x}^{(2)} \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > \quad .$$

In the above equation, the right hand side can be calculated by second-order tangent AD. Computing the right hand side and the Jacobian matrix $\frac{\partial F}{\partial \mathbf{x}}$ with AD, this system can be solved by using the same linear solver as applied for computation of the first-order symbolic tangent $\mathbf{x}^{(1)}$ in Equation (21), e.g. Gauss.

Proof (Version 2).

As shown in [NLLT12], the first-order symbolic tangent differentiation of $F(\mathbf{x}(\boldsymbol{\lambda}), \boldsymbol{\lambda}) = 0$ at the solution $\mathbf{x} = \mathbf{x}^*$ with respect to $\boldsymbol{\lambda}$, i.e., the computation of $\mathbf{x}^{(1)}$, yields

$$\begin{aligned}
\frac{\partial F}{\partial \mathbf{x}}(\mathbf{x}, \boldsymbol{\lambda}) \cdot \mathbf{x}^{(1)} &= -\frac{\partial F}{\partial \boldsymbol{\lambda}}(\mathbf{x}, \boldsymbol{\lambda}) \cdot \boldsymbol{\lambda}^{(1)} && (25)\\
\frac{\partial F}{\partial \mathbf{x}} \cdot \mathbf{x}^{(1)} &= - < \frac{\partial F}{\partial \boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(1)} > \\
\frac{\partial F}{\partial \mathbf{x} \partial \boldsymbol{\lambda}} \cdot \begin{pmatrix} \mathbf{x}^{(1)} \\ \mathbf{0}_m \end{pmatrix} &= - < \frac{\partial F}{\partial \mathbf{x} \partial \boldsymbol{\lambda}}, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix} > \\
\nabla F \cdot \begin{pmatrix} \mathbf{x}^{(1)} \\ \mathbf{0}_m \end{pmatrix} &= - < \nabla F, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix} > \quad , && (26)
\end{aligned}$$

which is a linear system of type $A\mathbf{c} = \mathbf{b}$, where $A = \nabla F$, $\mathbf{c} = \begin{pmatrix} \mathbf{x}^{(1)} \\ \mathbf{0}_m \end{pmatrix}$ and $\mathbf{b} = - < \nabla F, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix} >$.

An option to evaluate the second-order directional derivatives of the solution $\mathbf{x} = \mathbf{x}^*$ with respect to $\boldsymbol{\lambda}$ in $F(\mathbf{x}, \boldsymbol{\lambda}) = 0$ is to apply the symbolic first-order tangent version for linear solvers to Equation (26) .

As shown in [Gil08], for the linear system $A\mathbf{c} = \mathbf{b}$ we have

$$\begin{aligned}
\mathbf{c} &= \mathbf{L}(A, \mathbf{b}) \quad , \\
\mathbf{c}^{(1)} &= \mathbf{L}^{(1)}(A, A^{(1)}, \mathbf{b}, \mathbf{b}^{(1)}) =< \frac{\partial \mathbf{c}}{\partial A}, A^{(1)} > + < \frac{\partial \mathbf{c}}{\partial \mathbf{b}}, \mathbf{b}^{(1)} > \quad , && (27)
\end{aligned}$$

where

$$\begin{aligned}
A \cdot < \frac{\partial \mathbf{c}}{\partial A}, A^{(1)} > &= -A^{(1)} \cdot \mathbf{c} \quad , && (28)\\
A \cdot < \frac{\partial \mathbf{c}}{\partial \mathbf{b}}, \mathbf{b}^{(1)} > &= \mathbf{b}^{(1)} \quad . && (29)
\end{aligned}$$

For the computation of Equation (27) the matrices $A, A^{(1)} \in I\!R^{n \times (n+m)}$ are assumed to be serialized. Therefore, differentiation of $\mathbf{c} \in I\!R^{n+m}$ with respect to $A \in I\!R^{n^2+n\cdot m}$, gives a matrix $\frac{\partial \mathbf{c}}{\partial A} \in I\!R^{(n+m)\times(n^2+n\cdot m)}$. Projecting this matrix in direction $A^{(1)} \in I\!R^{n^2+n\cdot m}$ yields $< \frac{\partial \mathbf{c}}{\partial A}, A^{(1)} > \in I\!R^{n+m}$.

Computing $A^{(1)}$, $\mathbf{b}^{(1)}$ and $\mathbf{c}^{(1)}$ in Equation (26) yields

$$
\begin{aligned}
A &= \nabla F \quad , \\
A^{(1)} &=< \frac{d\,(\nabla F)}{d\boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} > \\
&=< \frac{\partial(\nabla F)}{\partial \boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} > + < \frac{\partial(\nabla F)}{\partial \mathbf{x}}, \underbrace{< \frac{\partial \mathbf{x}}{\partial \boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} >}_{=\mathbf{x}^{(2)}} > \\
&=< \frac{\partial(\nabla F)}{\partial \mathbf{x} \partial \boldsymbol{\lambda}}, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > + < \frac{\partial(\nabla F)}{\partial \mathbf{x} \partial \boldsymbol{\lambda}}, \begin{pmatrix} \mathbf{x}^{(2)} \\ \mathbf{0}_m \end{pmatrix} > \\
&=< \nabla^2 F, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > + < \nabla^2 F, \begin{pmatrix} \mathbf{x}^{(2)} \\ \mathbf{0}_m \end{pmatrix} > \\
&=< \nabla^2 F, \begin{pmatrix} \mathbf{x}^{(2)} \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > \quad ,
\end{aligned}
$$

where $A, A^{(1)} \in I\!R^{n \times (n+m)}$. Moreover,

$$
\begin{aligned}
\mathbf{b} &= - < \nabla F, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix} > \quad , \\
\mathbf{b}^{(1)} &= \frac{d}{d\boldsymbol{\lambda}} \left(- < \nabla F, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix} > \right) \cdot \boldsymbol{\lambda}^{(2)} \\
&= - < \frac{\partial < \nabla F, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix} >}{\partial \boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} > - < \frac{\partial < \nabla F, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix} >}{\partial \mathbf{x}}, \overbrace{< \frac{\partial \mathbf{x}}{\partial \boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} >}^{=\mathbf{x}^{(2)}} > \\
&= - < \frac{\partial < \nabla F, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix} >}{\partial \mathbf{x} \partial \boldsymbol{\lambda}}, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > - < \frac{\partial < \nabla F, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix} >}{\partial \mathbf{x} \partial \boldsymbol{\lambda}}, \begin{pmatrix} \mathbf{x}^{(2)} \\ \mathbf{0}_m \end{pmatrix} > \\
&= - < \nabla^2 F, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix}, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > - < \nabla^2 F, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix}, \begin{pmatrix} \mathbf{x}^{(2)} \\ \mathbf{0}_m \end{pmatrix} > \\
&= - < \nabla^2 F, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix}, \begin{pmatrix} \mathbf{x}^{(2)} \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > \quad ,
\end{aligned}
$$

and

$$
\begin{aligned}
\mathbf{c} &= \begin{pmatrix} \mathbf{x}^{(1)} \\ \mathbf{0}_m \end{pmatrix} \quad , \\
\mathbf{c}^{(1)} &= \frac{d}{d\boldsymbol{\lambda}} \begin{pmatrix} \mathbf{x}^{(1)} \\ \mathbf{0}_m \end{pmatrix} \cdot \boldsymbol{\lambda}^{(2)} = \frac{\partial \begin{pmatrix} \mathbf{x}^{(1)} \\ \mathbf{0}_m \end{pmatrix}}{\partial \boldsymbol{\lambda}} \cdot \boldsymbol{\lambda}^{(2)} \\
&= \begin{pmatrix} \frac{\partial \mathbf{x}^{(1)}}{\partial \boldsymbol{\lambda}} \cdot \boldsymbol{\lambda}^{(2)} \\ \mathbf{0}_m \end{pmatrix} = \begin{pmatrix} \mathbf{x}^{(1,2)} \\ \mathbf{0}_m \end{pmatrix} \quad .
\end{aligned}
$$

Now applying Equations (28)-(29) to the linear system in Equation (26) we have

$$\begin{aligned}
\nabla F \cdot < \frac{\partial \mathbf{c}}{\partial A}, A^{(1)} > &= - < \nabla^2 F, \begin{pmatrix} \mathbf{x}^{(2)} \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > \cdot \begin{pmatrix} \mathbf{x}^{(1)} \\ \mathbf{0}_m \end{pmatrix} \\
&= - < \nabla^2 F, \begin{pmatrix} \mathbf{x}^{(2)} \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix}, \begin{pmatrix} \mathbf{x}^{(1)} \\ \mathbf{0}_m \end{pmatrix} > \quad , \\
\nabla F \cdot < \frac{\partial \mathbf{c}}{\partial \mathbf{b}}, \mathbf{b}^{(1)} > &= - < \nabla^2 F, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix}, \begin{pmatrix} \mathbf{x}^{(2)} \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > \quad .
\end{aligned}$$

Consequently, Equation (27) yields

$$\begin{aligned}
\begin{pmatrix} \mathbf{x}^{(1,2)} \\ \mathbf{0}_m \end{pmatrix} &= - (\nabla F)^{-1} \cdot < \nabla^2 F, \begin{pmatrix} \mathbf{x}^{(2)} \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix}, \begin{pmatrix} \mathbf{x}^{(1)} \\ \mathbf{0}_m \end{pmatrix} > \\
&\quad - (\nabla F)^{-1} \cdot < \nabla^2 F, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix}, \begin{pmatrix} \mathbf{x}^{(2)} \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > \\
\nabla F \cdot \begin{pmatrix} \mathbf{x}^{(1,2)} \\ \mathbf{0}_m \end{pmatrix} &= - < \nabla^2 F, \begin{pmatrix} \mathbf{x}^{(2)} \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix}, \begin{pmatrix} \mathbf{x}^{(1)} \\ \mathbf{0}_m \end{pmatrix} > - < \nabla^2 F, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix}, \begin{pmatrix} \mathbf{x}^{(2)} \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > \\
\nabla F \cdot \begin{pmatrix} \mathbf{x}^{(1,2)} \\ \mathbf{0}_m \end{pmatrix} &= - < \nabla^2 F, \begin{pmatrix} \mathbf{x}^{(1)} \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix}, \begin{pmatrix} \mathbf{x}^{(2)} \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > \\
\frac{\partial F}{\partial \mathbf{x}} \cdot \mathbf{x}^{(1,2)} &= - < \nabla^2 F, \begin{pmatrix} \mathbf{x}^{(1)} \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix}, \begin{pmatrix} \mathbf{x}^{(2)} \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > \quad .
\end{aligned}$$

Both $\frac{\partial F}{\partial \mathbf{x}}$ and the right hand side can be computed automatically by AD.

The required memory for evaluating the symbolic second-order tangent nonlinear solver is the memory required by the nonlinear solver itself (e.g. Equation (7) in Newton's algorithm), i.e. $MEM(\mathbf{L}) \sim O(n^2)$. In Equation (19) the complexity of evaluating the right hand side is $O(1) \cdot Cost(F)$. The decomposition of the Jacobian ($\frac{\partial F}{\partial \mathbf{x}}$) which is done in the evaluation of the first-order partial derivatives with symbolic tangent (Equation (20) or Equation (25)) at the cost of $O(n^3)$ can also be used in evaluating the second-order directional derivatives. Solving the linear system (Equation (19)) e.g. with forward/backward substitution at the cost of $O(n^2)$, the overall complexity of evaluating the symbolic second-order tangent directional derivatives $\mathbf{x}^{(1,2)}$ is proportional to $O(n^3)$.

5 Second-Order Adjoint Nonlinear Solver

As in Section 4 we distinguish between second-order algorithmic and symbolic modes when deriving adjoint solvers for systems of nonlinear equations. Similar remarks regarding numerical consistency between the primal and the adjoint solvers apply.

5.1 Algorithmic Mode

As mentioned in the previous section, solving the nonlinear system (Equation (1)) with e.g. Newton, the nonlinear solver uses the Jacobian of the nonlinear system

($\nabla F(\mathbf{x}^i)$) at the current iterate $\mathbf{x}^i$ to determine the next Newton step. First-order adjoint version of Newton's algorithm requires second directional derivatives of the residual. Consequently, second-order adjoint version of the Newton's algorithm requires the third directional derivatives of the given implementation of F.

It should be considered that the memory requirement for the algorithmic adjoint nonlinear solver becomes proportional to the number of operations performed by the nonlinear solver. Data required within the reverse section is recorded in the forward section. The resulting memory requirement is likely to exceed the available resources for most real-world applications. Checkpointing techniques can help keeping the required memory feasible at the expense of additional function evaluations, See [GW08,Nau12] for details.

As it is shown in [NLLT12], the first-order algorithmic adjoint version of the given objective with Newton ' s algorithm which is used for the solution of the embedded parametrized systems of nonlinear equations results from the straight application of adjoint mode AD to Equations (6)–(8). The application of (incremental) adjoint mode AD to Equations (6)–(8) (without checkpointing) yields

for $i = 0, \ldots, \nu$

$$\begin{aligned}
(A, \tau) &:= \frac{\partial F}{\partial \mathbf{x}}(\mathbf{x}^i, \boldsymbol{\lambda}) && (30)\\
(\mathbf{b}, \tau) &:= -F(\mathbf{x}^i, \boldsymbol{\lambda}) \\
(\mathbf{s}, \tau) &:= \mathbf{L}(A, \mathbf{b}) && (31)\\
\mathbf{x}^{i+1} &:= \mathbf{x}^i + \mathbf{s} && (32)
\end{aligned}$$

for $i = \nu, \ldots, 0$

$$\begin{aligned}
\mathbf{x}^i_{(1)} &:= \mathbf{s}_{(1)} := \mathbf{x}^{i+1}_{(1)} \\
\begin{pmatrix} A_{(1)} \\ \mathbf{b}_{(1)} \end{pmatrix} &:= \mathbf{L}_{(1)}(\mathbf{s}_{(1)}, \tau) \\
\begin{pmatrix} \mathbf{x}^i_{(1)} \\ \boldsymbol{\lambda}_{(1)} \end{pmatrix} &:= \begin{pmatrix} \mathbf{x}^i_{(1)} \\ \boldsymbol{\lambda}_{(1)} \end{pmatrix} + < \mathbf{b}_{(1)}, \frac{\partial F}{\partial(\mathbf{x}, \lambda)}(\mathbf{x}^i, \boldsymbol{\lambda}) > (\tau) && (33)\\
&\quad + < A_{(1)}, \frac{\partial^2 F}{\partial \mathbf{x} \partial(\mathbf{x}, \boldsymbol{\lambda})}(\mathbf{x}^i, \boldsymbol{\lambda}) > (\tau).
\end{aligned}$$

Data required within the reverse section is recorded on a data structure [2] τ in the augmented forward section (Equations (30)–(32)). The input value of $\boldsymbol{\lambda}_{(1)}$ depends on the context in which the nonlinear solver is called. In the specific scenario given by Equations (3)–(5) it is initially equal to zero as adjoints of intermediate (neither input nor output) variables should be; see, for example,

[2] Using AD overloading tool, e.g. dco (Derivative Code by Overloading), datas required within the reverse section will be recorded on tape, whereas by using AD source transformation tool, e.g. dcc (Derivative Code Compiler), datas required within the reverse section will be recorded on stack.

[GW08]. In Equation (33) the projections of $\frac{\partial F}{\partial(\mathbf{x},\boldsymbol{\lambda})}(\mathbf{x}^i,\boldsymbol{\lambda})$ and $\frac{\partial^2 F}{\partial\mathbf{x}\partial(\mathbf{x},\boldsymbol{\lambda})}(\mathbf{x}^i,\boldsymbol{\lambda})$ in directions $\mathbf{b}_{(1)}$ and $A_{(1)}$ respectively depend on τ, i.e., the projections on the reverse section are dependent on the datas recorded on tape during the forward section.

Both the Jacobian accumulation in Equation (30) and the linear solver in Equation (31) are treated straightforwardly with an application of AD software.

As mensioned above, datas in the forward section are recorded on tape to be used in the calculations on the reverse section. In the following for simplicity and better readability we omit the τ.

Applying second-order algorithmic adjoint of AD to Equations (6)–(8) yields

for $i = 0, \ldots, \nu$

$$\begin{aligned}
A &:= \frac{\partial F}{\partial \mathbf{x}}(\mathbf{x}^i,\boldsymbol{\lambda}) \\
A^{(2)} &:=< \frac{\partial^2 F}{\partial\mathbf{x}\partial(\mathbf{x},\boldsymbol{\lambda})}(\mathbf{x}^i,\boldsymbol{\lambda}), \begin{pmatrix} \mathbf{x}^{i\ (2)} \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > \\
\mathbf{b} &:= -F(\mathbf{x}^i,\boldsymbol{\lambda}) \\
\mathbf{b}^{(2)} &:= - < \frac{\partial F}{\partial(\mathbf{x},\boldsymbol{\lambda})}(\mathbf{x}^i,\boldsymbol{\lambda}), \begin{pmatrix} \mathbf{x}^{i\ (2)} \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > \\
\mathbf{s} &:= \mathbf{L}(A,\mathbf{b}) \\
\mathbf{s}^{(2)} &:=< \frac{\partial \mathbf{L}}{\partial(A,\mathbf{b})}(A,\mathbf{b}), \begin{pmatrix} A^{(2)} \\ \mathbf{b}^{(2)} \end{pmatrix} > \\
\mathbf{x}^{i+1} &:= \mathbf{x}^i + \mathbf{s} \\
\mathbf{x}^{i+1\ (2)} &:= \mathbf{x}^{i\ (2)} + \mathbf{s}^{(2)} \quad ,
\end{aligned}$$

followed by

for $i = \nu, \ldots, 0$

$$\begin{aligned}
\mathbf{x}^i_{(1)} &:= \mathbf{s}_{(1)} := \mathbf{x}^{i+1}_{(1)} \\
\mathbf{s}^{(2)}_{(1)} &:= \mathbf{s}^{(2)}_{(1)} + \mathbf{x}^{i+1(2)}_{(1)} \\
\mathbf{x}^{(2)}_{(1)} &:= \mathbf{x}^{(2)}_{(1)} + \mathbf{x}^{i+1(2)}_{(1)} \\
\begin{pmatrix} A_{(1)} \\ \mathbf{b}_{(1)} \end{pmatrix} &:=< \mathbf{s}_{(1)}, \frac{\partial \mathbf{L}}{\partial(A,\mathbf{b})}(A,\mathbf{b}) > \\
\begin{pmatrix} A^{(2)}_{(1)} \\ \mathbf{b}^{(2)}_{(1)} \end{pmatrix} &:= \begin{pmatrix} A^{(2)}_{(1)} \\ \mathbf{b}^{(2)}_{(1)} \end{pmatrix} + < \mathbf{s}^{(2)}_{(1)}, \frac{\partial \mathbf{L}}{\partial(A,\mathbf{b})}(A,\mathbf{b}) > \\
&\qquad + < \mathbf{s}_{(1)}, \frac{\partial^2 \mathbf{L}}{\partial(A,\mathbf{b})^2}(A,\mathbf{b}), \begin{pmatrix} A^{(2)} \\ \mathbf{b}^{(2)} \end{pmatrix} >
\end{aligned}$$

$$\begin{pmatrix}\mathbf{x}^i_{(1)}\\ \boldsymbol{\lambda}_{(1)}\end{pmatrix} := \begin{pmatrix}\mathbf{x}^i_{(1)}\\ \boldsymbol{\lambda}_{(1)}\end{pmatrix} + < A_{(1)}, \frac{\partial^2 F}{\partial \mathbf{x}\partial(\mathbf{x},\boldsymbol{\lambda})}(\mathbf{x}^i,\boldsymbol{\lambda}) > \\ - < \mathbf{b}_{(1)}, \frac{\partial F}{\partial(\mathbf{x},\boldsymbol{\lambda})}(\mathbf{x}^i,\boldsymbol{\lambda}) > \tag{34}$$

$$\begin{pmatrix}\mathbf{x}^{i(2)}_{(1)}\\ \boldsymbol{\lambda}^{(2)}_{(1)}\end{pmatrix} := \begin{pmatrix}\mathbf{x}^{i(2)}_{(1)}\\ \boldsymbol{\lambda}^{(2)}_{(1)}\end{pmatrix} + < A^{(2)}_{(1)}, \frac{\partial^2 F}{\partial \mathbf{x}\partial(\mathbf{x},\boldsymbol{\lambda})}(\mathbf{x}^i,\boldsymbol{\lambda}) > \\ - < \mathbf{b}^{(2)}_{(1)}, \frac{\partial F}{\partial(\mathbf{x},\boldsymbol{\lambda})}(\mathbf{x}^i,\boldsymbol{\lambda}) > \\ + < A_{(1)}, \frac{\partial^3 F}{\partial \mathbf{x}\partial(\mathbf{x},\boldsymbol{\lambda})^2}(\mathbf{x}^i,\boldsymbol{\lambda}), \begin{pmatrix}\mathbf{x}^{i(2)}\\ \boldsymbol{\lambda}^{(2)}\end{pmatrix} > \\ - < \mathbf{b}_{(1)}, \frac{\partial^2 F}{\partial(\mathbf{x},\boldsymbol{\lambda})^2}(\mathbf{x}^i,\boldsymbol{\lambda}), \begin{pmatrix}\mathbf{x}^{i(2)}\\ \boldsymbol{\lambda}^{(2)}\end{pmatrix} > \tag{35}$$

where all the derivatives of F, e.g. $\frac{\partial F}{\partial(\mathbf{x},\boldsymbol{\lambda})}$ are evaluated at point $(\mathbf{x}^i,\boldsymbol{\lambda})$. The same as previous section, the differentiation of $\mathbf{L}(A,\mathbf{b}) \in I\!R^n$ with respect to $(A,\mathbf{b})$ is done through serialization of $(A,\mathbf{b})$, meaning that $A \in I\!R^{n\times n}$ and $\mathbf{b} \in I\!R^n$, $(A,\mathbf{b})$ is considered as a vector of size n^2+n. Consequently, $\frac{\partial \mathbf{L}}{\partial(A,\mathbf{b})}(A,\mathbf{b}) \in I\!R^{n\times(n^2+n)}$ and $\frac{\partial^2 \mathbf{L}}{\partial(A,\mathbf{b})^2}(A,\mathbf{b}) \in I\!R^{n\times(n^2+n)\times(n^2+n)}$. The serialization is also applied for $\begin{pmatrix}A_{(1)}\\ \mathbf{b}_{(1)}\end{pmatrix}$, $\begin{pmatrix}A^{(2)}\\ \mathbf{b}^{(2)}\end{pmatrix}$ and $\begin{pmatrix}A^{(2)}_{(1)}\\ \mathbf{b}^{(2)}_{(1)}\end{pmatrix} \in I\!R^{n^2+n}$. Furthermore, the expressions $< A_{(1)}, \frac{\partial^2 F}{\partial \mathbf{x}\partial(\mathbf{x},\boldsymbol{\lambda})}(\mathbf{x}^i,\boldsymbol{\lambda}) >$ in Equation (34) and $< A^{(2)}_{(1)}, \frac{\partial^2 F}{\partial \mathbf{x}\partial(\mathbf{x},\boldsymbol{\lambda})}(\mathbf{x}^i,\boldsymbol{\lambda}) >$ in Equation (35) denote a projection image dimension of length $n^2(\Leftarrow n\times n)$ of the first derivative of the Jacobian $\frac{\partial F}{\partial \mathbf{x}}(\mathbf{x}^i,\boldsymbol{\lambda})$ with respect to $\mathbf{x}$ and $\boldsymbol{\lambda}$ (the Hessian $\frac{\partial^2 F}{\partial \mathbf{x}\partial(\mathbf{x},\boldsymbol{\lambda})}(\mathbf{x}^i,\boldsymbol{\lambda})$) in the direction obtained by a corresponding serialization of $A_{(1)}$ and $A^{(2)}_{(1)}$ respectively. The expression $< A_{(1)}, \frac{\partial^3 F}{\partial \mathbf{x}\partial(\mathbf{x},\boldsymbol{\lambda})^2}(\mathbf{x}^i,\boldsymbol{\lambda}), \begin{pmatrix}\mathbf{x}^{i(2)}\\ \boldsymbol{\lambda}^{(2)}\end{pmatrix} >$ in Equation (35) denotes a projection image dimension of length $n^2(\Leftarrow n\times n)$ of the second derivative of the Jacobian $\frac{\partial F}{\partial \mathbf{x}}(\mathbf{x}^i,\boldsymbol{\lambda})$ with respect to $\mathbf{x}$ and $\boldsymbol{\lambda}$ (the 4-tensor $\frac{\partial^3 F}{\partial \mathbf{x}\partial(\mathbf{x},\boldsymbol{\lambda})^2}(\mathbf{x}^i,\boldsymbol{\lambda})$) in the direction obtained by a corresponding serialization of $A_{(1)}$ and in direction $\begin{pmatrix}\mathbf{x}^{i(2)}\\ \boldsymbol{\lambda}^{(2)}\end{pmatrix}$.

In the above equations, first, second and third derivatives of F with respect to $(\mathbf{x}^i,\boldsymbol{\lambda})$ are required when computing third-order adjoints of F with respect to $(\mathbf{x}^i,\boldsymbol{\lambda})$ using AD software tools, the function value as well as derivatives up to third-order are evaluated.

In this case, the number of operations is four times the operations (OPS) performed by the nonlinear solver itself, i.e., $\nu\cdot O(n^3)$. The required memory in this case is proportional to the number of operations, i.e., $\nu\cdot O(n^3)$. All these computations can be done automatically using AD software tools.

5.2 Symbolic Mode

Theorem 2. Symbolic Second-Order Adjoint Solvers of Nonlinear Equation: *Let* $\mathbf{r} = F(\mathbf{x}(\boldsymbol{\lambda}),\boldsymbol{\lambda}) : I\!R^n \times I\!R^m \to I\!R^n$ *for a given* $\boldsymbol{\lambda} \in I\!R^m$*, a vector* $\mathbf{x} \in R^n$ *is sought such that* $F(\mathbf{x}(\boldsymbol{\lambda}),\boldsymbol{\lambda}) = 0$*. Let* $\mathbf{z} \in I\!R^n$ *and*

$$\frac{\partial F}{\partial \mathbf{x}}^T(\mathbf{x},\boldsymbol{\lambda})\cdot \mathbf{z} = -\mathbf{x}_{(1)} \quad . \tag{36}$$

Furthermore, let $\mathbf{z}^{(2)} =< \frac{\partial \mathbf{z}}{\partial \boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} >$ *,* $\mathbf{z}^{(2)} \in I\!R^n$ *and*

$$\frac{\partial F}{\partial \mathbf{x}}^T \cdot \mathbf{z}^{(2)} = -< \mathbf{z}, \nabla^2 F, \begin{pmatrix}\mathbf{x}^{(2)}\\ \boldsymbol{\lambda}^{(2)}\end{pmatrix} > -\mathbf{x}^{(2)}_{(1)} \quad .$$

Second-order adjoint differentiation of $F(\mathbf{x},\boldsymbol{\lambda}) = 0$ *at the solution* $\mathbf{x} = \mathbf{x}^*$ *with respect to* $\boldsymbol{\lambda}$*, i.e., computation of* $\boldsymbol{\lambda}^{(2)}_{(1)} \in I\!R^m$*, yields*

$$\begin{pmatrix}\mathbf{0}_n\\ \boldsymbol{\lambda}^{(2)}_{(1)}\end{pmatrix} + =< \mathbf{z}, \nabla^2 F, \begin{pmatrix}\mathbf{x}^{(2)}\\ \boldsymbol{\lambda}^{(2)}\end{pmatrix} > + < \mathbf{z}^{(2)}, \nabla F > \quad . \tag{37}$$

Proof (Version 1). As shown in [NLLT12], first-order symbolic adjoint differentiation of $F(\mathbf{x}(\boldsymbol{\lambda}),\boldsymbol{\lambda}) = 0$ at the solution $\mathbf{x} = \mathbf{x}^*$ with respect to $\boldsymbol{\lambda}$, i.e., computation of $\boldsymbol{\lambda}_{(1)}$, yields

$$< \mathbf{z}, \frac{\partial F}{\partial \mathbf{x}}(\mathbf{x},\boldsymbol{\lambda}) >= -\mathbf{x}_{(1)} \quad , \tag{38}$$

$$\boldsymbol{\lambda}_{(1)} + =< \mathbf{z}, \frac{\partial F}{\partial \boldsymbol{\lambda}}(\mathbf{x},\boldsymbol{\lambda}) > \quad , \tag{39}$$

where Equation (38) is a linear system based on transposed Jacobian of F with respect to $\mathbf{x}$, followed by Equation (39) for computing $\boldsymbol{\lambda}_{(1)}$.

An alternative for evaluating the second directional derivatives of the solution $\mathbf{x} = \mathbf{x}^*$ with respect to $\boldsymbol{\lambda}$ in $F(\mathbf{x},\boldsymbol{\lambda}) = 0$ is to differentiate Equations (38)–(39) with respect to $\boldsymbol{\lambda}$. Differentiating Equation (38) yields

$$\frac{d}{d\boldsymbol{\lambda}} < \mathbf{z}, \frac{\partial F}{\partial \mathbf{x}}(\mathbf{x},\boldsymbol{\lambda}) > \cdot \boldsymbol{\lambda}^{(2)} = \frac{d}{d\boldsymbol{\lambda}}(-\mathbf{x}_{(1)}) \cdot \boldsymbol{\lambda}^{(2)} \quad .$$

Applying Theorem 1 to the above equation yields

$$< \frac{d}{d\boldsymbol{\lambda}}(\mathbf{z}) \cdot \boldsymbol{\lambda}^{(2)}, \frac{\partial F}{\partial \mathbf{x}} > + < \mathbf{z}, \frac{d}{d\boldsymbol{\lambda}}(\frac{\partial F}{\partial \mathbf{x}}) \cdot \boldsymbol{\lambda}^{(2)} >= -\frac{\partial \mathbf{x}_{(1)}}{\partial \boldsymbol{\lambda}} \cdot \boldsymbol{\lambda}^{(2)}$$

$$<< \frac{\partial \mathbf{z}}{\partial \boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} >, \frac{\partial F}{\partial \mathbf{x}} > + < \mathbf{z}, \frac{\partial^2 F}{\partial \mathbf{x} \partial \boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} > + < \mathbf{z}, \frac{\partial^2 F}{\partial \mathbf{x}^2}, < \frac{\partial \mathbf{x}}{\partial \boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} >> \\ = - < \frac{\partial \mathbf{x}_{(1)}}{\partial \boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} >$$

$$< \mathbf{z}^{(2)}, \frac{\partial F}{\partial \mathbf{x}} > + < \mathbf{z}, \frac{\partial^2 F}{\partial \mathbf{x} \partial \boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} > + < \mathbf{z}, \frac{\partial^2 F}{\partial \mathbf{x}^2}, \mathbf{x}^{(2)} >= -\mathbf{x}^{(2)}_{(1)}$$

and hence

$$\frac{\partial F}{\partial \mathbf{x}}^T \cdot \mathbf{z}^{(2)} = - < \mathbf{z}, \frac{\partial^2 F}{\partial \mathbf{x} \partial \boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} > - < \mathbf{z}, \frac{\partial^2 F}{\partial \mathbf{x}^2}, \mathbf{x}^{(2)} > -\mathbf{x}^{(2)}_{(1)} \tag{40}$$

$$\begin{aligned}\nabla^T F \cdot \mathbf{z}^{(2)} &= - < \mathbf{z}, \nabla^2 F, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > - < \mathbf{z}, \nabla^2 F, \begin{pmatrix} \mathbf{x}^{(2)} \\ \mathbf{0}_m \end{pmatrix} > - \begin{pmatrix} \mathbf{x}_{(1)}^{(2)} \\ \mathbf{0}_m \end{pmatrix} \\ &= - < \mathbf{z}, \nabla^2 F, \begin{pmatrix} \mathbf{x}^{(2)} \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > - \begin{pmatrix} \mathbf{x}_{(1)}^{(2)} \\ \mathbf{0}_m \end{pmatrix} \quad , \end{aligned} \tag{41}$$

which is a linear system that can be solved by using a linear solver in order to evaluate $\mathbf{z}^{(2)} \in I\!R^n$. The right hand side can be evaluated in AD by projection of the Hessian ($\nabla^2 F$) in directions $\mathbf{z}$ and $\begin{pmatrix} \mathbf{x}^{(2)} \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix}$. Furthermore, $\mathbf{x}_{(1)}^{(2)}$ can also be calculated in AD.

Differentiation of Equation (39) with respect to $\boldsymbol{\lambda}$ yields

$$\begin{aligned} \frac{d}{d\boldsymbol{\lambda}}(\boldsymbol{\lambda}_{(1)}) \cdot \boldsymbol{\lambda}^{(2)} + &= \frac{d}{d\boldsymbol{\lambda}} < \mathbf{z}, \frac{\partial F}{\partial \boldsymbol{\lambda}} > \cdot \boldsymbol{\lambda}^{(2)} \\ \frac{d\boldsymbol{\lambda}_{(1)}}{d\boldsymbol{\lambda}} \cdot \boldsymbol{\lambda}^{(2)} + &= < \frac{d}{d\boldsymbol{\lambda}}(\mathbf{z}) \cdot \boldsymbol{\lambda}^{(2)}, \frac{\partial F}{\partial \boldsymbol{\lambda}} > + < \mathbf{z}, \frac{d}{d\boldsymbol{\lambda}}(\frac{\partial F}{\partial \boldsymbol{\lambda}}) \cdot \boldsymbol{\lambda}^{(2)} > \\ < \frac{\partial \boldsymbol{\lambda}_{(1)}}{\partial \boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} > + &= << \frac{\partial \mathbf{z}}{\partial \boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} >, \frac{\partial F}{\partial \boldsymbol{\lambda}} > + < \mathbf{z}, < \frac{\partial^2 F}{\partial \boldsymbol{\lambda}^2}, \boldsymbol{\lambda}^{(2)} >> \\ &\quad + < \mathbf{z}, \frac{\partial^2 F}{\partial \boldsymbol{\lambda} \partial \mathbf{x}}, < \frac{\partial \mathbf{x}}{\partial \boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} >> \\ \boldsymbol{\lambda}_{(1)}^{(2)} + &= < \mathbf{z}^{(2)}, \frac{\partial F}{\partial \boldsymbol{\lambda}} > + < \mathbf{z}, \frac{\partial^2 F}{\partial \boldsymbol{\lambda}^2}, \boldsymbol{\lambda}^{(2)} > + < \mathbf{z}, \frac{\partial^2 F}{\partial \boldsymbol{\lambda} \partial \mathbf{x}}, \mathbf{x}^{(2)} > \\ \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}_{(1)}^{(2)} \end{pmatrix} + &= < \mathbf{z}^{(2)}, \frac{\partial F}{\partial (\mathbf{x}, \boldsymbol{\lambda})} > + < \mathbf{z}, \nabla^2 F, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > + < \mathbf{z}, \nabla^2 F, \begin{pmatrix} \mathbf{x}^{(2)} \\ \mathbf{0}_m \end{pmatrix} > \end{aligned} \tag{42}$$

$$\begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}_{(1)}^{(2)} \end{pmatrix} + = < \mathbf{z}^{(2)}, \nabla F > + < \mathbf{z}, \nabla^2 F, \begin{pmatrix} \mathbf{x}^{(2)} \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > \quad . \tag{43}$$

Evaluation of the above equation can also be done in AD by projection of the Jacobian (∇F) in direction $\mathbf{z}^{(2)}$. The $< \mathbf{z}, \nabla^2 F, \begin{pmatrix} \mathbf{x}^{(2)} \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} >$ projection is already computed in Equation (41).

Proof (Version 2).
As shown in [NLLT12], first-order symbolic adjoint differentiation of $F(\mathbf{x}(\boldsymbol{\lambda}), \boldsymbol{\lambda}) = 0$ at the solution $\mathbf{x} = \mathbf{x}^*$ with respect to $\boldsymbol{\lambda}$, i.e., the computation of $\boldsymbol{\lambda}_{(1)}$, yields

$$< \mathbf{z}, \frac{\partial F}{\partial \mathbf{x}}(\mathbf{x}, \boldsymbol{\lambda}) >= -\mathbf{x}_{(1)} \quad , \tag{44}$$

$$\boldsymbol{\lambda}_{(1)} + =< \mathbf{z}, \frac{\partial F}{\partial \boldsymbol{\lambda}}(\mathbf{x}, \boldsymbol{\lambda}) > \quad . \tag{45}$$

Equation (44) becomes

$$\frac{\partial F}{\partial \mathbf{x}}^T (\mathbf{x}, \boldsymbol{\lambda}) \cdot \mathbf{z} = -\mathbf{x}_{(1)}$$

$$\frac{\partial F}{\partial \mathbf{x} \partial \boldsymbol{\lambda}}^T \cdot \mathbf{z} = -\begin{pmatrix} \mathbf{x}_{(1)} \\ \mathbf{0}_m \end{pmatrix}$$

$$\nabla F^T \cdot \mathbf{z} = -\begin{pmatrix} \mathbf{x}_{(1)} \\ \mathbf{0}_m \end{pmatrix} \quad , \tag{46}$$

which is a linear system of type $A\mathbf{c} = \mathbf{b}$, with $A = \nabla F^T$, $\mathbf{c} = \mathbf{z}$ and $\mathbf{b} = -\begin{pmatrix} \mathbf{x}_{(1)} \\ \mathbf{0}_m \end{pmatrix}$.

An option to evaluate the second derivative of the solution $\mathbf{x} = \mathbf{x}^*$ with respect to $\boldsymbol{\lambda}$ in $F(\mathbf{x}, \boldsymbol{\lambda}) = 0$ is to apply the first-order tangent symbolic version for linear solvers to Equation (44) (or Equation (46)) and then differentiate the Equation (45) with respect to $\boldsymbol{\lambda}$.

As shown in [NL12], for the linear system $A\mathbf{c} = \mathbf{b}$ we have

$$\mathbf{c} = \mathbf{L}(A, \mathbf{b}) \quad ,$$

$$\mathbf{c}^{(1)} = \mathbf{L}^{(1)}(A, A^{(1)}, \mathbf{b}, \mathbf{b}^{(1)}) = < \frac{\partial \mathbf{c}}{\partial A}, A^{(1)} > + < \frac{\partial \mathbf{c}}{\partial \mathbf{b}}, \mathbf{b}^{(1)} > \quad , \tag{47}$$

where

$$A \cdot < \frac{\partial \mathbf{c}}{\partial A}, A^{(1)} > = -A^{(1)} \cdot \mathbf{c} \quad , \tag{48}$$

$$A \cdot < \frac{\partial \mathbf{c}}{\partial \mathbf{b}}, \mathbf{b}^{(1)} > = \mathbf{b}^{(1)} \quad . \tag{49}$$

The same as before, for the computation of Equation (47) the matrices $A, A^{(1)} \in I\!R^{n \times (n+m)}$ are assumed to be serialized. Therefore, differentiation of $\mathbf{c} \in I\!R^{n+m}$ with respect to $A \in I\!R^{n^2 + n \cdot m}$, gives a matrix $\frac{\partial \mathbf{c}}{\partial A} \in I\!R^{(n+m) \times (n^2 + n \cdot m)}$. Projecting this matrix in direction $A^{(1)} \in I\!R^{n^2 + n \cdot m}$ yields $< \frac{\partial \mathbf{c}}{\partial A}, A^{(1)} > \in I\!R^{n+m}$.

In this case $A = \nabla F^T$ and $\mathbf{c} = \mathbf{z}$, therefore $A^{(1)} \cdot \mathbf{c}$ yields

$$\begin{aligned} A^{(1)} \cdot \mathbf{c} &= (\frac{d}{d\boldsymbol{\lambda}}(\nabla F^T) \cdot \boldsymbol{\lambda}^{(2)}) \cdot \mathbf{z} \\ &= < \mathbf{z}, \frac{\partial(\nabla F^T)}{\partial \boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} > + < \mathbf{z}, \frac{\partial(\nabla F^T)}{\partial \mathbf{x}}, \overbrace{< \frac{\partial \mathbf{x}}{\partial \boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} >}^{=\mathbf{x}^{(2)}} > \\ &= < \mathbf{z}, \nabla^2 F, \begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > + < \mathbf{z}, \nabla^2 F, \begin{pmatrix} \mathbf{x}^{(2)} \\ \mathbf{0}_m \end{pmatrix} > \\ &= < \mathbf{z}, \nabla^2 F, \begin{pmatrix} \mathbf{x}^{(2)} \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > \quad . \end{aligned}$$

Hence, Equation (48) yields

$$\nabla F^T \cdot < \frac{\partial \mathbf{c}}{\partial A}, A^{(1)} > = - < \mathbf{z}, \nabla^2 F, \begin{pmatrix} \mathbf{x}^{(2)} \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > \quad .$$

In this case $\mathbf{b} = -\begin{pmatrix} \mathbf{x}_{(1)} \\ \mathbf{0}_m \end{pmatrix}$, therefore $\mathbf{b}^{(1)}$ yields

$$\mathbf{b}^{(1)} = \frac{d}{d\boldsymbol{\lambda}}(-\begin{pmatrix}\mathbf{x}_{(1)}\\ \mathbf{0}_m\end{pmatrix}) \cdot \boldsymbol{\lambda}^{(2)} = -\begin{pmatrix}\frac{\partial \mathbf{x}_{(1)}}{\partial \boldsymbol{\lambda}} \cdot \boldsymbol{\lambda}^{(2)}\\ \mathbf{0}_m\end{pmatrix} = -\begin{pmatrix}\mathbf{x}^{(2)}_{(1)}\\ \mathbf{0}_m\end{pmatrix} \quad .$$

Consequently, Equation (49) yields

$$\nabla F^T \cdot < \frac{\partial \mathbf{c}}{\partial \mathbf{b}}, \mathbf{b}^{(1)} >= -\begin{pmatrix}\mathbf{x}^{(2)}_{(1)}\\ \mathbf{0}_m\end{pmatrix} \quad .$$

In this case $\mathbf{c} = \mathbf{z}$, therefore $\mathbf{c}^{(1)}$ yields

$$\mathbf{c}^{(1)} = \frac{d}{d\boldsymbol{\lambda}}(\mathbf{z}) \cdot \boldsymbol{\lambda}^{(2)} = \frac{\partial \mathbf{z}}{\partial \boldsymbol{\lambda}} \cdot \boldsymbol{\lambda}^{(2)} =< \frac{\partial \mathbf{z}}{\partial \boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} >= \mathbf{z}^{(2)} \quad .$$

Consequently, Equation (47) becomes

$$\mathbf{z}^{(2)} = (\nabla F)^{-T} \cdot \left(- < \mathbf{z}, \nabla^2 F, \begin{pmatrix}\mathbf{x}^{(2)}\\ \boldsymbol{\lambda}^{(2)}\end{pmatrix} >\right) - (\nabla F)^{-T} \cdot \begin{pmatrix}\mathbf{x}^{(2)}_{(1)}\\ \mathbf{0}_m\end{pmatrix}$$

$$\nabla F^T \cdot \mathbf{z}^{(2)} = - < \mathbf{z}, \nabla^2 F, \begin{pmatrix}\mathbf{x}^{(2)}\\ \boldsymbol{\lambda}^{(2)}\end{pmatrix} > - \begin{pmatrix}\mathbf{x}^{(2)}_{(1)}\\ \mathbf{0}_m\end{pmatrix} \quad . \tag{50}$$

Both $\frac{\partial F}{\partial \mathbf{x}}$ and the right hand side can be computed automatically by AD. The linear system can be solved by using a linear solver in order to evaluate $\mathbf{z}^{(2)}$.

The differentiation of Equation (45) with respect to $\boldsymbol{\lambda}$ is similar to *proof 1* Equations (42)–(43). As a result we have

$$\begin{pmatrix}\mathbf{0}_n\\ \boldsymbol{\lambda}^{(2)}_{(1)}\end{pmatrix} + =< \mathbf{z}^{(2)}, \nabla F > + < \mathbf{z}, \nabla^2 F, \begin{pmatrix}\mathbf{x}^{(2)}\\ \boldsymbol{\lambda}^{(2)}\end{pmatrix} > \quad .$$

The required memory for evaluating the symbolic second-order adjoint nonlinear solver is the memory required by the nonlinear solver itself (e.g. Equation (7) in Newton's algorithm), i.e. $MEM(\mathbf{L}) \sim O(n^2)$. In Equation (37), the complexity of evaluating the right hand side is $O(1) \cdot Cost(F)$. The decomposition of the Jacobian which is done in Equation (36) for the evaluation of $\mathbf{z}$ at the cost of $O(n^3)$ can also be used in evaluating the second-order directional derivatives. Solving the linear system Equation (41) (or Equation (50)) e.g. with forward/backward substitution at the cost of $O(n^2)$, the overall complexity of evaluating the symbolic second-order adjoint directional derivatives $\boldsymbol{\lambda}^{(2)}_{(1)}$ is proportional to $O(n^3)$.

6 IMPLEMENTATION

One can differentiate a system of numerical simulation by applying AD tools. If the simulation system contains a nonlinear solver, the preferable way is to differentiate the nonlinear system symbolically, for example, in our system (Equations (3)–(5)), differentiating Equation (3) and Equation (5) with algorithmic mode

(AD tools) and evaluating the derivatives of Equation (4) theoretically. This means that the theoretical results should be integrated into the existing software, which is not straightforward. For this purpose, we define an initiative generic API [3], which facilitates the exploitation of mathematical and structural knowledge inside of often highly complex tangent and adjoint numerical simulations.

Similar to [NLLT12], as a representative case study for the implementation of higher-level (user-defined) intrinsics in the context of overloading AD tools we consider the solver {S(n,x,lbd)} for systems of n nonlinear equations with inputs x=x^0 and {lbd}=$\boldsymbol{\lambda}$ and output x=x^*. More generically, the proposed approach allows users of AD tools to treat arbitrary parts of the primal code as *external functions*. The latter yield *gaps* in the tape due to their passive evaluation within the forward section of the adjoint code. These gaps need to be filled by corresponding user-defined adjoint functions to be called by the tape interpreter within the reverse section of the adjoint code. This concept is part of the overloading AD tool `dco` [NLL14].

In the following we focus on the external function interface of `dco/c++` in the context of second-order tangent and adjoint modes. The preferred method of implementation of second-order tangent external functions is through replacement of the overloaded primal function with a user-defined version. One should not expect to be presented with *the* method for filling gaps in the data flow of second-order tangent or adjoint numerical simulations. There are always several alternatives that implement mathematically equivalent functions. The particular choice made for `dco/c++` is meant to be both intuitive and easy to maintain. The overloading AD tool ADOL-C [GJU96] features a similar, but less generic external function concept.

Our simulation system (Equations (3)–(5)) yields

```
template <typename TYPE>
inline TYPE costfunction(int n, std::vector<TYPE> &z) {
    std::vector<TYPE> lambda(n), x(n, 0), residual(n);
    int iter = 0;
    preprocessor(n, z, lambda);
    if (Alg)
        iter = Alg_S(n, x, lambda);
    else
        iter = Symb_S(n, x, lambda);
    std::vector<double> x_a(n);
    generate_measurements(n, x_a);
    TYPE J = postprocessor(n, x, x_a);
    return J;
}
```

In the above implementation, λ is initialized by calling preprocessor function. After initializing λ the nonlinear solver is called, in which λ is input and $\mathbf{x}$ is input as well as output. There are two alternative implementations for the nonlinear solver. If the differentiation is done algorithmically, Alg_S function should be called, otherwise, if the differentiation is done symbolically, Symb_S function should be called. At last, the optimization function (postprocessor) is called, which optimizes the inputs (λ) with respect to the real measurements (x_a).

[3] Application Programming Interface

6.1 Algorithmic Approach

The primal nonlinear solver function S is made generic with respect to the floating-point data type {FT} yielding

```
template <class FT>
void S(int n, std::vector<FT> &x, std::vector<FT> &lbd);
```

Thus it can be instantiated with the `dco/c++` data type {dco::t2s_t1s::type} and `dco/c++` data type {dco::t2s_a1s::type} , which implement second-order tangent and adjoint modes respectively.

```
template <class T>
int Alg_S(int n, std::vector<T> &x, std::vector<T> &lambda)
{   return S(n, x, lambda); }
```

In adjoint mode, a tape of the entire computation is generated and interpreted as discussed in Section 5.1. Therefore, there is no gap in tape.

6.2 Symbolic Approach

In the following, we focus on the symbolic second-order tangent and adjoint modes for nonlinear solvers discussed in Sections 4.2 and 5.2 respectively.

Based on Fig.2 in [NLLT12], Fig. 2 in the following is the linearized DAG for second-order tangent mode of our simulation system.

In Listing 1.1, the computation of both first- and second-order tangent mode of the nonlinear solver is shown. As mentioned in Section 4.2, the first-order symbolic tangent differentiation of a nonlinear system $F(\mathbf{x}(\boldsymbol{\lambda}), \boldsymbol{\lambda}) = 0$ at the solution $\mathbf{x} = \mathbf{x}^*$ with respect to $\boldsymbol{\lambda}$, i.e., computation of $\mathbf{x}^{(1)}$, yields

$$\frac{\partial F}{\partial \mathbf{x}}(\mathbf{x}, \boldsymbol{\lambda}) \cdot \mathbf{x}^{(1)} = - < \frac{\partial F}{\partial \boldsymbol{\lambda}}(\mathbf{x}, \boldsymbol{\lambda}), \boldsymbol{\lambda}^{(1)} > \quad . \tag{51}$$

Differentiating the first-order tangent mode of the nonlinear solver results the second-order tangent mode of it, i.e.

$$\frac{\partial F}{\partial \mathbf{x}} \cdot \mathbf{x}^{(1,2)} = - < \nabla^2 F, \begin{pmatrix} \mathbf{x}^{(1)} \\ \boldsymbol{\lambda}^{(1)} \end{pmatrix}, \begin{pmatrix} \mathbf{x}^{(2)} \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > \quad . \tag{52}$$

The computation of $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(1,2)}$ amount to the solution of linear systems which are based on the Jacobian matrix $\frac{\partial F}{\partial \mathbf{x}}$.

A specialization of the generic primal solver `S` for `dco`'s scalar second-order tangent type {dco::t2s_t1s::type} is shown in Listing 1.1.

```
int Symb_S(int n, std::vector<dco::t2s_t1s::type> &x, std::vector<
    dco::t2s_t1s::type> &lambda)
{
    std::vector<dco::t2s_t1s::type> residual(n);
    std::vector<double> px(n), plambda(n), presidual(n), t1lambda(n)
        , t2lambda(n), tu1(n), tu2(n), b1(n), b2(n), out1(n), rhs(n),
        t2s_t1s_u(n);
    std::vector<std::vector<double> > J_dr_dx(n, vector<double>(n));
    int** P;
```

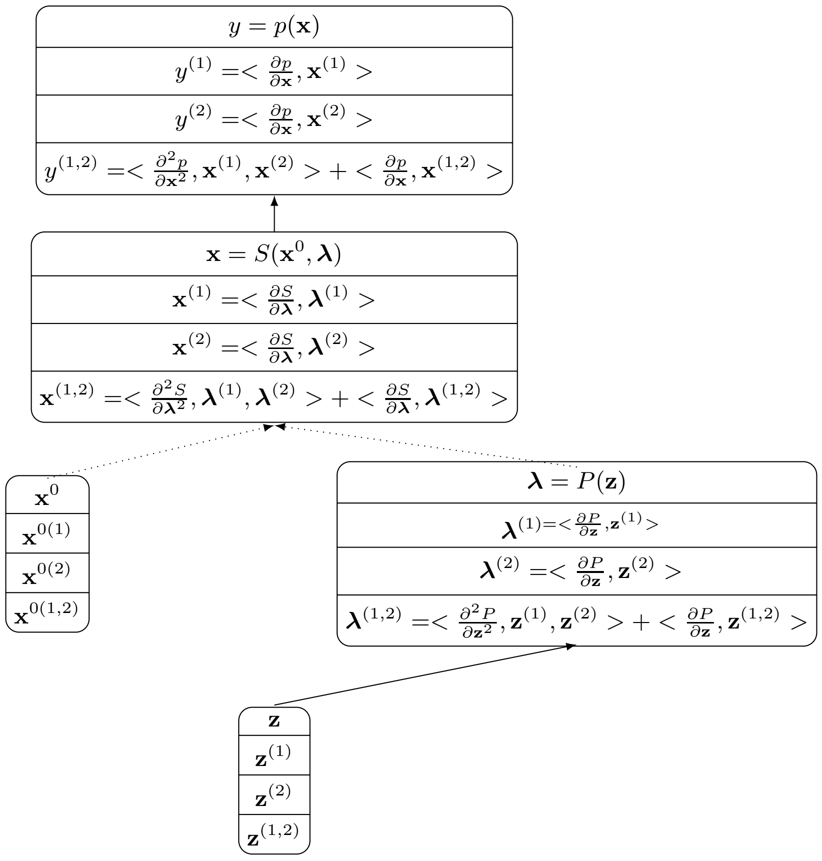

$y = p(\mathbf{x})$
$y^{(1)} =< \frac{\partial p}{\partial \mathbf{x}}, \mathbf{x}^{(1)} >$
$y^{(2)} =< \frac{\partial p}{\partial \mathbf{x}}, \mathbf{x}^{(2)} >$
$y^{(1,2)} =< \frac{\partial^2 p}{\partial \mathbf{x}^2}, \mathbf{x}^{(1)}, \mathbf{x}^{(2)} > + < \frac{\partial p}{\partial \mathbf{x}}, \mathbf{x}^{(1,2)} >$
$\mathbf{x} = S(\mathbf{x}^0, \boldsymbol{\lambda})$
$\mathbf{x}^{(1)} =< \frac{\partial S}{\partial \boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(1)} >$
$\mathbf{x}^{(2)} =< \frac{\partial S}{\partial \boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} >$
$\mathbf{x}^{(1,2)} =< \frac{\partial^2 S}{\partial \boldsymbol{\lambda}^2}, \boldsymbol{\lambda}^{(1)}, \boldsymbol{\lambda}^{(2)} > + < \frac{\partial S}{\partial \boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(1,2)} >$
$\mathbf{x}^0$
$\mathbf{x}^{0(1)}$
$\mathbf{x}^{0(2)}$
$\mathbf{x}^{0(1,2)}$
$\boldsymbol{\lambda} = P(\mathbf{z})$
$\boldsymbol{\lambda}^{(1)} =< \frac{\partial P}{\partial \mathbf{z}}, \mathbf{z}^{(1)} >$
$\boldsymbol{\lambda}^{(2)} =< \frac{\partial P}{\partial \mathbf{z}}, \mathbf{z}^{(2)} >$
$\boldsymbol{\lambda}^{(1,2)} =< \frac{\partial^2 P}{\partial \mathbf{z}^2}, \mathbf{z}^{(1)}, \mathbf{z}^{(2)} > + < \frac{\partial P}{\partial \mathbf{z}}, \mathbf{z}^{(1,2)} >$
$\mathbf{z}$
$\mathbf{z}^{(1)}$
$\mathbf{z}^{(2)}$
$\mathbf{z}^{(1,2)}$

Fig. 2. Implementation of second-order tangent mode NLS: Solid lines represent the computation of derivatives in algorithmic mode. Dotted lines denote the computation of derivatives in symbolic mode. In the given example, we have $\mathbf{x}^{0(1)} = \mathbf{x}^{0(2)} = \mathbf{x}^{0(1,2)} = 0$, therefore they are not mentioned in the proceeding computations.

```
    dco::t2s_t1s::get(x, px);
    dco::t2s_t1s::get(lambda, plambda);
    int iter = S(n, px, plambda);
    dco::t2s_t1s::get(lambda, t1lambda, 1);
    dco::t2s_t1s::get(lambda, t2lambda, 0, 2);
    dco::t2s_t1s::set(lambda, plambda);
    for (int i=0; i<n; i++)
        x[i] = px[i];
    F(n, x, lambda, residual);
    dco::t2s_t1s::get(residual, b1, 1);
    dco::t2s_t1s::get(residual, b2, 2);
    for (int ii = 0; ii < n; ii++) {
        b1[ii] *= -1;
        b2[ii] *= -1;
    }
    dr_dx_tgl(n, px, plambda, presidual, J_dr_dx);
    P = LUDecomp(n, J_dr_dx);
    FBsolve(n, P, J_dr_dx, b1, tu1);
    FBsolve(n, P, J_dr_dx, b2, tu2);
    dco::t2s_t1s::set(x, tu1, 1);
    dco::t2s_t1s::set(x, tu2, 2);
    dco::t2s_t1s::set(lambda, plambda);
    dco::t2s_t1s::set(x, px);
```

```
30      F(n, x, lambda, residual);
31      dco::t2s_t1s::get(residual, out1, 1, 2);
32      for(int i=0; i<n; i++)
33          rhs[i]= (-1)*out1[i];
34      FBsolve(n, P, J_dr_dx, rhs, t2s_t1s_u);
35      dco::t2s_t1s::set(x, t2s_t1s_u, 1, 2);
36      return iter;
37  }
```

Listing 1.1. External function for symbolic second-order tangent mode of S

In Listing 1.1, $\lambda^{(\mathbf{1})}$ and $\lambda^{(\mathbf{2})}$ are evaluated in lines 10 and 11 respectively. The function F is the nonlinear problem. Having $\lambda^{(\mathbf{1})}$ and $\lambda^{(\mathbf{2})}$, b1=$< \frac{\partial F}{\partial \boldsymbol{\lambda}}(\mathbf{x}, \boldsymbol{\lambda}), \boldsymbol{\lambda}^{(1)} >$ and b2=$< \frac{\partial F}{\partial \boldsymbol{\lambda}}(\mathbf{x}, \boldsymbol{\lambda}), \boldsymbol{\lambda}^{(2)} >$ are evaluated in lines 16 and 17 respectively. The Jacobian matrix(J_dr_dx) is computed using the algorithmic tangent mode in line 22 and in line 23 it is decomposed. In lines 24 and 25 the linear systems for $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$ in Equation (51) are solved respectively. In line 26, tu1 is set to $\mathbf{x}^{(1)}$ and in line 27, tu2 is set to $\mathbf{x}^{(2)}$. Up to this point, the first-order tangent derivatives are evaluated. The right hand side of Equation (52) is computed in line 33. The linear system in Equation (52) is solved in line 34 using the already decomposed Jacobian matrix. Finally, t2s_t1s_u is set to $\mathbf{x}^{(1,2)}$ in line 35.

Based on Fig.2 in [NLLT12], Fig. 3 in the following is the linearized DAG for second-order adjoint mode of our simulation system.

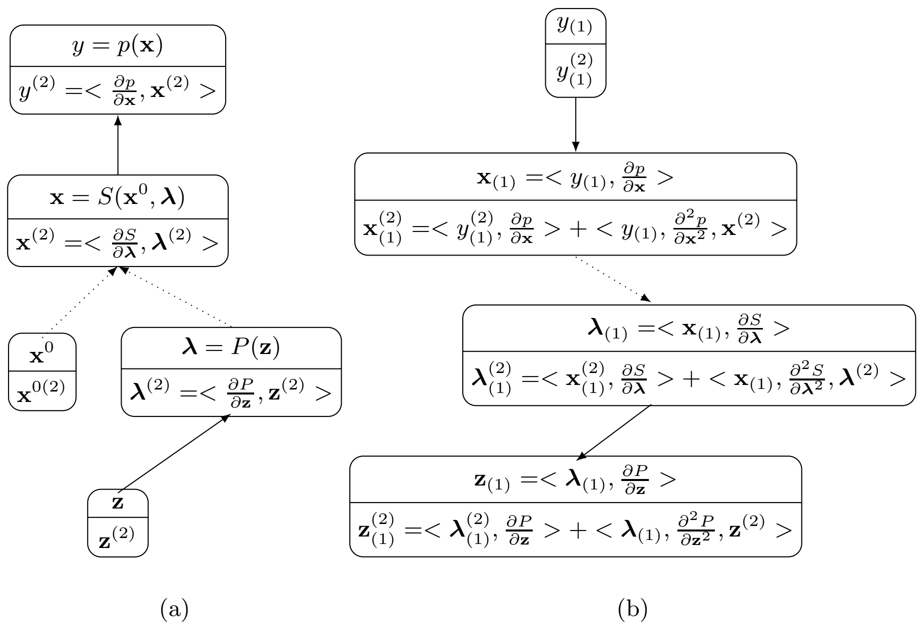

$y = p(\mathbf{x})$
$y^{(2)} =< \frac{\partial p}{\partial \mathbf{x}}, \mathbf{x}^{(2)} >$
$\mathbf{x} = S(\mathbf{x}^0, \boldsymbol{\lambda})$
$\mathbf{x}^{(2)} =< \frac{\partial S}{\partial \boldsymbol{\lambda}}, \boldsymbol{\lambda}^{(2)} >$
$\mathbf{x}^0$
$\mathbf{x}^{0(2)}$
$\boldsymbol{\lambda} = P(\mathbf{z})$
$\boldsymbol{\lambda}^{(2)} =< \frac{\partial P}{\partial \mathbf{z}}, \mathbf{z}^{(2)} >$
$\mathbf{z}$
$\mathbf{z}^{(2)}$
$y_{(1)}$
$y_{(1)}^{(2)}$
$\mathbf{x}_{(1)} =< y_{(1)}, \frac{\partial p}{\partial \mathbf{x}} >$
$\mathbf{x}_{(1)}^{(2)} =< y_{(1)}^{(2)}, \frac{\partial p}{\partial \mathbf{x}} > + < y_{(1)}, \frac{\partial^2 p}{\partial \mathbf{x}^2}, \mathbf{x}^{(2)} >$
$\boldsymbol{\lambda}_{(1)} =< \mathbf{x}_{(1)}, \frac{\partial S}{\partial \boldsymbol{\lambda}} >$
$\boldsymbol{\lambda}_{(1)}^{(2)} =< \mathbf{x}_{(1)}^{(2)}, \frac{\partial S}{\partial \boldsymbol{\lambda}} > + < \mathbf{x}_{(1)}, \frac{\partial^2 S}{\partial \boldsymbol{\lambda}^2}, \boldsymbol{\lambda}^{(2)} >$
$\mathbf{z}_{(1)} =< \boldsymbol{\lambda}_{(1)}, \frac{\partial P}{\partial \mathbf{z}} >$
$\mathbf{z}_{(1)}^{(2)} =< \boldsymbol{\lambda}_{(1)}^{(2)}, \frac{\partial P}{\partial \mathbf{z}} > + < \boldsymbol{\lambda}_{(1)}, \frac{\partial^2 P}{\partial \mathbf{z}^2}, \mathbf{z}^{(2)} >$
(a)
(b)

Fig. 3. *Mind the gap* in the tape – Implementation of second-order adjoint symbolic NLS mode: Solid lines represent the generation (in the forward section of the tangent and adjoint code shown in (a)) and interpretation (in the reverse section of the first- and second-order adjoint code shown in (b)) of the tape. Dotted lines denote gaps in the tape to be filled by a corresponding user-defined adjoint function. In the given example, we have $\mathbf{x}^{0(2)} = 0$, therefore $\mathbf{x}^{0(2)}$ is not mentioned in the proceeding computations.

In the following implementations, the computation of both first- and second-order adjoint mode of the nonlinear solver is shown. As mentioned in Section 5.2, the first-order symbolic adjoint differentiation of a nonlinear system $F(\mathbf{x}(\lambda), \lambda) = 0$ at the solution $\mathbf{x} = \mathbf{x}^*$ with respect to λ, i.e., computation of $\lambda_{(\mathbf{1})}$, yields:

$$\boldsymbol{\lambda}_{(1)} + =< \mathbf{z}, \frac{\partial F}{\partial \boldsymbol{\lambda}}(\mathbf{x}, \boldsymbol{\lambda}) >, \quad \text{where} \quad \frac{\partial F}{\partial \mathbf{x}}(\mathbf{x}, \boldsymbol{\lambda})^T \cdot \mathbf{z} = -\mathbf{x}_{(1)} \ . \tag{53}$$

Differentiating the first-order adjoint mode of the nonlinear solver results the second-order adjoint mode of it, i.e., computation of $\boldsymbol{\lambda}_{(1)}^{(2)}$, which yields

$$\begin{pmatrix} \mathbf{0}_n \\ \boldsymbol{\lambda}_{(1)}^{(2)} \end{pmatrix} + =< \mathbf{z}, \nabla^2 F, \begin{pmatrix} \mathbf{x}^{(2)} \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > + < \mathbf{z}^{(2)}, \nabla F > \quad , \tag{54}$$

where,

$$\frac{\partial F}{\partial \mathbf{x}}^T \cdot \mathbf{z}^{(2)} = - < \mathbf{z}, \nabla^2 F, \begin{pmatrix} \mathbf{x}^{(2)} \\ \boldsymbol{\lambda}^{(2)} \end{pmatrix} > - \mathbf{x}_{(1)}^{(2)} \quad . \tag{55}$$

A specialization of the generic primal solver S for dco's scalar second-order adjoint type {dco::t2s_a1s::type} marks the gap in the tape, records data that is required for filling the gap during interpretation, and runs the primal solver passively (without taping); see Listing 1.2.

```
typedef dco::t2s_a1s::type::VALUE_TYPE base_type;
typedef base_type::VALUE_TYPE base_base_type;
typedef dco::t2s_a1s::external_function_data ext_data_S;
void t2s_a1s_S(dco::t2s_a1s::external_function_data* data);

int Symb_S(int n, std::vector<dco::t2s_a1s::type> &x, std::vector<
    dco::t2s_a1s::type> &lambda)
{
    ext_data_S *data= dco::t2s_a1s::global_tape->create_ext_fcn_data
        <ext_data_S >(t2s_a1s_S);
    std::vector<base_type> plambda(n), px(n);
    std::vector<base_base_type> pplambda(n), ppx(n), b2(n),
        presidual(n), dxdl_tgl(n), t2lambda(n);
    std::vector<std::vector< base_base_type> > J_dr_dx(n, vector<
        base_base_type> (n));
    std::vector<dco::t2s_a1s::type> residual(n);
    data->register_input(lambda, plambda);
    dco::t2s_a1s::get(x, ppx);
    dco::t2s_a1s::get(lambda, pplambda);
    int its = S(n, ppx, pplambda);
    dco::t2s_a1s::get(lambda, t2lambda, 0, 2);
    dco::t1s::set(plambda, pplambda);
    for (int i=0; i<n; i++)
        x[i] = ppx[i];
    F(n, x, lambda, residual);
    dco::t2s_a1s::get(residual, b2, 0, 2);
    for (int i = 0; i < n; i++) b2[i] *= -1;
    dr_dx_tgl(n, ppx, pplambda, presidual, J_dr_dx);
    int **P = LUDecomp(n, J_dr_dx);
    FBsolve(n, P, J_dr_dx, b2, dxdl_tgl);
    for (int i=0; i<n; i++){
        px[i] = ppx[i];
29            plambda[i] = pplambda[i]; }
30        dco::t1s::set(px, dxdl_tgl, 1);
31        dco::t1s::set(plambda, t2lambda, 1);
32        data->write_to_checkpoint(n);
33        data->write_to_checkpoint(plambda);
34        data->write_to_checkpoint(px);
35        data->register_output(px, x);
36        return its;
37    }
```

Listing 1.2. External function for symbolic second-order adjoint mode of S

In Listing 1.2, data type base_type is used for overloaded first-order solution and data type base_base_type is used for explicit second-order solution. The passive evaluation of primal and tangent mode of the solver augmented with recording of data required by t2_a1_S. There is a forward declaration of t2_a1_S in line 4. In function Symb_S, the first-order tangent mode of the primal nonlinear solver (S) is computed (see Equation (51)). For this purpose, the evaluation of the Jacobian matrix is required, which is done by algorithmic tangent mode in line 24 and is decomposed in line 25. The evaluation of b1=$< \frac{\partial F}{\partial \boldsymbol{\lambda}}(\mathbf{x}, \boldsymbol{\lambda}), \boldsymbol{\lambda}^{(1)} >$ is done in line 22. The linear system in Equation (51) is solved in line 26. In lines 30 and 31, dxdl_tgl is set to $\mathbf{x}^{(1)}$ and t2lambda is set to $\boldsymbol{\lambda}^{(1)}$ respectively. At the end, the required datas for backward interpretation are written to checkpoint.

The tape interpreter fills the gap between the tapes of P and p by calling the function {t2s_a1s_S} which implements the $\boldsymbol{\lambda}_{(1)}$ and $\boldsymbol{\lambda}_{(1)}^{(2)}$ symbolically, see Listing 1.3. Refer to Fig. 3 for graphical illustration.

```
39    void t2s_a1s_S(dco::t2s_a1s::external_function_data* data)
40    {
41        int n;
42        data->read_from_checkpoint(n);
43        std::vector<double> lambda(n), x(n), a1_x(n), min_a1_x(n),
              t2lambda(n), z(n), r(n), dz_dl(n), t2_a1_x(n), dxdl(n,0),
              a1s_l(n,0), a1s_x(n,0), dxdl_tgl(n),  rhs1(n), res1(n),
              dfdl_mult_dzdl(n, 0);
44        std::vector<base_type> xb(n);
45        std::vector<dco::t2s_a1s::type> alambda(n), alambda_1(n), ax(n),
               ax_1(n), ares(n), ares_1(n);
46        int **P;
47        std::vector<std::vector<double> > J_dr_dx(n, vector<double>(n));
48        data->get_output_adjoint(xb);
49        dco::t1s::get(xb, a1_x);
50        dco::t1s::get(xb, t2_a1_x, 1);
51        std::vector<base_type> plambda(n), px(n), output(n);
52        data->read_from_checkpoint(plambda);
53        data->read_from_checkpoint(px);
54        dco::t1s::get(px, dxdl_tgl, 1);
55        dco::t1s::get(plambda, t2lambda, 1);
56        dco::t1s::get(px, x);
57        dco::t1s::get(plambda, lambda);
58        dr_dx_adj(n, x, lambda, r, J_dr_dx);
59        P = LUDecomp(n, J_dr_dx);
60        for (int i=0; i<n; i++) min_a1_x[i]=(-1)*a1_x[i];
61        FBsolveT(n, P, J_dr_dx, min_a1_x, z);
62        dco::t2s_a1s::tape::iterator pos1 = dco::t2s_a1s::global_tape->
              get_position();
63      for (int i=0; i<n; i++) {
64          alambda[i] = lambda[i];
65          ax[i] = x[i];}
66      dco::t2s_a1s::set(alambda, t2lambda , 0, 2);
67      dco::t2s_a1s::global_tape->register_variable(alambda);
68      dco::t2s_a1s::set(ax, dxdl_tgl , 0, 2);
69      dco::t2s_a1s::global_tape->register_variable(ax);
70      dco::t2s_a1s::tape::iterator pos2 = dco::t2s_a1s::global_tape->
            get_position();
71      F(n, ax, alambda, ares);
72      dco::t2s_a1s::set(ares, z, -1);
73      dco::t2s_a1s::global_tape->interpret_adjoint_to(pos2);
74      dco::t2s_a1s::get(alambda, dxdl, -1);
75      dco::t1s::set(output, dxdl);
76      dco::t2s_a1s::get(alambda, a1s_l, -1, 2);
77      dco::t2s_a1s::get(ax, a1s_x, -1, 2);
78      dco::t2s_a1s::global_tape->reset_to(pos1);
79      for (int i=0; i<n; i++)
80          rhs1[i] = (-1)*(a1s_x[i]+t2_a1_x[i]);
81      FBsolveT(n, P, J_dr_dx, rhs1, dz_dl);
82      for (int i=0; i<n; i++) {
83          alambda_1[i] = lambda[i];
84          ax_1[i] = x[i];}
85      dco::t2s_a1s::global_tape->register_variable(alambda_1);
86      dco::t2s_a1s::tape::iterator pos3 = dco::t2s_a1s::global_tape->
            get_position();
87      F(n, ax_1, alambda_1, ares_1);
88      dco::t2s_a1s::set(ares_1, dz_dl, -1);
89      dco::t2s_a1s::global_tape->interpret_adjoint_to(pos3);
90      dco::t2s_a1s::get(alambda_1, dfdl_mult_dzdl, -1);
91      dco::t2s_a1s::global_tape->reset_to(pos1);
92      for (int i=0; i<n; i++) {
93          res1[i] = a1s_l[i] + dfdl_mult_dzdl[i];
94          dco::t1s::set(output[i], res1[i], 1);
95          data->increment_input_adjoint(output[i]);
96      }
97  }
```

Listing 1.3. Adjoint function t2s_a1s_S

In Listing 1.3, the datas that were written to checkpoint will be read. In line 48 the adjoint of the output, i.e. $\mathbf{x}_{(1)}$, is evaluated and it is set to a1_x in the next line. In line 50, $\mathbf{x}_{(1)}^{(2)}$ is set to t2_a1_x. The evaluations of the $\mathbf{x}^{(1)}$ and $\boldsymbol{\lambda}^{(1)}$ that were done in Listing 1.2 in lines 30 and 31 will be read here in lines 54 and 55 respectively. The Jacobian matrix is computed with algorithmic adjoint mode in line 58 and is decomposed in the next line. In line 61 the linear system in Equation (53) is solved for computing $\mathbf{z}$ with transposed Jacobian matrix. In line 74, the first-order symbolic adjoint dxdl=$\boldsymbol{\lambda}_{(1)}$ in Equation (53) is evaluated. The right hand side of Equation (55) is computed in line 80 and the linear system for $\mathbf{z}^{(2)}$ with transposed Jacobian is solved in the next line. The right hand side of Equation (54) is computed in line 93. The variable output was set in line 75 to $\boldsymbol{\lambda}_{(1)}$, therefore, in line 94 output$^{(1)}$ = $\boldsymbol{\lambda}_{(1)}^{(2)}$ =res1. At the end, the output is incremented in line 95.

The benefit of this approach is two-fold. First, taping of the nonlinear solver is avoided yielding a substantial reduction in memory requirement of the overloading based adjoint. Second, the actual adjoint mappings can be implemented in t2s_a1s_S more efficiently than by interpretation of a corresponding tape. As a general approach the external function feature can/should be applied whenever a similar reduction in memory requirement / computational cost can be expected. Users of `dco/c++` are encouraged to extend the run time library with user-defined intrinsics for (domain-specific) numerical kernels such as, for example, turbulence models in computational fluid dynamics or pay off functions in mathematical finance.

The external function interface of `dco/c++` facilitates a hybrid overloading / source transformation approach to AD. Currently, none of the available source transformation tools covers the entire latest C++ or Fortran standards. Moreover, these tools can be expected to struggle keeping up with the evolution of the programming languages for the foreseeable future. Mature source transformation AD tools such as Tapenade [HP13] can handle (considerable subsets of) C and/or Fortran 95. They can (and should) be applied to suitable selected parts of the given C++XX or Fortran 20XX code. Integration into an enclosing overloading AD solution via the external function interface is typically rather straight forward. The hybrid approach to AD promises further improvements in terms of robustness and computational efficiency.

7 Case Studies

In this section we consider a case study for one dimensional and another case study for two dimensional nonlinear system. We optimize the case studies using a second-order derivative-based method, in which the derivatives are computed with both symbolic and algorithmic tangent and adjoint modes. Furthermore, the run-time overhead as well as the memory requirement of various methods of evaluating second-order directional derivatives are compared.

7.1 One Dimensional Eliptic PDE

As a case study we solve the one dimensional nonlinear differential equation

$$\begin{aligned} \nabla^2(z \cdot u^*) + u^* \cdot \nabla(z \cdot u^*) = 0 \quad &\text{on} \quad \Omega = (0,1) \\ u^* = 10 \ \text{ and } \ z = 1 \quad &\text{for} \ \ x = 0 \\ u^* = 20 \ \text{ and } \ z = 1 \quad &\text{for} \ \ x = 1 \end{aligned}$$

with parameters $z(x)$. For given measurements $u^m(x)$ we state the following parameter fitting problem for z

$$z^* = \arg \min_{z \in I\!R} J(z)$$

with

$$J(z) = \|u(x,z) - u^m(x)\|_2^2 \ . \tag{59}$$

The measurements $u^m(x)$ are generated by a given set of parameters (the "real" parameter distribution $z^*(x)$)[4]. With an equidistant central finite difference discretization we get for a given $\mathbf{u}$ (discretized and, hence, vector-valued variables

[4] Here we do not have the real datas, so we apply a small perturbation to our results and suppose that these are the real ones.

are written as bold letters) the residual function

$$[\mathbf{r}]_i = \frac{1}{h^2} \cdot ([\mathbf{z}]_{i-1}[\mathbf{u}]_{i-1} - 2[\mathbf{z}]_i[\mathbf{u}]_i + [\mathbf{z}]_{i+1}[\mathbf{u}]_{i+1}) \\ +[\mathbf{u}]_i \cdot \frac{1}{2h} \cdot ([\mathbf{z}]_{i+1}[\mathbf{u}]_{i+1} - [\mathbf{z}]_{i-1}[\mathbf{u}]_{i-1})$$

with $h = 1/n$ and n the number of discretization points yielding a system of n nonlinear equations

$$\mathbf{r}(\mathbf{u}, \mathbf{z}) = 0 \quad , \quad \mathbf{u} \in I\!R^n, \mathbf{z} \in I\!R^n \ , \tag{62}$$

which is solved by Newton's method yielding in the i-th Newton iteration the linear system

$$\nabla \mathbf{r}|_{\mathbf{u}=\mathbf{u}_i} \cdot \Delta\mathbf{u} = -\mathbf{r}|_{\mathbf{u}=\mathbf{u}^i} \ .$$

The vector $\mathbf{u}^i$ is updated with the Newton step

$$\mathbf{u}^{i+1} = \mathbf{u}^i + \Delta\mathbf{u} \ .$$

In order to solve the parameter fitting problem, we apply a Newton's method in optimization algorithm which includes a small step size $\alpha > 0$ to the algorithmic objective $J(\mathbf{z})$

$$\mathbf{z}^{k+1} = \mathbf{z}^k - \alpha \cdot \nabla^2 J(\mathbf{z}^k)^{-1} \cdot \nabla J(\mathbf{z}^k) \quad ,$$

where the computation of the Jacobian and Hessian of J at the current iterate $\mathbf{z}^k$ includes the differentiation of the nonlinear solver for $\mathbf{u}^*$, i.e., differentiation of the solver for Equation (62).

According to the Equations (3)–(5), the preprocessor $\boldsymbol{\lambda} = P(\mathbf{z})$ is the identity, while the nonlinear problem is the one dimensional nonlinear function (Equation (60)–(61)) which is solved by a nonlinear solver (e.g. Newton's algorithm) and the postprocessor $p(\mathbf{u})$ computes the cost functional $J(\mathbf{z})$ (Equation (59)).

The goal is to apply Newton's method in optimization algorithm (Equation (63)) in order to minimize the postprocessor J and optimize the initial values $\mathbf{z}_i$ (the inputs) in the preprocessor. For this purpose the computation of the Jacobian and Hessian of J at the current iterate $\mathbf{z}^k$ is required, this includes the evaluation of the Jacobian and Hessian of the postprocessor, nonlinear solver and preprocessor.

In the following the algorithmic FoF (forward over forward) is the evaluation of the first- and second-order directional derivatives with the first- and second-order tangent mode of AD (Section 3.1 and Section 3.3) respectively. The algorithmic FoR (forward over reverse) is the evaluation of the first- and second-order directional derivatives with the first- and second-order adjoint mode of AD (Section 3.2 and Section 3.4) respectively. The symbolic FoF is the evaluation of first- and second-order directional derivatives with symbolic first- and second-order tangent mode as Equation (20) and Equation (19) respectively. The symbolic FoR is the evaluation of first- and second-order directional derivatives with symbolic first- and second-order adjoint mode as Equation (39) and Equation (37) respectively.

A straightforward approach for approximating derivatives involves the use of finite differences. Finite difference approximation (FD) for the derivatives are derived by truncating the Taylor series expansion of the function $g = f(x)$ about a point x. The accuracy of the approximations critically depends on the step size h. More specifically, finite difference approximations for the derivatives can be inaccurate because of truncation or condition errors.

The evaluation of the second-order directional derivatives with finite difference in several variables yields

$$\frac{\partial^2 f}{\partial x \partial y} \approx \frac{f(x+h_1,y+h_2)-f(x+h_1,y-h_2)-f(x-h_1,y+h_2)+f(x-h_1,y-h_2)}{4h_1h_2}.$$

The discrepancy of evaluating the second-order directional derivatives of the postprocessor J with respect to the initial values in preprocessor $\mathbf{z}_i$ with finite difference (Equation 7.1) and algorithmic FoF in the first iteration of the optimization yields

$$Discrepancy = \| \frac{\partial^2 J}{\partial z^2}_{FD} - \frac{\partial^2 J}{\partial z^2}_{Algorithmic} \| \quad .$$

Setting $n = 10$ and the accuracy of the nonlinear solver $\delta = 10^{-6}$, the discrepancy for different step sizes of finite difference $h = h_1 = h_2$ is shown in Table 2.

h	Discrepancy
10^{-1}	16.67
10^{-2}	0.15
10^{-3}	0.001
10^{-4}	0.0001
10^{-5}	0.019
10^{-6}	1.92

Table 2. Discrepancy of the evaluation of second-order directional derivatives with finite difference and algorithmic for different step sizes of finite difference.

The discrepancy of the results will be very small by setting $h = h_1 = h_2 = 10^{-4}$, however this step size is not the smallest one. Therefore finding a suitable step size in finite difference approximation is very crucial.

One should also notice that for the same conditions, i.e., $n = 10$ and $\delta = 10^{-6}$, the discrepancy of the second-order directional derivatives computed by algorithmic and symbolic is proportional to zero (see Figure 6).

The fully algorithmic part of the derivative computation is done by the software tool `dco/c++` [LLN11], implementing AD by overloading in C++. The implementation of symbolic tangent over tangent and tangent over adjoint methods is supported by a custom user interface (see Section 6). For the symbolic part of the derivative computation, the evaluation of the directional derivatives of the preprocessor and the postprocessor will be done by algorithmic mode and only the first- and second-order directional derivatives of the nonlinear solver will be evaluated symbolically.

In the following we compare run time of the optimization of postprocessor (J) with respect to the preprocessor (initial values $\mathbf{z}_i$) by Equation (63) for various differentiated versions of the nonlinear solver. In this paper we set ϵ as the accuracy of the postprocessor ($J(\mathbf{z}) < \epsilon$), which computes the costfunctional (Equation (59)) and δ would be the accuracy of our nonlinear solver ($\mathbf{r}(\mathbf{u}, \mathbf{z}) < \delta$).

	Symbolic		Algorithmic		Finite
n	FoF	FoR	FoF	FoR	Diff.
10	0.1	0.05	0.47	0.09	0.3
20	0.99	0.1	4.25	0.35	2.54
30	4.59	0.28	21.92	1.27	16.18
40	14.58	0.6	73.23	3.4	41.17
50	36.99	1.16	186.87	7.4	-
80	277.55	4.84	1475.36	41.26	-
100	813.47	10.88	-	106.39	-
150	-	44.0	-	560.11	-
200	-	127.69	-	1726.65	-
250	-	307.53	-	-	-

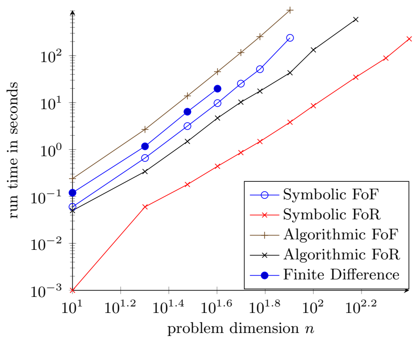


Fig. 4. Run time (in seconds) overhead comparison of optimizing our 1D sample problem with different differentiation methods. The (-) sign implies no convergence after 2000 seconds.

In Fig. 4 we observe the expected behaviour for the computational overhead induced by the various differentiation methods for optimizing the postprocessor J with respect to the initial values in preprocessor $\mathbf{z}$. In this example the accuracy of the nonlinear solver (Newton) for solving the nonlinear system (Equation (62)) is set to $\delta = 1e-6$ and the accuracy of the optimization in postprocessor is set to $\epsilon = 1e-6$. We observe that in both cases of symbolic and algorithmic, the complexity is less if the derivatives be calculated by the adjoint mode. Optimization using symbolic first- and second-order adjoint mode for calculation of the directional derivatives requires less duration, however, the optimization spends the most time by applying algorithmic first- and second-order tangent for evaluating the derivatives. Optimization using finite difference with $h_1 = h_2 = 10^{-5}$ spends less time than algorithmic second-order tangent, but for ($n >= 50$) it does not converge and it is not as accurate as other differentiation methods introduced in this paper.

In our test case, for second-order algorithmic adjoint, the first- and second-order directional derivatives of preprocessing, nonlinear solver and postprocessing are calculated with adjoint AD, this means the storage of some variables in tape and then use the stored variables in tape for backward propagation of derivative values. In this case the checkpoints are not needed. However, for second-order symbolic adjoint, the directional derivatives of preprocessing and postprocessing are calculated by algorithmic adjoint, in which some variables are also stored in tape in order to be used for backward propagation of derivative values. But the directional derivatives of nonlinear solver are computed with symbolic first- and second-order adjoint, in which the required variables for computing the directional derivatives will not be stored in tape, but in checkpoints. The memory requirement of second-order algorithmic adjoint contains the memory occupied by tape, however for symbolic version the memory requirement is the memory occupied by tape and checkpoint.

Memory Requirement ($\nu = 5$)		
n	Symbolic FoR	Algorithmic FoR
10	0.006	0.92
20	0.01	4.38
30	0.02	11.46
40	0.02	23.21
50	0.03	40.71
60	0.03	65.02
80	0.04	138.37
100	0.05	251.8
150	0.08	766.79
200	0.11	1719.2
300	0.16	5470.3
500	0.27	24096.1

Table 3. Memory Requirement for evaluating the second-order adjoint mode of the nonlinear solver (Newton) for our 1D sample problem in MB for different Problem-Dimensions.

Table 3 illustrates the memory requirement for evaluating the second-order symbolic and algorithmic adjoint mode of the nonlinear solver (Newton) for our 1D sample problem with constant number of nonlinear solver's (Newton) iterations ($\nu = 5$) and different problem sizes (n). It shows that, the memory requirement in second-order algorithmic adjoint mode is considerably higher than in the symbolic one, but for both cases it is proportional to the size of the problem.

Table 4 illustrates the memory requirement for evaluating the second-order symbolic and algorithmic adjoint mode of the nonlinear solver (Newton) for our 1D sample problem with constant problem dimension $n = 40$ and different nonlinear solver's (Newton) iterations (ν). In the symbolic mode the differentiation is done symbolically after the computation of the exact solution of the nonlinear system. It means that the memory is just needed to register the solutions of the nonlinear solver (here $\tilde{\mathbf{x}}$) and not the intermediate values of the nonlinear solver's algorithm. Therefore by setting the problem size (n) constant, it remains unchanged by varying the number of nonlinear solver's iterations (ν). However, in the algorithmic mode for calculation of the derivatives we should go through the algorithm (line-by-line differentiation) ν times, so the memory requirement is proportional to the number of iterations.

Memory Requirement ($n = 40$)		
ν	Symbolic FoR	Algorithmic FoR
1	0.02	4.49
2	0.02	9.02
3	0.02	13.75
4	0.02	18.48
5	0.02	23.21
6	0.02	27.94
10	0.02	46.85
15	0.02	70.49
20	0.02	94.14
30	0.02	141.42
50	0.02	236.0
100	0.02	472.43
300	0.02	1418.16
500	0.02	2363.88
1000	0.02	4728.2
2000	0.02	9456.84

Table 4. Memory Requirement for evaluating the second-order adjoint mode of the nonlinear solver (Newton) for our 1D sample problem in MB for different number of Newton's iterations.

7.2 Two Dimensional Eliptic PDE

As a two dimensional case study, we consider the two-dimensional Solid Fuel Ignition problem (also known as the Bratu problem) from the MINPACK-2 test problem collection [ACM91] which is given by the elliptic partial differential equation

$$\Delta x - z \cdot e^x = 0 \quad , \tag{64}$$

where $x = x(q_0, q_1)$ is computed over some bounded domain $\Omega \subsetneq I\!R^2$ with boundary $\Gamma \subsetneq I\!R^2$ and Dirichlet boundary conditions $x(q_0, q_1) = g(q_0, q_1)$ for $(q_0, q_1) \in \Gamma$. For simplicity, we focus on the unit square $\Omega = [0, 1]^2$ and we set

$$g(q_0, q_1) = \begin{cases} 1 & \text{if } q_0 = 1 \\ 0 & \text{otherwise} \end{cases} \quad .$$

We use finite differences as a basic discretization method. Its aim is to replace the differential

$$\Delta x \equiv \frac{\partial^2}{\partial q_0^2} + \frac{\partial^2}{\partial q_1^2}$$

with a set of algebraic equations, thus transforming Equation (64) into a system of nonlinear equations that can be solved by nonlinear solvers.

According to [Nau12] the system of nonlinear equations to be solved is

$$-4 \cdot \mathbf{x}_{i,j} + \mathbf{x}_{i+1,j} + \mathbf{x}_{i-1,j} + \mathbf{x}_{i,j+1} + \mathbf{x}_{i,j-1} = h^2 \cdot \mathbf{z}_{i,j} \cdot e^{\mathbf{x}_{i,j}} \quad , \tag{65}$$

for $i, j = 1, \ldots, n-1$. Discretization of the boundary conditions yields $\mathbf{x}_{n,j} = 1$ and $\mathbf{x}_{i,0} = \mathbf{x}_{0,j} = \mathbf{x}_{i,n} = 0$ for $i, j = 1, \ldots, n-1$.

According to the Equations (3)–(5), the preprocessor is the identity $\boldsymbol{\lambda}_{i,j} = \mathbf{z}_{i,j}$, the solution process is the two dimensional nonlinear function (Equation (65)) which is solved by a nonlinear solvers (e.g. Newton's algorithm). Considering $\mathbf{x}^m_{i,j}$ as real datas[5] the postprocessor computes the cost functional which is

$$J(\mathbf{z}) = \|\mathbf{x}^*(z) - \mathbf{x}^m\|_2^2 \, . \tag{66}$$

The goal is to apply Newton's method in optimization algorithm (Equation (63)) in order to minimize the postprocessor J and optimize the initial values (the inputs) $\mathbf{z}_{i,j}$ in the preprocessor. For this purpose the computation of the Jacobian and Hessian of J at the current iterate $\mathbf{z}^k$ is required, this includes the evaluation of the Jacobian and Hessian of the postprocessor, nonlinear solver and preprocessor.

The fully algorithmic part of the derivative computation is done by the software tool `dco/c++` [LLN11], implementing AD by overloading in C++. The implementation of symbolic tangent over tangent and tangent over adjoint methods is supported by a custom user interface. For the symbolic part of the derivative computation, the evaluation of the directional derivatives of the preprocessor and the postprocessor will be done by algorithmic mode and only the first- and second-order directional derivatives of the nonlinear solver will be evaluated symbolically.

In the following, we set ϵ as the accuracy of the postprocessor ($J(\mathbf{z}) < \epsilon$), which computes the costfunctional (see Equation (66)) and δ would be the accuracy of our nonlinear solver (Newton) to solve the nonlinear system Equation (65).

In this section we compare run time of the optimization of postprocessor (J) with respect to the preprocessor (initial values $\mathbf{z}_{i,j}$) by Equation (63) for various differentiated versions of the nonlinear solver.

In Fig. 5 we observe the expected behaviour for the computational overhead induced by the various differentiation methods for optimizing the postprocessor J with respect to the initial values in preprocessor $\mathbf{z}$. In this example the accuracy of the nonlinear solver for solving the nonlinear problem (e.g. with Newton) is set to $\delta = 1e - 12$ and the accuracy of the optimization is set to $\epsilon = 1e - 2$. It shows that in both cases of symbolic and algorithmic, the complexity is less if the derivatives be calculated by the adjoint mode. Optimization using symbolic first- and second-order adjoint mode for calculation of directional derivatives requires less duration, however, the optimization spends the most time by applying algorithmic first- and second-order tangent for evaluating the derivatives.

[5] Here again we do not have the real datas, so we apply a small perturbation to our results and suppose that these are the real ones.

n	Symbolic		Algorithmic	
	FoF	FoR	FoF	FoR
4 × 4	0.05	0.01	0.17	0.03
6 × 6	0.84	0.05	3.92	0.29
8 × 8	11.32	0.35	64.67	2.54
10 × 10	89.59	1.39	573.51	14.67
11 × 11	218.59	2.62	1454.23	31.95
12 × 12	495.53	4.83	-	74.17
15 × 15	-	25.46	-	485.43
20 × 20	-	277.36	-	-

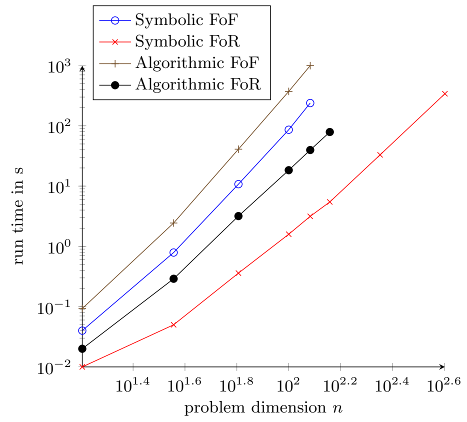


Fig. 5. Run time (in seconds) overhead comparison of optimizing our 2D sample problem with different differentiation methods. The (-) sign implies no convergence after 2000 seconds.

One should also consider that, as mentioned before, the memory requirement of second-order algorithmic contains the memory occupied by tape, however for symbolic version the memory requirement is the memory occupied by tape and checkpoint.

Memory Requirement ($\nu = 8$)		
n	Symbolic FoR	Algorithmic FoR
3 × 3	0.003	0.15
4× 4	0.005	0.83
5× 5	0.007	3.1
6× 6	0.01	9.16
10× 10	0.03	201.21
15× 15	0.06	2438.57
20 × 20	0.11	14401.5
25 × 25	0.17	56988.3
30 × 30	0.24	-

Table 5. Memory Requirement for evaluating the second-order adjoint mode of the nonlinear solver (Newton) for our 2D sample problem in MB for different Problem-Dimensions. The (-) sign implies that the memory is full on a machine with 128 GB RAM.

Table 5 illustrates the memory requirement for evaluating the second-order symbolic and algorithmic adjoint mode of the nonlinear solver (Newton) for our 2D sample problem with constant number of nonlinear solver's (Newton) iterations ($\nu = 8$) and different problem sizes (n) of nonlinear system for adjoint second-order algorithmic and symbolic. It shows that the memory requirement of algorithmic adjoint mode is considerably higher than the symbolic one, but the memory requirement is proportional to the size of the problem for both cases. In the algorithmic mode, the memory was full for $n >= (30 \times 30)$ on a machine (Heisenberg) with 128 GB RAM.

Table 6 illustrates the memory requirement for evaluating the second-order symbolic and algorithmic adjoint mode of the nonlinear solver (Newton) for our

Memory Requirement ($n = 10 \times 10$)		
ν	Symbolic FoR	Algorithmic FoR
1	0.03	22.94
2	0.03	48.41
3	0.03	73.88
4	0.03	99.34
5	0.03	124.81
6	0.03	150.28
8	0.03	201.21
10	0.03	252.14
15	0.03	379.47
20	0.03	506.8
30	0.03	761.47
50	0.03	1270.79
100	0.03	2544.1
200	0.03	5090.73
500	0.03	12730.6
1000	0.03	25463.8

Table 6. Memory Requirement for evaluating the second-order adjoint mode of the nonlinear solver (Newton) for our 2D sample problem in MB for different number of Newton's iterations.

1D sample problem with constant problem dimension $n = 10 \times 10$ and different number of nonlinear solver's (Newton) iterations (ν). In the symbolic mode the differentiation is done symbolically after the computation of the exact solution of the nonlinear system. It means that the memory is just needed to register the solutions of the nonlinear solver (here $\tilde{\mathbf{x}}$) and not the intermediate values of the nonlinear solver's algorithm. Therefore by setting the problem size (n) constant, it remains unchanged by varying the number of nonlinear solver's iterations (ν). However, in the algorithmic mode for calculation of the derivatives we should go through the algorithm (line-by-line differentiation) ν times, so the memory requirement is proportional to the number of iterations.

In the solution process the directional derivatives of the solution are computed by a *symbolic* version of the solver under the assumption that the exact solution $\mathbf{x}^*$ has been reached. $F(\mathbf{x}, \mathbf{z}) = 0$ can be differentiated symbolically in this case. However, for computing the derivatives with algorithmic modes of AD, the whole algorithm would be differentiated. For example, suppose $F(\mathbf{x}, \mathbf{z}) < \delta$. The discrepancies in the results computed by second-order algorithmic and symbolic nonlinear solvers depend on the accuracy (δ) of the approximation of the *primal* solution process. In the following we compare the directional derivatives of the postprocessor J (Equation 66) with respect to the initial values in preprocessor $\mathbf{z}_{i,j}$ computed with symbolic FoF and algorithmic FoF for different nonlinear solver accuracies (δ) in the first iteration of the optimization's algorithm

$$Q = \| \frac{\partial^2 J}{\partial \mathbf{z}^2}_{Symbolic} - \frac{\partial^2 J}{\partial \mathbf{z}^2}_{Algorithmic} \|.$$

As it is shown in the Fig. 6, for $n = (6 \times 6)$ in our 2D test case, by increasing the δ in Newton algorithm (our nonlinear solver) the discrepancies in the results

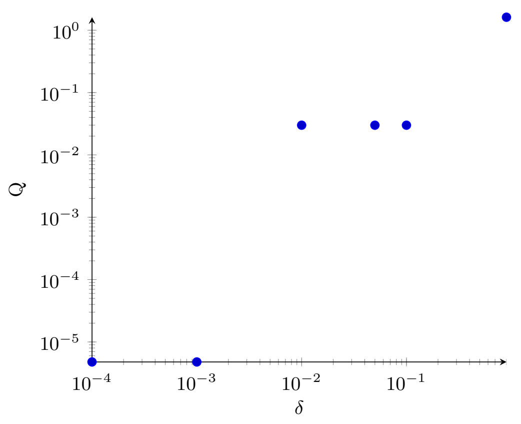


Fig. 6. The Discrepancies in the Derivatives Computed by Second-Order Algorithmic and Symbolic for different Nonlinear Solver Accuracies (δ).

computed by second-order algorithmic and symbolic will be larger. Therefore, in order to have a more exact derivative evaluation in symbolic computation of the derivatives of nonlinear systems, the accuracy of the nonlinear solver (δ) should tend to zero.

8 Conclusion

In this paper we discussed second-order algorithmic and symbolic direct solvers for systems of nonlinear equations. Mathematical insight yields a reduction of the computational overhead for evaluating second-order directional derivatives. Computing derivatives by a fully algorithmic method corresponding to a straight application of AD without taking into account any mathematical or structural properties of the numerical method turns out to be the worst approach in terms of computational efficiency. The performance of the different approaches depends on the number of (Newton) iterations ν and on the problem size n. As an alternative we considered the symbolic second-order differentiation of numerical simulation programs which contain calls to solvers for parameterized systems of n nonlinear equations and compared them with the algorithmic version (AD) of computing second derivatives. In Fig. 4 for 1D- and in Fig. 5 for 2D-eliptic partial differential equation (PDE) we observe that in both cases of second-order symbolic and algorithmic, the complexity of the optimization is less if the derivatives be calculated by the adjoint mode. Optimization using symbolic first- and second-order adjoint mode for evaluation of the directional derivatives requires less time, however, applying algorithmic first- and second-order tangent for the evaluation of the directional derivatives spends the most time.

Directional derivatives of the solution are computed by a symbolic tangent and adjoint version of the solver under the assumption that the exact solution $\mathbf{x}^*$ has been reached. $F(\mathbf{x}, \boldsymbol{\lambda}) = 0$ can be differentiated symbolically in this case, however algorithmic tangent and adjoint versions of the solver compute directional derivatives of the approximation of the solution which are actually computed by the algorithm. This yields the discrepancies in the results computed by algorithmic and symbolic tangent and adjoint. In Fig. 6 it is shown that by increasing the accuracy of the nonlinear solver (δ) the discrepancies in the results computed by second-order algorithmic and symbolic will be larger.

Furthermore, the memory requirement of adjoint second-order symbolic is the memory which is used by tape and checkpoint and it depends on the size of the problem (refer to Table 3 and Table 5 for 1D- and 2D-eliptic PDE respectively), whereas in algorithmic version it is the memory which is used by tape and it depends on the size of the problem and on the number of iterations performed by the nonlinear solver's algorithm (refer to Table 4 and Table 6 for 1D- and 2D-eliptic PDE respectively).

1D Case Study

In the following, the implementation of our 1-dim nonlinear problem, i.e. 1-dim case study in Section 7.1, as well as the nonlinear solver (here, Newton's algorithm) are presented. The complete codes of the second-order symbolic and algorithmic tangent and adjoint modes of the nonlinear solver are illustrated in Section 6. At the end, the implementation of the *main* functions for second-order tangent, adjoint and finite difference of the nonlinear solver are exposed.

Nonlinear Problem

```
/******************** Function.hpp ********************/
template <typename TYPE>
inline void preprocessor(int n, std::vector<TYPE> &z, std::vector<
    TYPE> &lambda)
{
     for(int i=0; i<n; i++)
          lambda[i] = z[i];
}

template <typename TYPE2>
inline TYPE2 F(int n, std::vector<TYPE2> &x, std::vector<TYPE2> &
    lambda, std::vector<TYPE2> &residual) {
     TYPE2 delta = (TYPE2)(1.0 / n);
     TYPE2 norm_res = (TYPE2)(0);
     TYPE2 left, right;

     for (int i = 0; i < n; i++) {
          if (i == 0)
               left = 10;
          else
               left = (TYPE2)lambda[i-1]*x[i-1];
          if (i == n-1)
               right = 20;
          else
               right = (TYPE2)lambda[i+1]*x[i+1];
          residual[i] = (right + left - 2*(TYPE2)lambda[i]*x[i]) /
                         (delta*delta) + x[i]*(right-left)/(2*delta);
          norm_res += pow(residual[i], 2);
     }
     norm_res = sqrt(norm_res);
     return norm_res;
}

template <typename TYPE>
inline TYPE postprocessor(int n, std::vector<TYPE> &x, std::vector<
    double> &x_a) {

     TYPE J = (TYPE)0.;
     for (int i=0; i<n; i++)
          J += pow(x_a[i] - x[i], 2); // x_a are real measurements.
     J /= (TYPE)(n);
     return J;
}
```

Newton's Solver for Nonlinear Equations

```
/******************** Newton.hpp ********************/
```

```
template <class T>
int S(int n, std::vector<T> &x, std::vector<T> &lambda)
{
    std::vector<T> dx(n), r(n), x_orig(n);
    std::vector<T> residual(n), rph(n), rmh(n), xph(n), xmh(n);

    // compute Jacobian of F with respect to x
    vector<vector<T> > Jacobian(n, vector<T>(n));
    T norm_residual;
    int nr_iterations;
    int iter=0;

    for (nr_iterations=0;; nr_iterations++) {
        // compute Jacobian and residual using finite difference
        for (int i=0; i<n; i++)
            x_orig[i]=x[i];
        double hh=1e-8;
        for (int i=0; i<n; i++) {
         for (int j=0; j<n; j++) {
            x[j] = (T)x_orig[j];
            xph[j] = (T)x_orig[j];
            xmh[j] = (T)x_orig[j];}
            F(n,x,lambda,r);
            xph[i] = xph[i] + hh;
            F(n,xph,lambda,rph);
            xmh[i] = xmh[i] - hh;
            F(n,xmh,lambda,rmh);
            for (int j=0; j<n; j++) {
                T deriv = (rph[j]-rmh[j])/(2*hh);
                T derivative = (T) 0.;
                derivative = deriv;
                residual[j]=r[j];
                Jacobian[j][i] = derivative;
            } // j
        } // i
        // solve Newton system by LU-decomposition of Jacobian
        // negate residual
        for(int i=0; i<n; i++)
            residual[i] = -residual[i];
        int **P = LUDecomp(n, Jacobian);
        FBsolve(n, P, Jacobian, residual, dx);
        // update x and evaluate new residual
        for (int i=0; i<n; i++) x[i]+=dx[i];
        norm_residual=F(n,x,lambda,r);
        for (int j=0; j<n; j++) residual[j] = r[j];
        iter++;
        if (norm_residual<=newton_eps) break;
    }
    return iter;
}
```

main Functions:
Second-Order Tangent Mode

```
/******************** main_t2s_t1s.cpp ********************/
#include <iostream>
#include <stdlib.h>
#include <fenv.h>
#include <vector>

```

```
bool Disc;
bool Cont;
const double newton_eps = 1e-6;
const double eps = 1e-6;

#include "dco.hpp"
DCO_DEFINE_GLOBAL_TAPE_POINTER

#include "Function.hpp"
#include "Newton.hpp"

using namespace std;

int main(int argc, char *argv[]) {
    Disc = false;
    Cont = true;
    int n=10;
    if (argc >= 2)
        n = atoi(argv[1]);
    if (argc >= 3)
        Disc = true;
    std::vector<double> xv(n), z(n,1.1), dJ_dz(n,0);
    std::vector<std::vector<double> > H_dJ_dz(n, std::vector<double
        >(n));
    std::vector<dco::t2s_t1s::type> az(n);
    for (int i=0; i<n; i++)
        for (int j=0; j<n; j++)
            H_dJ_dz[i][j] = 0.;
    // Newton Optimization
    int iter = 0;
    double norm_dJ = 10;
    double J;
    double sum = 0;
    while(norm_dJ > eps) {
        ++iter;
        for (int ii=0; ii<n; ii++) {
            for(int jj=0; jj<n; jj++) {
                for (int i = 0; i < n; i++)  az[i] = z[i];
                dco::t2s_t1s::set(az[ii], 1.0, 0, 2);
                dco::t2s_t1s::set(az[jj], 1.0, 1);
                dco::t2s_t1s::type aJ = costfunction(n, az);
                dco::t2s_t1s::get(aJ, J);
                dco::t2s_t1s::get(aJ,dJ_dz[jj], 1);
                dco::t2s_t1s::get(aJ, H_dJ_dz[ii][jj], 1, 2);
                sum += H_dJ_dz[ii][jj]*H_dJ_dz[ii][jj];
            }  // jj

        }  // ii

        Gauss(n, H_dJ_dz, dJ_dz, xv);
        double alpha = 1.;
        double tJ = J+1;
        while (tJ > J) {
            for (int i=0; i<n; i++) {
                double val;
                dco::t2s_t1s::get(az[i], val);
                z[i] = val - alpha*xv[i];
            }
            tJ = costfunction(n, z);
            alpha /= 2.0;
        }   // while (tJ > J)
        J = tJ;
        norm_dJ = 0;
        for (int i=0; i<n; i++) {
            norm_dJ += dJ_dz[i]*dJ_dz[i];
        }
        norm_dJ = sqrt(norm_dJ);
    }  // while(norm_J > eps)

    cout << "--------------------------------------" << endl;
    cout << "J=  " << J << endl;
    cout << "Norm of second derivative is:  " << sqrt(sum) << endl;
    return 0;
}
```

Second-Order Adjoint Mode

```
/******************* main_t2s_a1s.cpp *******************/
#include <iostream>
#include <stdlib.h>
#include <fenv.h>
#include <vector>

bool Disc;
bool Cont;
const double newton_eps = 1e-6;
const double eps = 1e-6;

#include "dco.hpp"
DCO_DEFINE_GLOBAL_TAPE_POINTER

#include "Function.hpp"
#include "Newton.hpp"

using namespace std;

int main(int argc, char *argv[]) {
    Disc = false;
    Cont = true;
    int n=10;
    if (argc >= 2)
        n = atoi(argv[1]);
    if (argc >= 3)
        Disc = true;
    std::vector<double> xv(n), z(n, 1.1), dJ_dz(n, 0);
    std::vector<std::vector<double> > H_dJ_dz(n, std::vector<double>
         (n));
    std::vector<dco::t2s_a1s::type> az(n);
    for (int i=0; i<n; i++)
        for (int j=0; j<n; j++)
            H_dJ_dz[i][j] = 0.;
     // take chunk tape
    dco::t2s_a1s::global_tape = dco::t2s_a1s::tape::create();

    // Newton Optimization
    int iter = 0;
    double norm_dJ = 10;
    double J;
    double sum = 0;
    while(norm_dJ > eps) {
        ++iter;
        for (int ii=0; ii<n; ii++) {
            for (int i=0; i<n; i++)  az[i] = z[i];
            dco::t2s_a1s::set(az[ii], 1.0, 0, 2);
            for (int i=0; i<n; i++)
                dco::t2s_a1s::global_tape->register_variable(az[i]);
            dco::t2s_a1s::tape::iterator position = dco::t2s_a1s::
                global_tape->get_position();
            dco::t2s_a1s::global_tape->zero_adjoints();
            for (int i=0; i<n; i++) dco::t2s_a1s::set(az[i], z[i]);
            dco::t2s_a1s::type aJ = costfunction(n, az);
            double mem_tape = dco::t2s_a1s::global_tape->
                get_tape_memory_size();
            double mem_checkpoint = dco::t2s_a1s::global_tape->
                get_checkpoint_memory_size();
            cout << "mem_tape:␣␣" << mem_tape << endl;
            cout << "mem_checkpoint:␣␣" << mem_checkpoint << endl;
            dco::t2s_a1s::set(aJ, 1., -1);
            dco::t2s_a1s::get(aJ, J);
            dco::t2s_a1s::global_tape->
                interpret_and_reset_adjoint_to(position);
            for (int i=0; i<n; i++) {
                dco::t2s_a1s::get(az[i], dJ_dz[i], -1);
                dco::t2s_a1s::get(az[i], H_dJ_dz[i][ii], -1, 2);
                sum += H_dJ_dz[i][ii]*H_dJ_dz[i][ii];
            }

        } //ii

        Gauss(n, H_dJ_dz, dJ_dz, xv);
        double alpha = 1.;
        double tJ = J+1;
        while (tJ > J) {
            for (int i=0; i<n; i++) {
                double val;
                dco::t2s_a1s::get(az[i], val);
                z[i] = val - alpha*xv[i];
            }
            tJ = costfunction(n, z);
            alpha /= 2.0;
        }   // while (tJ > J)
        J = tJ;
        norm_dJ = 0;
        for (int i=0; i<n; i++) {
            norm_dJ += dJ_dz[i]*dJ_dz[i];
        }
        norm_dJ = sqrt(norm_dJ);
    }  // while(norm_J > eps)
    cout << "------------------------------------------" << endl;
    cout << "J=␣␣" << J << endl;
    cout << "Norm␣of␣second␣derivative␣is:␣␣" << sqrt(sum) << endl;
    return 0;
}
```

Second-Order Finite Difference

```
/******************** main_fd.cpp ************************/
#include <iostream>
#include <stdlib.h>
```

```
#include <fenv.h>
#include <vector>

double passive_time;
double forward_time;

const double newton_eps = 1e-6;
const double eps = 1e-6;

#include "dco.hpp"

#include "Function_FD.hpp"
#include "Newton.hpp"

using namespace std;

int main(int argc, char *argv[]) {

    int n=10;
    if (argc >= 2)
        n = atoi(argv[1]);

    std::vector<double> xv(n), z(n, 1.1), z_fd(n), dJ_dz(n, 0);
    std::vector<std::vector<double> > H_dJ_dz(n, std::vector<double>
         (n));
    for (int i=0; i<n; i++)
        for (int j=0; j<n; j++)
            H_dJ_dz[i][j] = 0.;

    // Newton Optimization
    int iter = 0;
    double norm_dJ = 10;
    double J;
    double sum = 0;

    double h1 = 1e-05;
    double h2 = 1e-05;

    while(norm_dJ > eps) {

        // computing the Jacobian with Finite Difference
        for (int ii=0; ii<n; ii++) {
            for (int i = 0; i < n; i++)
                z_fd[i] = z[i];
            z_fd[ii] += h1;
            double Jph = costfunction(n, z_fd);
            z_fd[ii] -= 2*h1;
            double Jmh = costfunction(n, z_fd);
            dJ_dz[ii] = (Jph-Jmh) / (2*h1);
        }
        // computing the Hessian with Finite Difference
        for (int ii=0; ii<n; ii++) {
            for(int jj=0; jj<n; jj++) {
                for (int i = 0; i < n; i++)
                    z_fd[i] = z[i];
                z_fd[ii] += h1;
                z_fd[jj] += h2;
                double J_1 = costfunction(n, z_fd);
                z_fd[jj] -= 2*h2;
                double J_2 = costfunction(n, z_fd);
                z_fd[ii] -= 2*h1;
                double J_4 = costfunction(n, z_fd);
                z_fd[jj] += 2*h2;
                double J_3 = costfunction(n, z_fd);
                H_dJ_dz[ii][jj] = (J_1 - J_2 - J_3 + J_4) / (4*h1*h2
                    );
                sum += H_dJ_dz[ii][jj]*H_dJ_dz[ii][jj];
            }  // jj
        }  // ii

        J = costfunction(n, z);
        Gauss(n, H_dJ_dz, dJ_dz, xv);
        double alpha = 1.;
        double tJ = J+1;
        while (tJ > J) {
            for (int i=0; i<n; i++) {
                double val = z[i];
                z[i] = val - alpha*xv[i];
            }
            tJ = costfunction(n, z);
            alpha /= 2.0;
        }   // while (tJ > J)

        J = tJ;
        norm_dJ = 0;
        for (int i=0; i<n; i++)
            norm_dJ += dJ_dz[i]*dJ_dz[i];
        norm_dJ = sqrt(norm_dJ);

    }  // while(norm_J > eps)
    cout << "----------------------------------------" << endl;
    cout << "J=  " << J << endl;
    cout << "Norm of second derivative is:  " << sqrt(sum) << endl;
    return 0;
}
```

References

[ACM91] B. Averik, R. Carter, and J. Moré. The Minpack-2 test problem collection (preliminary version). Technical Report 150, Mathematical and Computer Science Division, Argonne National Laboratory, Argonne, IL, 1991.

[Bau74] F. Bauer. Computational graphs and rounding error. *SIAM Journal on Numerical Analysis*, 11:87–96, 1974.

[DS88] N. Dunford and J. T. Schwartz. *Linear Operators, Part 1, General Theory.* Wiley, 1988.

[Gil08] M. B. Giles. Collected matrix derivative results for forward and reverse mode algorithmic differentiation. In C. H. Bischof, H. M. Bücker, P. D. Hovland, U. Naumann, and J. Utke, editors, *Advances in Automatic Differentiation*, volume 64 of *Lecture Notes in Computational Science and Engineering*, pages 35–44. Springer, Berlin, 2008.

[GJU96] A. Griewank, D. Juedes, and J. Utke. ADOL-C, a package for the Automatic Differentiation of algorithms written in C/C++. *ACM Transactions on Mathematical Software*, 22:131–167, 1996.

[GW08] A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation.* Number 105 in Other Titles in Applied Mathematics. SIAM, Philadelphia, PA, 2nd edition, 2008.

[HP13] L. Hascoet and V. Pascual. The Tapenade automatic differentiation tool: principles, model, and specification. *ACM Transactions on Mathematical Software*, 39(3):20, 2013.

[LLN11] K. Leppkes, J. Lotz, and U. Naumann. `dco/c++` – derivative code by overloading in C++. Technical Report AIB-2011-05, RWTH Aachen, June 2011.

[Nau12] U. Naumann. *The Art of Differentiating Computer Programs. An Introduction to Algorithmic differentiation*. Number 24 in Software, Environments, and Tools. SIAM, Philadelphia, PA, 2012.

[NL12] U. Naumann and J. Lotz. Algorithmic differentiation of numerical methods: Tangent-linear and adjoint direct solvers for systems of linear equations. Technical Report AIB-2012-10, LuFG Informatik 12: Software and Tools for Computational Engineering, RWTH Aachen, June 2012. Submitted.

[NLL14] U. Naumann, K. Leppkes, and J. Lotz. `dco/c++` user guide. Technical Report AIB-2014-03, RWTH Aachen University, January 2014.

[NLLT12] U. Naumann, J. Lotz, K. Leppkes, and M. Towara. Algorithmic differentiation of numerical methods: Tangent-linear and adjoint solvers for systems of nonlinear equations. Technical Report AIB-2012-15, Software and Tools for Computational Engineering, RWTH Aachen University LuFG Informatik 12, 2012.

Aachener Informatik-Berichte

This is the list of all technical reports since 1987. To obtain copies of reports please consult

http://aib.informatik.rwth-aachen.de/

or send your request to:

Informatik-Bibliothek, RWTH Aachen, Ahornstr. 55, 52056 Aachen,
Email: `biblio@informatik.rwth-aachen.de`

1987-01 * Fachgruppe Informatik: Jahresbericht 1986
1987-02 * David de Frutos Escrig, Klaus Indermark: Equivalence Relations of Non-Deterministic Ianov-Schemes
1987-03 * Manfred Nagl: A Software Development Environment based on Graph Technology
1987-04 * Claus Lewerentz, Manfred Nagl, Bernhard Westfechtel: On Integration Mechanisms within a Graph-Based Software Development Environment
1987-05 * Reinhard Rinn: Über Eingabeanomalien bei verschiedenen Inferenzmodellen
1987-06 * Werner Damm, Gert Döhmen: Specifying Distributed Computer Architectures in AADL*
1987-07 * Gregor Engels, Claus Lewerentz, Wilhelm Schäfer: Graph Grammar Engineering: A Software Specification Method
1987-08 * Manfred Nagl: Set Theoretic Approaches to Graph Grammars
1987-09 * Claus Lewerentz, Andreas Schürr: Experiences with a Database System for Software Documents
1987-10 * Herbert Klaeren, Klaus Indermark: A New Implementation Technique for Recursive Function Definitions
1987-11 * Rita Loogen: Design of a Parallel Programmable Graph Reduction Machine with Distributed Memory
1987-12 J. Börstler, U. Möncke, R. Wilhelm: Table compression for tree automata
1988-01 * Gabriele Esser, Johannes Rückert, Frank Wagner Gesellschaftliche Aspekte der Informatik
1988-02 * Peter Martini, Otto Spaniol: Token-Passing in High-Speed Backbone Networks for Campus-Wide Environments
1988-03 * Thomas Welzel: Simulation of a Multiple Token Ring Backbone
1988-04 * Peter Martini: Performance Comparison for HSLAN Media Access Protocols
1988-05 * Peter Martini: Performance Analysis of Multiple Token Rings
1988-06 * Andreas Mann, Johannes Rückert, Otto Spaniol: Datenfunknetze
1988-07 * Andreas Mann, Johannes Rückert: Packet Radio Networks for Data Exchange
1988-08 * Andreas Mann, Johannes Rückert: Concurrent Slot Assignment Protocol for Packet Radio Networks
1988-09 * W. Kremer, F. Reichert, J. Rückert, A. Mann: Entwurf einer Netzwerktopologie für ein Mobilfunknetz zur Unterstützung des öffentlichen Straßenverkehrs

1988-10 *	Kai Jakobs: Towards User-Friendly Networking
1988-11 *	Kai Jakobs: The Directory - Evolution of a Standard
1988-12 *	Kai Jakobs: Directory Services in Distributed Systems - A Survey
1988-13 *	Martine Schümmer: RS-511, a Protocol for the Plant Floor
1988-14 *	U. Quernheim: Satellite Communication Protocols - A Performance Comparison Considering On-Board Processing
1988-15 *	Peter Martini, Otto Spaniol, Thomas Welzel: File Transfer in High Speed Token Ring Networks: Performance Evaluation by Approximate Analysis and Simulation
1988-16 *	Fachgruppe Informatik: Jahresbericht 1987
1988-17 *	Wolfgang Thomas: Automata on Infinite Objects
1988-18 *	Michael Sonnenschein: On Petri Nets and Data Flow Graphs
1988-19 *	Heiko Vogler: Functional Distribution of the Contextual Analysis in Block-Structured Programming Languages: A Case Study of Tree Transducers
1988-20 *	Thomas Welzel: Einsatz des Simulationswerkzeuges QNAP2 zur Leistungsbewertung von Kommunikationsprotokollen
1988-21 *	Th. Janning, C. Lewerentz: Integrated Project Team Management in a Software Development Environment
1988-22 *	Joost Engelfriet, Heiko Vogler: Modular Tree Transducers
1988-23 *	Wolfgang Thomas: Automata and Quantifier Hierarchies
1988-24 *	Uschi Heuter: Generalized Definite Tree Languages
1989-01 *	Fachgruppe Informatik: Jahresbericht 1988
1989-02 *	G. Esser, J. Rückert, F. Wagner (Hrsg.): Gesellschaftliche Aspekte der Informatik
1989-03 *	Heiko Vogler: Bottom-Up Computation of Primitive Recursive Tree Functions
1989-04 *	Andy Schürr: Introduction to PROGRESS, an Attribute Graph Grammar Based Specification Language
1989-05	J. Börstler: Reuse and Software Development - Problems, Solutions, and Bibliography (in German)
1989-06 *	Kai Jakobs: OSI - An Appropriate Basis for Group Communication?
1989-07 *	Kai Jakobs: ISO's Directory Proposal - Evolution, Current Status and Future Problems
1989-08 *	Bernhard Westfechtel: Extension of a Graph Storage for Software Documents with Primitives for Undo/Redo and Revision Control
1989-09 *	Peter Martini: High Speed Local Area Networks - A Tutorial
1989-10 *	P. Davids, Th. Welzel: Performance Analysis of DQDB Based on Simulation
1989-11 *	Manfred Nagl (Ed.): Abstracts of Talks presented at the WG '89 15th International Workshop on Graphtheoretic Concepts in Computer Science
1989-12 *	Peter Martini: The DQDB Protocol - Is it Playing the Game?
1989-13 *	Martine Schümmer: CNC/DNC Communication with MAP
1989-14 *	Martine Schümmer: Local Area Networks for Manufactoring Environments with hard Real-Time Requirements

1989-15 *	M. Schümmer, Th. Welzel, P. Martini: Integration of Field Bus and MAP Networks - Hierarchical Communication Systems in Production Environments
1989-16 *	G. Vossen, K.-U. Witt: SUXESS: Towards a Sound Unification of Extensions of the Relational Data Model
1989-17 *	J. Derissen, P. Hruschka, M.v.d. Beeck, Th. Janning, M. Nagl: Integrating Structured Analysis and Information Modelling
1989-18	A. Maassen: Programming with Higher Order Functions
1989-19 *	Mario Rodriguez-Artalejo, Heiko Vogler: A Narrowing Machine for Syntax Directed BABEL
1989-20	H. Kuchen, R. Loogen, J.J. Moreno Navarro, M. Rodriguez Artalejo: Graph-based Implementation of a Functional Logic Language
1990-01 *	Fachgruppe Informatik: Jahresbericht 1989
1990-02 *	Vera Jansen, Andreas Potthoff, Wolfgang Thomas, Udo Wermuth: A Short Guide to the AMORE System (Computing Automata, MOnoids and Regular Expressions)
1990-03 *	Jerzy Skurczynski: On Three Hierarchies of Weak SkS Formulas
1990-04	R. Loogen: Stack-based Implementation of Narrowing
1990-05	H. Kuchen, A. Wagener: Comparison of Dynamic Load Balancing Strategies
1990-06 *	Kai Jakobs, Frank Reichert: Directory Services for Mobile Communication
1990-07 *	Kai Jakobs: What's Beyond the Interface - OSI Networks to Support Cooperative Work
1990-08 *	Kai Jakobs: Directory Names and Schema - An Evaluation
1990-09 *	Ulrich Quernheim, Dieter Kreuer: Das CCITT - Signalisierungssystem Nr. 7 auf Satellitenstrecken; Simulation der Zeichengabestrecke
1990-11	H. Kuchen, R. Loogen, J.J. Moreno Navarro, M. Rodriguez Artalejo: Lazy Narrowing in a Graph Machine
1990-12 *	Kai Jakobs, Josef Kaltwasser, Frank Reichert, Otto Spaniol: Der Computer fährt mit
1990-13 *	Rudolf Mathar, Andreas Mann: Analyzing a Distributed Slot Assignment Protocol by Markov Chains
1990-14	A. Maassen: Compilerentwicklung in Miranda - ein Praktikum in funktionaler Programmierung (written in german)
1990-15 *	Manfred Nagl, Andreas Schürr: A Specification Environment for Graph Grammars
1990-16	A. Schürr: PROGRESS: A VHL-Language Based on Graph Grammars
1990-17 *	Marita Möller: Ein Ebenenmodell wissensbasierter Konsultationen - Unterstützung für Wissensakquisition und Erklärungsfähigkeit
1990-18 *	Eric Kowalewski: Entwurf und Interpretation einer Sprache zur Beschreibung von Konsultationsphasen in Expertensystemen
1990-20	Y. Ortega Mallen, D. de Frutos Escrig: A Complete Proof System for Timed Observations
1990-21 *	Manfred Nagl: Modelling of Software Architectures: Importance, Notions, Experiences
1990-22	H. Fassbender, H. Vogler: A Call-by-need Implementation of Syntax Directed Functional Programming

1991-01	Guenther Geiler (ed.), Fachgruppe Informatik: Jahresbericht 1990
1991-03	B. Steffen, A. Ingolfsdottir: Characteristic Formulae for Processes with Divergence
1991-04	M. Portz: A new class of cryptosystems based on interconnection networks
1991-05	H. Kuchen, G. Geiler: Distributed Applicative Arrays
1991-06 *	Ludwig Staiger: Kolmogorov Complexity and Hausdorff Dimension
1991-07 *	Ludwig Staiger: Syntactic Congruences for w-languages
1991-09 *	Eila Kuikka: A Proposal for a Syntax-Directed Text Processing System
1991-10	K. Gladitz, H. Fassbender, H. Vogler: Compiler-based Implementation of Syntax-Directed Functional Programming
1991-11	R. Loogen, St. Winkler: Dynamic Detection of Determinism in Functional Logic Languages
1991-12 *	K. Indermark, M. Rodriguez Artalejo (Eds.): Granada Workshop on the Integration of Functional and Logic Programming
1991-13 *	Rolf Hager, Wolfgang Kremer: The Adaptive Priority Scheduler: A More Fair Priority Service Discipline
1991-14 *	Andreas Fasbender, Wolfgang Kremer: A New Approximation Algorithm for Tandem Networks with Priority Nodes
1991-15	J. Börstler, A. Zündorf: Revisiting extensions to Modula-2 to support reusability
1991-16	J. Börstler, Th. Janning: Bridging the gap between Requirements Analysis and Design
1991-17	A. Zündorf, A. Schürr: Nondeterministic Control Structures for Graph Rewriting Systems
1991-18 *	Matthias Jarke, John Mylopoulos, Joachim W. Schmidt, Yannis Vassiliou: DAIDA: An Environment for Evolving Information Systems
1991-19	M. Jeusfeld, M. Jarke: From Relational to Object-Oriented Integrity Simplification
1991-20	G. Hogen, A. Kindler, R. Loogen: Automatic Parallelization of Lazy Functional Programs
1991-21 *	Prof. Dr. rer. nat. Otto Spaniol: ODP (Open Distributed Processing): Yet another Viewpoint
1991-22	H. Kuchen, F. Lücking, H. Stoltze: The Topology Description Language TDL
1991-23	S. Graf, B. Steffen: Compositional Minimization of Finite State Systems
1991-24	R. Cleaveland, J. Parrow, B. Steffen: The Concurrency Workbench: A Semantics Based Tool for the Verification of Concurrent Systems
1991-25 *	Rudolf Mathar, Jürgen Mattfeldt: Optimal Transmission Ranges for Mobile Communication in Linear Multihop Packet Radio Networks
1991-26	M. Jeusfeld, M. Staudt: Query Optimization in Deductive Object Bases
1991-27	J. Knoop, B. Steffen: The Interprocedural Coincidence Theorem
1991-28	J. Knoop, B. Steffen: Unifying Strength Reduction and Semantic Code Motion
1991-30	T. Margaria: First-Order theories for the verification of complex FSMs
1991-31	B. Steffen: Generating Data Flow Analysis Algorithms from Modal Specifications
1992-01	Stefan Eherer (ed.), Fachgruppe Informatik: Jahresbericht 1991

1992-02 *	Bernhard Westfechtel: Basismechanismen zur Datenverwaltung in strukturbezogenen Hypertextsystemen
1992-04	S. A. Smolka, B. Steffen: Priority as Extremal Probability
1992-05 *	Matthias Jarke, Carlos Maltzahn, Thomas Rose: Sharing Processes: Team Coordination in Design Repositories
1992-06	O. Burkart, B. Steffen: Model Checking for Context-Free Processes
1992-07 *	Matthias Jarke, Klaus Pohl: Information Systems Quality and Quality Information Systems
1992-08 *	Rudolf Mathar, Jürgen Mattfeldt: Analyzing Routing Strategy NFP in Multihop Packet Radio Networks on a Line
1992-09 *	Alfons Kemper, Guido Moerkotte: Grundlagen objektorientierter Datenbanksysteme
1992-10	Matthias Jarke, Manfred Jeusfeld, Andreas Miethsam, Michael Gocek: Towards a logic-based reconstruction of software configuration management
1992-11	Werner Hans: A Complete Indexing Scheme for WAM-based Abstract Machines
1992-12	W. Hans, R. Loogen, St. Winkler: On the Interaction of Lazy Evaluation and Backtracking
1992-13 *	Matthias Jarke, Thomas Rose: Specification Management with CAD
1992-14	Th. Noll, H. Vogler: Top-down Parsing with Simultaneous Evaluation on Noncircular Attribute Grammars
1992-15	A. Schuerr, B. Westfechtel: Graphgrammatiken und Graphersetzungssysteme(written in german)
1992-16 *	Graduiertenkolleg Informatik und Technik (Hrsg.): Forschungsprojekte des Graduiertenkollegs Informatik und Technik
1992-17	M. Jarke (ed.): ConceptBase V3.1 User Manual
1992-18 *	Clarence A. Ellis, Matthias Jarke (Eds.): Distributed Cooperation in Integrated Information Systems - Proceedings of the Third International Workshop on Intelligent and Cooperative Information Systems
1992-19-00	H. Kuchen, R. Loogen (eds.): Proceedings of the 4th Int. Workshop on the Parallel Implementation of Functional Languages
1992-19-01	G. Hogen, R. Loogen: PASTEL - A Parallel Stack-Based Implementation of Eager Functional Programs with Lazy Data Structures (Extended Abstract)
1992-19-02	H. Kuchen, K. Gladitz: Implementing Bags on a Shared Memory MIMD-Machine
1992-19-03	C. Rathsack, S.B. Scholz: LISA - A Lazy Interpreter for a Full-Fledged Lambda-Calculus
1992-19-04	T.A. Bratvold: Determining Useful Parallelism in Higher Order Functions
1992-19-05	S. Kahrs: Polymorphic Type Checking by Interpretation of Code
1992-19-06	M. Chakravarty, M. Köhler: Equational Constraints, Residuation, and the Parallel JUMP-Machine
1992-19-07	J. Seward: Polymorphic Strictness Analysis using Frontiers (Draft Version)
1992-19-08	D. Gärtner, A. Kimms, W. Kluge: pi-Redˆ+ - A Compiling Graph-Reduction System for a Full Fledged Lambda-Calculus

1992-19-09 D. Howe, G. Burn: Experiments with strict STG code
1992-19-10 J. Glauert: Parallel Implementation of Functional Languages Using Small Processes
1992-19-11 M. Joy, T. Axford: A Parallel Graph Reduction Machine
1992-19-12 A. Bennett, P. Kelly: Simulation of Multicache Parallel Reduction
1992-19-13 K. Langendoen, D.J. Agterkamp: Cache Behaviour of Lazy Functional Programs (Working Paper)
1992-19-14 K. Hammond, S. Peyton Jones: Profiling scheduling strategies on the GRIP parallel reducer
1992-19-15 S. Mintchev: Using Strictness Information in the STG-machine
1992-19-16 D. Rushall: An Attribute Grammar Evaluator in Haskell
1992-19-17 J. Wild, H. Glaser, P. Hartel: Statistics on storage management in a lazy functional language implementation
1992-19-18 W.S. Martins: Parallel Implementations of Functional Languages
1992-19-19 D. Lester: Distributed Garbage Collection of Cyclic Structures (Draft version)
1992-19-20 J.C. Glas, R.F.H. Hofman, W.G. Vree: Parallelization of Branch-and-Bound Algorithms in a Functional Programming Environment
1992-19-21 S. Hwang, D. Rushall: The nu-STG machine: a parallelized Spineless Tagless Graph Reduction Machine in a distributed memory architecture (Draft version)
1992-19-22 G. Burn, D. Le Metayer: Cps-Translation and the Correctness of Optimising Compilers
1992-19-23 S.L. Peyton Jones, P. Wadler: Imperative functional programming (Brief summary)
1992-19-24 W. Damm, F. Liu, Th. Peikenkamp: Evaluation and Parallelization of Functions in Functional + Logic Languages (abstract)
1992-19-25 M. Kesseler: Communication Issues Regarding Parallel Functional Graph Rewriting
1992-19-26 Th. Peikenkamp: Charakterizing and representing neededness in functional loginc languages (abstract)
1992-19-27 H. Doerr: Monitoring with Graph-Grammars as formal operational Models
1992-19-28 J. van Groningen: Some implementation aspects of Concurrent Clean on distributed memory architectures
1992-19-29 G. Ostheimer: Load Bounding for Implicit Parallelism (abstract)
1992-20 H. Kuchen, F.J. Lopez Fraguas, J.J. Moreno Navarro, M. Rodriguez Artalejo: Implementing Disequality in a Lazy Functional Logic Language
1992-21 H. Kuchen, F.J. Lopez Fraguas: Result Directed Computing in a Functional Logic Language
1992-22 H. Kuchen, J.J. Moreno Navarro, M.V. Hermenegildo: Independent AND-Parallel Narrowing
1992-23 T. Margaria, B. Steffen: Distinguishing Formulas for Free
1992-24 K. Pohl: The Three Dimensions of Requirements Engineering
1992-25 * R. Stainov: A Dynamic Configuration Facility for Multimedia Communications
1992-26 * Michael von der Beeck: Integration of Structured Analysis and Timed Statecharts for Real-Time and Concurrency Specification

1992-27	W. Hans, St. Winkler: Aliasing and Groundness Analysis of Logic Programs through Abstract Interpretation and its Safety
1992-28 *	Gerhard Steinke, Matthias Jarke: Support for Security Modeling in Information Systems Design
1992-29	B. Schinzel: Warum Frauenforschung in Naturwissenschaft und Technik
1992-30	A. Kemper, G. Moerkotte, K. Peithner: Object-Orientation Axiomatised by Dynamic Logic
1992-32 *	Bernd Heinrichs, Kai Jakobs: Timer Handling in High-Performance Transport Systems
1992-33 *	B. Heinrichs, K. Jakobs, K. Lenßen, W. Reinhardt, A. Spinner: Euro-Bridge: Communication Services for Multimedia Applications
1992-34	C. Gerlhof, A. Kemper, Ch. Kilger, G. Moerkotte: Partition-Based Clustering in Object Bases: From Theory to Practice
1992-35	J. Börstler: Feature-Oriented Classification and Reuse in IPSEN
1992-36	M. Jarke, J. Bubenko, C. Rolland, A. Sutcliffe, Y. Vassiliou: Theories Underlying Requirements Engineering: An Overview of NATURE at Genesis
1992-37 *	K. Pohl, M. Jarke: Quality Information Systems: Repository Support for Evolving Process Models
1992-38	A. Zuendorf: Implementation of the imperative / rule based language PROGRES
1992-39	P. Koch: Intelligentes Backtracking bei der Auswertung funktional-logischer Programme
1992-40 *	Rudolf Mathar, Jürgen Mattfeldt: Channel Assignment in Cellular Radio Networks
1992-41 *	Gerhard Friedrich, Wolfgang Neidl: Constructive Utility in Model-Based Diagnosis Repair Systems
1992-42 *	P. S. Chen, R. Hennicker, M. Jarke: On the Retrieval of Reusable Software Components
1992-43	W. Hans, St.Winkler: Abstract Interpretation of Functional Logic Languages
1992-44	N. Kiesel, A. Schuerr, B. Westfechtel: Design and Evaluation of GRAS, a Graph-Oriented Database System for Engineering Applications
1993-01 *	Fachgruppe Informatik: Jahresbericht 1992
1993-02 *	Patrick Shicheng Chen: On Inference Rules of Logic-Based Information Retrieval Systems
1993-03	G. Hogen, R. Loogen: A New Stack Technique for the Management of Runtime Structures in Distributed Environments
1993-05	A. Zündorf: A Heuristic for the Subgraph Isomorphism Problem in Executing PROGRES
1993-06	A. Kemper, D. Kossmann: Adaptable Pointer Swizzling Strategies in Object Bases: Design, Realization, and Quantitative Analysis
1993-07 *	Graduiertenkolleg Informatik und Technik (Hrsg.): Graduiertenkolleg Informatik und Technik
1993-08 *	Matthias Berger: k-Coloring Vertices using a Neural Network with Convergence to Valid Solutions
1993-09	M. Buchheit, M. Jeusfeld, W. Nutt, M. Staudt: Subsumption between Queries to Object-Oriented Databases

1993-10	O. Burkart, B. Steffen: Pushdown Processes: Parallel Composition and Model Checking
1993-11 *	R. Große-Wienker, O. Hermanns, D. Menzenbach, A. Pollacks, S. Repetzki, J. Schwartz, K. Sonnenschein, B. Westfechtel: Das SUKITS-Projekt: A-posteriori-Integration heterogener CIM-Anwendungssysteme
1993-12 *	Rudolf Mathar, Jürgen Mattfeldt: On the Distribution of Cumulated Interference Power in Rayleigh Fading Channels
1993-13	O. Maler, L. Staiger: On Syntactic Congruences for omega-languages
1993-14	M. Jarke, St. Eherer, R. Gallersdoerfer, M. Jeusfeld, M. Staudt: ConceptBase - A Deductive Object Base Manager
1993-15	M. Staudt, H.W. Nissen, M.A. Jeusfeld: Query by Class, Rule and Concept
1993-16 *	M. Jarke, K. Pohl, St. Jacobs et al.: Requirements Engineering: An Integrated View of Representation Process and Domain
1993-17 *	M. Jarke, K. Pohl: Establishing Vision in Context: Towards a Model of Requirements Processes
1993-18	W. Hans, H. Kuchen, St. Winkler: Full Indexing for Lazy Narrowing
1993-19	W. Hans, J.J. Ruz, F. Saenz, St. Winkler: A VHDL Specification of a Shared Memory Parallel Machine for Babel
1993-20 *	K. Finke, M. Jarke, P. Szczurko, R. Soltysiak: Quality Management for Expert Systems in Process Control
1993-21	M. Jarke, M.A. Jeusfeld, P. Szczurko: Three Aspects of Intelligent Cooperation in the Quality Cycle
1994-01	Margit Generet, Sven Martin (eds.), Fachgruppe Informatik: Jahresbericht 1993
1994-02	M. Lefering: Development of Incremental Integration Tools Using Formal Specifications
1994-03 *	P. Constantopoulos, M. Jarke, J. Mylopoulos, Y. Vassiliou: The Software Information Base: A Server for Reuse
1994-04 *	Rolf Hager, Rudolf Mathar, Jürgen Mattfeldt: Intelligent Cruise Control and Reliable Communication of Mobile Stations
1994-05 *	Rolf Hager, Peter Hermesmann, Michael Portz: Feasibility of Authentication Procedures within Advanced Transport Telematics
1994-06 *	Claudia Popien, Bernd Meyer, Axel Kuepper: A Formal Approach to Service Import in ODP Trader Federations
1994-07	P. Peters, P. Szczurko: Integrating Models of Quality Management Methods by an Object-Oriented Repository
1994-08 *	Manfred Nagl, Bernhard Westfechtel: A Universal Component for the Administration in Distributed and Integrated Development Environments
1994-09 *	Patrick Horster, Holger Petersen: Signatur- und Authentifikationsverfahren auf der Basis des diskreten Logarithmusproblems
1994-11	A. Schürr: PROGRES, A Visual Language and Environment for PROgramming with Graph REwrite Systems
1994-12	A. Schürr: Specification of Graph Translators with Triple Graph Grammars
1994-13	A. Schürr: Logic Based Programmed Structure Rewriting Systems
1994-14	L. Staiger: Codes, Simplifying Words, and Open Set Condition

1994-15 * Bernhard Westfechtel: A Graph-Based System for Managing Configurations of Engineering Design Documents

1994-16 P. Klein: Designing Software with Modula-3

1994-17 I. Litovsky, L. Staiger: Finite acceptance of infinite words

1994-18 G. Hogen, R. Loogen: Parallel Functional Implementations: Graphbased vs. Stackbased Reduction

1994-19 M. Jeusfeld, U. Johnen: An Executable Meta Model for Re-Engineering of Database Schemas

1994-20 * R. Gallersdörfer, M. Jarke, K. Klabunde: Intelligent Networks as a Data Intensive Application (INDIA)

1994-21 M. Mohnen: Proving the Correctness of the Static Link Technique Using Evolving Algebras

1994-22 H. Fernau, L. Staiger: Valuations and Unambiguity of Languages, with Applications to Fractal Geometry

1994-24 * M. Jarke, K. Pohl, R. Dömges, St. Jacobs, H. W. Nissen: Requirements Information Management: The NATURE Approach

1994-25 * M. Jarke, K. Pohl, C. Rolland, J.-R. Schmitt: Experience-Based Method Evaluation and Improvement: A Process Modeling Approach

1994-26 * St. Jacobs, St. Kethers: Improving Communication and Decision Making within Quality Function Deployment

1994-27 * M. Jarke, H. W. Nissen, K. Pohl: Tool Integration in Evolving Information Systems Environments

1994-28 O. Burkart, D. Caucal, B. Steffen: An Elementary Bisimulation Decision Procedure for Arbitrary Context-Free Processes

1995-01 * Fachgruppe Informatik: Jahresbericht 1994

1995-02 Andy Schürr, Andreas J. Winter, Albert Zündorf: Graph Grammar Engineering with PROGRES

1995-03 Ludwig Staiger: A Tight Upper Bound on Kolmogorov Complexity by Hausdorff Dimension and Uniformly Optimal Prediction

1995-04 Birgitta König-Ries, Sven Helmer, Guido Moerkotte: An experimental study on the complexity of left-deep join ordering problems for cyclic queries

1995-05 Sophie Cluet, Guido Moerkotte: Efficient Evaluation of Aggregates on Bulk Types

1995-06 Sophie Cluet, Guido Moerkotte: Nested Queries in Object Bases

1995-07 Sophie Cluet, Guido Moerkotte: Query Optimization Techniques Exploiting Class Hierarchies

1995-08 Markus Mohnen: Efficient Compile-Time Garbage Collection for Arbitrary Data Structures

1995-09 Markus Mohnen: Functional Specification of Imperative Programs: An Alternative Point of View of Functional Languages

1995-10 Rainer Gallersdörfer, Matthias Nicola: Improving Performance in Replicated Databases through Relaxed Coherency

1995-11 * M.Staudt, K.von Thadden: Subsumption Checking in Knowledge Bases

1995-12 * G.V.Zemanek, H.W.Nissen, H.Hubert, M.Jarke: Requirements Analysis from Multiple Perspectives: Experiences with Conceptual Modeling Technology

1995-13 *	M.Staudt, M.Jarke: Incremental Maintenance of Externally Materialized Views
1995-14 *	P.Peters, P.Szczurko, M.Jeusfeld: Oriented Information Management: Conceptual Models at Work
1995-15 *	Matthias Jarke, Sudha Ram (Hrsg.): WITS 95 Proceedings of the 5th Annual Workshop on Information Technologies and Systems
1995-16 *	W.Hans, St.Winkler, F.Saenz: Distributed Execution in Functional Logic Programming
1996-01 *	Jahresbericht 1995
1996-02	Michael Hanus, Christian Prehofer: Higher-Order Narrowing with Definitional Trees
1996-03 *	W.Scheufele, G.Moerkotte: Optimal Ordering of Selections and Joins in Acyclic Queries with Expensive Predicates
1996-04	Klaus Pohl: PRO-ART: Enabling Requirements Pre-Traceability
1996-05	Klaus Pohl: Requirements Engineering: An Overview
1996-06 *	M.Jarke, W.Marquardt: Design and Evaluation of Computer–Aided Process Modelling Tools
1996-07	Olaf Chitil: The Sigma-Semantics: A Comprehensive Semantics for Functional Programs
1996-08 *	S.Sripada: On Entropy and the Limitations of the Second Law of Thermodynamics
1996-09	Michael Hanus (Ed.): Proceedings of the Poster Session of ALP96 - Fifth International Conference on Algebraic and Logic Programming
1996-09-0	Michael Hanus (Ed.): Proceedings of the Poster Session of ALP 96 - Fifth International Conference on Algebraic and Logic Programming: Introduction and table of contents
1996-09-1	Ilies Alouini: An Implementation of Conditional Concurrent Rewriting on Distributed Memory Machines
1996-09-2	Olivier Danvy, Karoline Malmkjær: On the Idempotence of the CPS Transformation
1996-09-3	Víctor M. Gulías, José L. Freire: Concurrent Programming in Haskell
1996-09-4	Sébastien Limet, Pierre Réty: On Decidability of Unifiability Modulo Rewrite Systems
1996-09-5	Alexandre Tessier: Declarative Debugging in Constraint Logic Programming
1996-10	Reidar Conradi, Bernhard Westfechtel: Version Models for Software Configuration Management
1996-11 *	C.Weise, D.Lenzkes: A Fast Decision Algorithm for Timed Refinement
1996-12 *	R.Dömges, K.Pohl, M.Jarke, B.Lohmann, W.Marquardt: PRO-ART/CE* — An Environment for Managing the Evolution of Chemical Process Simulation Models
1996-13 *	K.Pohl, R.Klamma, K.Weidenhaupt, R.Dömges, P.Haumer, M.Jarke: A Framework for Process-Integrated Tools
1996-14 *	R.Gallersdörfer, K.Klabunde, A.Stolz, M.Eßmajor: INDIA — Intelligent Networks as a Data Intensive Application, Final Project Report, June 1996
1996-15 *	H.Schimpe, M.Staudt: VAREX: An Environment for Validating and Refining Rule Bases

1996-16 *	M.Jarke, M.Gebhardt, S.Jacobs, H.Nissen: Conflict Analysis Across Heterogeneous Viewpoints: Formalization and Visualization
1996-17	Manfred A. Jeusfeld, Tung X. Bui: Decision Support Components on the Internet
1996-18	Manfred A. Jeusfeld, Mike Papazoglou: Information Brokering: Design, Search and Transformation
1996-19 *	P.Peters, M.Jarke: Simulating the impact of information flows in networked organizations
1996-20	Matthias Jarke, Peter Peters, Manfred A. Jeusfeld: Model-driven planning and design of cooperative information systems
1996-21 *	G.de Michelis, E.Dubois, M.Jarke, F.Matthes, J.Mylopoulos, K.Pohl, J.Schmidt, C.Woo, E.Yu: Cooperative information systems: a manifesto
1996-22 *	S.Jacobs, M.Gebhardt, S.Kethers, W.Rzasa: Filling HTML forms simultaneously: CoWeb architecture and functionality
1996-23 *	M.Gebhardt, S.Jacobs: Conflict Management in Design
1997-01	Michael Hanus, Frank Zartmann (eds.): Jahresbericht 1996
1997-02	Johannes Faassen: Using full parallel Boltzmann Machines for Optimization
1997-03	Andreas Winter, Andy Schürr: Modules and Updatable Graph Views for PROgrammed Graph REwriting Systems
1997-04	Markus Mohnen, Stefan Tobies: Implementing Context Patterns in the Glasgow Haskell Compiler
1997-05 *	S.Gruner: Schemakorrespondenzaxiome unterstützen die paargrammatische Spezifikation inkrementeller Integrationswerkzeuge
1997-06	Matthias Nicola, Matthias Jarke: Design and Evaluation of Wireless Health Care Information Systems in Developing Countries
1997-07	Petra Hofstedt: Taskparallele Skelette für irregulär strukturierte Probleme in deklarativen Sprachen
1997-08	Dorothea Blostein, Andy Schürr: Computing with Graphs and Graph Rewriting
1997-09	Carl-Arndt Krapp, Bernhard Westfechtel: Feedback Handling in Dynamic Task Nets
1997-10	Matthias Nicola, Matthias Jarke: Integrating Replication and Communication in Performance Models of Distributed Databases
1997-11 *	R. Klamma, P. Peters, M. Jarke: Workflow Support for Failure Management in Federated Organizations
1997-13	Markus Mohnen: Optimising the Memory Management of Higher-Order Functional Programs
1997-14	Roland Baumann: Client/Server Distribution in a Structure-Oriented Database Management System
1997-15	George Botorog: High-Level Parallel Programming and the Efficient Implementation of Numerical Algorithms
1998-01 *	Fachgruppe Informatik: Jahresbericht 1997
1998-02	Stefan Gruner, Manfred Nagel, Andy Schürr: Fine-grained and Structure-Oriented Document Integration Tools are Needed for Development Processes

1998-03	Stefan Gruner: Einige Anmerkungen zur graphgrammatischen Spezifikation von Integrationswerkzeugen nach Westfechtel, Janning, Lefering und Schürr
1998-04 *	O. Kubitz: Mobile Robots in Dynamic Environments
1998-05	Martin Leucker, Stephan Tobies: Truth - A Verification Platform for Distributed Systems
1998-06 *	Matthias Oliver Berger: DECT in the Factory of the Future
1998-07	M. Arnold, M. Erdmann, M. Glinz, P. Haumer, R. Knoll, B. Paech, K. Pohl, J. Ryser, R. Studer, K. Weidenhaupt: Survey on the Scenario Use in Twelve Selected Industrial Projects
1998-09 *	Th. Lehmann: Geometrische Ausrichtung medizinischer Bilder am Beispiel intraoraler Radiographien
1998-10 *	M. Nicola, M. Jarke: Performance Modeling of Distributed and Replicated Databases
1998-11 *	Ansgar Schleicher, Bernhard Westfechtel, Dirk Jäger: Modeling Dynamic Software Processes in UML
1998-12 *	W. Appelt, M. Jarke: Interoperable Tools for Cooperation Support using the World Wide Web
1998-13	Klaus Indermark: Semantik rekursiver Funktionsdefinitionen mit Striktheitsinformation
1999-01 *	Jahresbericht 1998
1999-02 *	F. Huch: Verifcation of Erlang Programs using Abstract Interpretation and Model Checking — Extended Version
1999-03 *	R. Gallersdörfer, M. Jarke, M. Nicola: The ADR Replication Manager
1999-04	María Alpuente, Michael Hanus, Salvador Lucas, Germán Vidal: Specialization of Functional Logic Programs Based on Needed Narrowing
1999-05 *	W. Thomas (Ed.): DLT 99 - Developments in Language Theory Fourth International Conference
1999-06 *	Kai Jakobs, Klaus-Dieter Kleefeld: Informationssysteme für die angewandte historische Geographie
1999-07	Thomas Wilke: CTL+ is exponentially more succinct than CTL
1999-08	Oliver Matz: Dot-Depth and Monadic Quantifier Alternation over Pictures
2000-01 *	Jahresbericht 1999
2000-02	Jens Vöge, Marcin Jurdzinski A Discrete Strategy Improvement Algorithm for Solving Parity Games
2000-03	D. Jäger, A. Schleicher, B. Westfechtel: UPGRADE: A Framework for Building Graph-Based Software Engineering Tools
2000-04	Andreas Becks, Stefan Sklorz, Matthias Jarke: Exploring the Semantic Structure of Technical Document Collections: A Cooperative Systems Approach
2000-05	Mareike Schoop: Cooperative Document Management
2000-06	Mareike Schoop, Christoph Quix (eds.): Proceedings of the Fifth International Workshop on the Language-Action Perspective on Communication Modelling
2000-07 *	Markus Mohnen, Pieter Koopman (Eds.): Proceedings of the 12th International Workshop of Functional Languages

2000-08	Thomas Arts, Thomas Noll: Verifying Generic Erlang Client-Server Implementations
2001-01 *	Jahresbericht 2000
2001-02	Benedikt Bollig, Martin Leucker: Deciding LTL over Mazurkiewicz Traces
2001-03	Thierry Cachat: The power of one-letter rational languages
2001-04	Benedikt Bollig, Martin Leucker, Michael Weber: Local Parallel Model Checking for the Alternation Free mu-Calculus
2001-05	Benedikt Bollig, Martin Leucker, Thomas Noll: Regular MSC Languages
2001-06	Achim Blumensath: Prefix-Recognisable Graphs and Monadic Second-Order Logic
2001-07	Martin Grohe, Stefan Wöhrle: An Existential Locality Theorem
2001-08	Mareike Schoop, James Taylor (eds.): Proceedings of the Sixth International Workshop on the Language-Action Perspective on Communication Modelling
2001-09	Thomas Arts, Jürgen Giesl: A collection of examples for termination of term rewriting using dependency pairs
2001-10	Achim Blumensath: Axiomatising Tree-interpretable Structures
2001-11	Klaus Indermark, Thomas Noll (eds.): Kolloquium Programmiersprachen und Grundlagen der Programmierung
2002-01 *	Jahresbericht 2001
2002-02	Jürgen Giesl, Aart Middeldorp: Transformation Techniques for Context-Sensitive Rewrite Systems
2002-03	Benedikt Bollig, Martin Leucker, Thomas Noll: Generalised Regular MSC Languages
2002-04	Jürgen Giesl, Aart Middeldorp: Innermost Termination of Context-Sensitive Rewriting
2002-05	Horst Lichter, Thomas von der Maßen, Thomas Weiler: Modelling Requirements and Architectures for Software Product Lines
2002-06	Henry N. Adorna: 3-Party Message Complexity is Better than 2-Party Ones for Proving Lower Bounds on the Size of Minimal Nondeterministic Finite Automata
2002-07	Jörg Dahmen: Invariant Image Object Recognition using Gaussian Mixture Densities
2002-08	Markus Mohnen: An Open Framework for Data-Flow Analysis in Java
2002-09	Markus Mohnen: Interfaces with Default Implementations in Java
2002-10	Martin Leucker: Logics for Mazurkiewicz traces
2002-11	Jürgen Giesl, Hans Zantema: Liveness in Rewriting
2003-01 *	Jahresbericht 2002
2003-02	Jürgen Giesl, René Thiemann: Size-Change Termination for Term Rewriting
2003-03	Jürgen Giesl, Deepak Kapur: Deciding Inductive Validity of Equations
2003-04	Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, Stephan Falke: Improving Dependency Pairs
2003-05	Christof Löding, Philipp Rohde: Solving the Sabotage Game is PSPACE-hard
2003-06	Franz Josef Och: Statistical Machine Translation: From Single-Word Models to Alignment Templates

2003-07	Horst Lichter, Thomas von der Maßen, Alexander Nyßen, Thomas Weiler: Vergleich von Ansätzen zur Feature Modellierung bei der Softwareproduktlinienentwicklung
2003-08	Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, Stephan Falke: Mechanizing Dependency Pairs
2004-01 *	Fachgruppe Informatik: Jahresbericht 2003
2004-02	Benedikt Bollig, Martin Leucker: Message-Passing Automata are expressively equivalent to EMSO logic
2004-03	Delia Kesner, Femke van Raamsdonk, Joe Wells (eds.): HOR 2004 – 2nd International Workshop on Higher-Order Rewriting
2004-04	Slim Abdennadher, Christophe Ringeissen (eds.): RULE 04 – Fifth International Workshop on Rule-Based Programming
2004-05	Herbert Kuchen (ed.): WFLP 04 – 13th International Workshop on Functional and (Constraint) Logic Programming
2004-06	Sergio Antoy, Yoshihito Toyama (eds.): WRS 04 – 4th International Workshop on Reduction Strategies in Rewriting and Programming
2004-07	Michael Codish, Aart Middeldorp (eds.): WST 04 – 7th International Workshop on Termination
2004-08	Klaus Indermark, Thomas Noll: Algebraic Correctness Proofs for Compiling Recursive Function Definitions with Strictness Information
2004-09	Joachim Kneis, Daniel Mölle, Stefan Richter, Peter Rossmanith: Parameterized Power Domination Complexity
2004-10	Zinaida Benenson, Felix C. Gärtner, Dogan Kesdogan: Secure Multi-Party Computation with Security Modules
2005-01 *	Fachgruppe Informatik: Jahresbericht 2004
2005-02	Maximillian Dornseif, Felix C. Gärtner, Thorsten Holz, Martin Mink: An Offensive Approach to Teaching Information Security: "Aachen Summer School Applied IT Security"
2005-03	Jürgen Giesl, René Thiemann, Peter Schneider-Kamp: Proving and Disproving Termination of Higher-Order Functions
2005-04	Daniel Mölle, Stefan Richter, Peter Rossmanith: A Faster Algorithm for the Steiner Tree Problem
2005-05	Fabien Pouget, Thorsten Holz: A Pointillist Approach for Comparing Honeypots
2005-06	Simon Fischer, Berthold Vöcking: Adaptive Routing with Stale Information
2005-07	Felix C. Freiling, Thorsten Holz, Georg Wicherski: Botnet Tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks
2005-08	Joachim Kneis, Peter Rossmanith: A New Satisfiability Algorithm With Applications To Max-Cut
2005-09	Klaus Kursawe, Felix C. Freiling: Byzantine Fault Tolerance on General Hybrid Adversary Structures
2005-10	Benedikt Bollig: Automata and Logics for Message Sequence Charts
2005-11	Simon Fischer, Berthold Vöcking: A Counterexample to the Fully Mixed Nash Equilibrium Conjecture

2005-12	Neeraj Mittal, Felix Freiling, S. Venkatesan, Lucia Draque Penso: Efficient Reductions for Wait-Free Termination Detection in Faulty Distributed Systems
2005-13	Carole Delporte-Gallet, Hugues Fauconnier, Felix C. Freiling: Revisiting Failure Detection and Consensus in Omission Failure Environments
2005-14	Felix C. Freiling, Sukumar Ghosh: Code Stabilization
2005-15	Uwe Naumann: The Complexity of Derivative Computation
2005-16	Uwe Naumann: Syntax-Directed Derivative Code (Part I: Tangent-Linear Code)
2005-17	Uwe Naumann: Syntax-directed Derivative Code (Part II: Intraprocedural Adjoint Code)
2005-18	Thomas von der Maßen, Klaus Müller, John MacGregor, Eva Geisberger, Jörg Dörr, Frank Houdek, Harbhajan Singh, Holger Wußmann, Hans-Veit Bacher, Barbara Paech: Einsatz von Features im Software-Entwicklungsprozess - Abschlußbericht des GI-Arbeitskreises "Features"
2005-19	Uwe Naumann, Andre Vehreschild: Tangent-Linear Code by Augmented LL-Parsers
2005-20	Felix C. Freiling, Martin Mink: Bericht über den Workshop zur Ausbildung im Bereich IT-Sicherheit Hochschulausbildung, berufliche Weiterbildung, Zertifizierung von Ausbildungsangeboten am 11. und 12. August 2005 in Köln organisiert von RWTH Aachen in Kooperation mit BITKOM, BSI, DLR und Gesellschaft fuer Informatik (GI) e.V.
2005-21	Thomas Noll, Stefan Rieger: Optimization of Straight-Line Code Revisited
2005-22	Felix Freiling, Maurice Herlihy, Lucia Draque Penso: Optimal Randomized Fair Exchange with Secret Shared Coins
2005-23	Heiner Ackermann, Alantha Newman, Heiko Röglin, Berthold Vöcking: Decision Making Based on Approximate and Smoothed Pareto Curves
2005-24	Alexander Becher, Zinaida Benenson, Maximillian Dornseif: Tampering with Motes: Real-World Physical Attacks on Wireless Sensor Networks
2006-01 *	Fachgruppe Informatik: Jahresbericht 2005
2006-02	Michael Weber: Parallel Algorithms for Verification of Large Systems
2006-03	Michael Maier, Uwe Naumann: Intraprocedural Adjoint Code Generated by the Differentiation-Enabled NAGWare Fortran Compiler
2006-04	Ebadollah Varnik, Uwe Naumann, Andrew Lyons: Toward Low Static Memory Jacobian Accumulation
2006-05	Uwe Naumann, Jean Utke, Patrick Heimbach, Chris Hill, Derya Ozyurt, Carl Wunsch, Mike Fagan, Nathan Tallent, Michelle Strout: Adjoint Code by Source Transformation with OpenAD/F
2006-06	Joachim Kneis, Daniel Mölle, Stefan Richter, Peter Rossmanith: Divide-and-Color
2006-07	Thomas Colcombet, Christof Löding: Transforming structures by set interpretations
2006-08	Uwe Naumann, Yuxiao Hu: Optimal Vertex Elimination in Single-Expression-Use Graphs
2006-09	Tingting Han, Joost-Pieter Katoen: Counterexamples in Probabilistic Model Checking

2006-10	Mesut Günes, Alexander Zimmermann, Martin Wenig, Jan Ritzerfeld, Ulrich Meis: From Simulations to Testbeds - Architecture of the Hybrid MCG-Mesh Testbed
2006-11	Bastian Schlich, Michael Rohrbach, Michael Weber, Stefan Kowalewski: Model Checking Software for Microcontrollers
2006-12	Benedikt Bollig, Joost-Pieter Katoen, Carsten Kern, Martin Leucker: Replaying Play in and Play out: Synthesis of Design Models from Scenarios by Learning
2006-13	Wong Karianto, Christof Löding: Unranked Tree Automata with Sibling Equalities and Disequalities
2006-14	Danilo Beuche, Andreas Birk, Heinrich Dreier, Andreas Fleischmann, Heidi Galle, Gerald Heller, Dirk Janzen, Isabel John, Ramin Tavakoli Kolagari, Thomas von der Maßen, Andreas Wolfram: Report of the GI Work Group "Requirements Management Tools for Product Line Engineering"
2006-15	Sebastian Ullrich, Jakob T. Valvoda, Torsten Kuhlen: Utilizing optical sensors from mice for new input devices
2006-16	Rafael Ballagas, Jan Borchers: Selexels: a Conceptual Framework for Pointing Devices with Low Expressiveness
2006-17	Eric Lee, Henning Kiel, Jan Borchers: Scrolling Through Time: Improving Interfaces for Searching and Navigating Continuous Audio Timelines
2007-01 *	Fachgruppe Informatik: Jahresbericht 2006
2007-02	Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann, and Harald Zankl: SAT Solving for Termination Analysis with Polynomial Interpretations
2007-03	Jürgen Giesl, René Thiemann, Stephan Swiderski, and Peter Schneider-Kamp: Proving Termination by Bounded Increase
2007-04	Jan Buchholz, Eric Lee, Jonathan Klein, and Jan Borchers: coJIVE: A System to Support Collaborative Jazz Improvisation
2007-05	Uwe Naumann: On Optimal DAG Reversal
2007-06	Joost-Pieter Katoen, Thomas Noll, and Stefan Rieger: Verifying Concurrent List-Manipulating Programs by LTL Model Checking
2007-07	Alexander Nyßen, Horst Lichter: MeDUSA - MethoD for UML2-based Design of Embedded Software Applications
2007-08	Falk Salewski and Stefan Kowalewski: Achieving Highly Reliable Embedded Software: An empirical evaluation of different approaches
2007-09	Tina Kraußer, Heiko Mantel, and Henning Sudbrock: A Probabilistic Justification of the Combining Calculus under the Uniform Scheduler Assumption
2007-10	Martin Neuhäußer, Joost-Pieter Katoen: Bisimulation and Logical Preservation for Continuous-Time Markov Decision Processes
2007-11	Klaus Wehrle (editor): 6. Fachgespräch Sensornetzwerke
2007-12	Uwe Naumann: An L-Attributed Grammar for Adjoint Code
2007-13	Uwe Naumann, Michael Maier, Jan Riehme, and Bruce Christianson: Second-Order Adjoints by Source Code Manipulation of Numerical Programs

2007-14	Jean Utke, Uwe Naumann, Mike Fagan, Nathan Tallent, Michelle Strout, Patrick Heimbach, Chris Hill, and Carl Wunsch: OpenAD/F: A Modular, Open-Source Tool for Automatic Differentiation of Fortran Codes
2007-15	Volker Stolz: Temporal assertions for sequential and concurrent programs
2007-16	Sadeq Ali Makram, Mesut Güneç, Martin Wenig, Alexander Zimmermann: Adaptive Channel Assignment to Support QoS and Load Balancing for Wireless Mesh Networks
2007-17	René Thiemann: The DP Framework for Proving Termination of Term Rewriting
2007-18	Uwe Naumann: Call Tree Reversal is NP-Complete
2007-19	Jan Riehme, Andrea Walther, Jörg Stiller, Uwe Naumann: Adjoints for Time-Dependent Optimal Control
2007-20	Joost-Pieter Katoen, Daniel Klink, Martin Leucker, and Verena Wolf: Three-Valued Abstraction for Probabilistic Systems
2007-21	Tingting Han, Joost-Pieter Katoen, and Alexandru Mereacre: Compositional Modeling and Minimization of Time-Inhomogeneous Markov Chains
2007-22	Heiner Ackermann, Paul W. Goldberg, Vahab S. Mirrokni, Heiko Röglin, and Berthold Vöcking: Uncoordinated Two-Sided Markets
2008-01 *	Fachgruppe Informatik: Jahresbericht 2007/2008
2008-02	Henrik Bohnenkamp, Marielle Stoelinga: Quantitative Testing
2008-03	Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann, Harald Zankl: Maximal Termination
2008-04	Uwe Naumann, Jan Riehme: Sensitivity Analysis in Sisyphe with the AD-Enabled NAGWare Fortran Compiler
2008-05	Frank G. Radmacher: An Automata Theoretic Approach to the Theory of Rational Tree Relations
2008-06	Uwe Naumann, Laurent Hascoet, Chris Hill, Paul Hovland, Jan Riehme, Jean Utke: A Framework for Proving Correctness of Adjoint Message Passing Programs
2008-07	Alexander Nyßen, Horst Lichter: The MeDUSA Reference Manual, Second Edition
2008-08	George B. Mertzios, Stavros D. Nikolopoulos: The λ-cluster Problem on Parameterized Interval Graphs
2008-09	George B. Mertzios, Walter Unger: An optimal algorithm for the k-fixed-endpoint path cover on proper interval graphs
2008-10	George B. Mertzios, Walter Unger: Preemptive Scheduling of Equal-Length Jobs in Polynomial Time
2008-11	George B. Mertzios: Fast Convergence of Routing Games with Splittable Flows
2008-12	Joost-Pieter Katoen, Daniel Klink, Martin Leucker, Verena Wolf: Abstraction for stochastic systems by Erlang's method of stages
2008-13	Beatriz Alarcón, Fabian Emmes, Carsten Fuhs, Jürgen Giesl, Raúl Gutiérrez, Salvador Lucas, Peter Schneider-Kamp, René Thiemann: Improving Context-Sensitive Dependency Pairs
2008-14	Bastian Schlich: Model Checking of Software for Microcontrollers
2008-15	Joachim Kneis, Alexander Langer, Peter Rossmanith: A New Algorithm for Finding Trees with Many Leaves

2008-16	Hendrik vom Lehn, Elias Weingärtner and Klaus Wehrle: Comparing recent network simulators: A performance evaluation study
2008-17	Peter Schneider-Kamp: Static Termination Analysis for Prolog using Term Rewriting and SAT Solving
2008-18	Falk Salewski: Empirical Evaluations of Safety-Critical Embedded Systems
2008-19	Dirk Wilking: Empirical Studies for the Application of Agile Methods to Embedded Systems
2009-01 *	Fachgruppe Informatik: Jahresbericht 2009
2009-02	Taolue Chen, Tingting Han, Joost-Pieter Katoen, Alexandru Mereacre: Quantitative Model Checking of Continuous-Time Markov Chains Against Timed Automata Specifications
2009-03	Alexander Nyßen: Model-Based Construction of Embedded & Real-Time Software - A Methodology for Small Devices
2009-05	George B. Mertzios, Ignasi Sau, Shmuel Zaks: A New Intersection Model and Improved Algorithms for Tolerance Graphs
2009-06	George B. Mertzios, Ignasi Sau, Shmuel Zaks: The Recognition of Tolerance and Bounded Tolerance Graphs is NP-complete
2009-07	Joachim Kneis, Alexander Langer, Peter Rossmanith: Derandomizing Non-uniform Color-Coding I
2009-08	Joachim Kneis, Alexander Langer: Satellites and Mirrors for Solving Independent Set on Sparse Graphs
2009-09	Michael Nett: Implementation of an Automated Proof for an Algorithm Solving the Maximum Independent Set Problem
2009-10	Felix Reidl, Fernando Sánchez Villaamil: Automatic Verification of the Correctness of the Upper Bound of a Maximum Independent Set Algorithm
2009-11	Kyriaki Ioannidou, George B. Mertzios, Stavros D. Nikolopoulos: The Longest Path Problem is Polynomial on Interval Graphs
2009-12	Martin Neuhäußer, Lijun Zhang: Time-Bounded Reachability in Continuous-Time Markov Decision Processes
2009-13	Martin Zimmermann: Time-optimal Winning Strategies for Poset Games
2009-14	Ralf Huuck, Gerwin Klein, Bastian Schlich (eds.): Doctoral Symposium on Systems Software Verification (DS SSV'09)
2009-15	Joost-Pieter Katoen, Daniel Klink, Martin Neuhäußer: Compositional Abstraction for Stochastic Systems
2009-16	George B. Mertzios, Derek G. Corneil: Vertex Splitting and the Recognition of Trapezoid Graphs
2009-17	Carsten Kern: Learning Communicating and Nondeterministic Automata
2009-18	Paul Hänsch, Michaela Slaats, Wolfgang Thomas: Parametrized Regular Infinite Games and Higher-Order Pushdown Strategies
2010-01 *	Fachgruppe Informatik: Jahresbericht 2010
2010-02	Daniel Neider, Christof Löding: Learning Visibly One-Counter Automata in Polynomial Time
2010-03	Holger Krahn: MontiCore: Agile Entwicklung von domänenspezifischen Sprachen im Software-Engineering

2010-04	René Wörzberger: Management dynamischer Geschäftsprozesse auf Basis statischer Prozessmanagementsysteme
2010-05	Daniel Retkowitz: Softwareunterstützung für adaptive eHome-Systeme
2010-06	Taolue Chen, Tingting Han, Joost-Pieter Katoen, Alexandru Mereacre: Computing maximum reachability probabilities in Markovian timed automata
2010-07	George B. Mertzios: A New Intersection Model for Multitolerance Graphs, Hierarchy, and Efficient Algorithms
2010-08	Carsten Otto, Marc Brockschmidt, Christian von Essen, Jürgen Giesl: Automated Termination Analysis of Java Bytecode by Term Rewriting
2010-09	George B. Mertzios, Shmuel Zaks: The Structure of the Intersection of Tolerance and Cocomparability Graphs
2010-10	Peter Schneider-Kamp, Jürgen Giesl, Thomas Ströder, Alexander Serebrenik, René Thiemann: Automated Termination Analysis for Logic Programs with Cut
2010-11	Martin Zimmermann: Parametric LTL Games
2010-12	Thomas Ströder, Peter Schneider-Kamp, Jürgen Giesl: Dependency Triples for Improving Termination Analysis of Logic Programs with Cut
2010-13	Ashraf Armoush: Design Patterns for Safety-Critical Embedded Systems
2010-14	Michael Codish, Carsten Fuhs, Jürgen Giesl, Peter Schneider-Kamp: Lazy Abstraction for Size-Change Termination
2010-15	Marc Brockschmidt, Carsten Otto, Christian von Essen, Jürgen Giesl: Termination Graphs for Java Bytecode
2010-16	Christian Berger: Automating Acceptance Tests for Sensor- and Actuator-based Systems on the Example of Autonomous Vehicles
2010-17	Hans Grönniger: Systemmodell-basierte Definition objektbasierter Modellierungssprachen mit semantischen Variationspunkten
2010-18	Ibrahim Armaç: Personalisierte eHomes: Mobilität, Privatsphäre und Sicherheit
2010-19	Felix Reidl: Experimental Evaluation of an Independent Set Algorithm
2010-20	Wladimir Fridman, Christof Löding, Martin Zimmermann: Degrees of Lookahead in Context-free Infinite Games
2011-01 *	Fachgruppe Informatik: Jahresbericht 2011
2011-02	Marc Brockschmidt, Carsten Otto, Jürgen Giesl: Modular Termination Proofs of Recursive Java Bytecode Programs by Term Rewriting
2011-03	Lars Noschinski, Fabian Emmes, Jürgen Giesl: A Dependency Pair Framework for Innermost Complexity Analysis of Term Rewrite Systems
2011-04	Christina Jansen, Jonathan Heinen, Joost-Pieter Katoen, Thomas Noll: A Local Greibach Normal Form for Hyperedge Replacement Grammars
2011-06	Johannes Lotz, Klaus Leppkes, and Uwe Naumann: dco/c++ - Derivative Code by Overloading in C++
2011-07	Shahar Maoz, Jan Oliver Ringert, Bernhard Rumpe: An Operational Semantics for Activity Diagrams using SMV
2011-08	Thomas Ströder, Fabian Emmes, Peter Schneider-Kamp, Jürgen Giesl, Carsten Fuhs: A Linear Operational Semantics for Termination and Complexity Analysis of ISO Prolog
2011-09	Markus Beckers, Johannes Lotz, Viktor Mosenkis, Uwe Naumann (Editors): Fifth SIAM Workshop on Combinatorial Scientific Computing

2011-10	Markus Beckers, Viktor Mosenkis, Michael Maier, Uwe Naumann: Adjoint Subgradient Calculation for McCormick Relaxations
2011-11	Nils Jansen, Erika Ábrahám, Jens Katelaan, Ralf Wimmer, Joost-Pieter Katoen, Bernd Becker: Hierarchical Counterexamples for Discrete-Time Markov Chains
2011-12	Ingo Felscher, Wolfgang Thomas: On Compositional Failure Detection in Structured Transition Systems
2011-13	Michael Förster, Uwe Naumann, Jean Utke: Toward Adjoint OpenMP
2011-14	Daniel Neider, Roman Rabinovich, Martin Zimmermann: Solving Muller Games via Safety Games
2011-16	Niloofar Safiran, Uwe Naumann: Toward Adjoint OpenFOAM
2011-17	Carsten Fuhs: SAT Encodings: From Constraint-Based Termination Analysis to Circuit Synthesis
2011-18	Kamal Barakat: Introducing Timers to pi-Calculus
2011-19	Marc Brockschmidt, Thomas Ströder, Carsten Otto, Jürgen Giesl: Automated Detection of Non-Termination and NullPointerExceptions for Java Bytecode
2011-24	Callum Corbett, Uwe Naumann, Alexander Mitsos: Demonstration of a Branch-and-Bound Algorithm for Global Optimization using McCormick Relaxations
2011-25	Callum Corbett, Michael Maier, Markus Beckers, Uwe Naumann, Amin Ghobeity, Alexander Mitsos: Compiler-Generated Subgradient Code for McCormick Relaxations
2011-26	Hongfei Fu: The Complexity of Deciding a Behavioural Pseudometric on Probabilistic Automata
2012-01	Fachgruppe Informatik: Annual Report 2012
2012-02	Thomas Heer: Controlling Development Processes
2012-03	Arne Haber, Jan Oliver Ringert, Bernhard Rumpe: MontiArc - Architectural Modeling of Interactive Distributed and Cyber-Physical Systems
2012-04	Marcus Gelderie: Strategy Machines and their Complexity
2012-05	Thomas Ströder, Fabian Emmes, Jürgen Giesl, Peter Schneider-Kamp, and Carsten Fuhs: Automated Complexity Analysis for Prolog by Term Rewriting
2012-06	Marc Brockschmidt, Richard Musiol, Carsten Otto, Jürgen Giesl: Automated Termination Proofs for Java Programs with Cyclic Data
2012-07	André Egners, Björn Marschollek, and Ulrike Meyer: Hackers in Your Pocket: A Survey of Smartphone Security Across Platforms
2012-08	Hongfei Fu: Computing Game Metrics on Markov Decision Processes
2012-09	Dennis Guck, Tingting Han, Joost-Pieter Katoen, and Martin R. Neuhäußer: Quantitative Timed Analysis of Interactive Markov Chains
2012-10	Uwe Naumann and Johannes Lotz: Algorithmic Differentiation of Numerical Methods: Tangent-Linear and Adjoint Direct Solvers for Systems of Linear Equations
2012-12	Jürgen Giesl, Thomas Ströder, Peter Schneider-Kamp, Fabian Emmes, and Carsten Fuhs: Symbolic Evaluation Graphs and Term Rewriting — A General Methodology for Analyzing Logic Programs

2012-15	Uwe Naumann, Johannes Lotz, Klaus Leppkes, and Markus Towara: Algorithmic Differentiation of Numerical Methods: Tangent-Linear and Adjoint Solvers for Systems of Nonlinear Equations
2012-16	Georg Neugebauer and Ulrike Meyer: SMC-MuSe: A Framework for Secure Multi-Party Computation on MultiSets
2012-17	Viet Yen Nguyen: Trustworthy Spacecraft Design Using Formal Methods
2013-01 *	Fachgruppe Informatik: Annual Report 2013
2013-02	Michael Reke: Modellbasierte Entwicklung automobiler Steuerungssysteme in Klein- und mittelständischen Unternehmen
2013-03	Markus Towara and Uwe Naumann: A Discrete Adjoint Model for OpenFOAM
2013-04	Max Sagebaum, Nicolas R. Gauger, Uwe Naumann, Johannes Lotz, and Klaus Leppkes: Algorithmic Differentiation of a Complex C++ Code with Underlying Libraries
2013-05	Andreas Rausch and Marc Sihling: Software & Systems Engineering Essentials 2013
2013-06	Marc Brockschmidt, Byron Cook, and Carsten Fuhs: Better termination proving through cooperation
2013-07	André Stollenwerk: Ein modellbasiertes Sicherheitskonzept für die extrakorporale Lungenunterstützung
2013-08	Sebastian Junges, Ulrich Loup, Florian Corzilius and Erika Ábrahám: On Gröbner Bases in the Context of Satisfiability-Modulo-Theories Solving over the Real Numbers
2013-10	Joost-Pieter Katoen, Thomas Noll, Thomas Santen, Dirk Seifert, and Hao Wu: Performance Analysis of Computing Servers using Stochastic Petri Nets and Markov Automata
2013-12	Marc Brockschmidt, Fabian Emmes, Stephan Falke, Carsten Fuhs, and Jürgen Giesl: Alternating Runtime and Size Complexity Analysis of Integer Programs
2013-13	Michael Eggert, Roger Häußling, Martin Henze, Lars Hermerschmidt, René Hummen, Daniel Kerpen, Antonio Navarro Pérez, Bernhard Rumpe, Dirk Thißen, and Klaus Wehrle: SensorCloud: Towards the Interdisciplinary Development of a Trustworthy Platform for Globally Interconnected Sensors and Actuators
2013-14	Jörg Brauer: Automatic Abstraction for Bit-Vectors using Decision Procedures
2013-19	Florian Schmidt, David Orlea, and Klaus Wehrle: Support for error tolerance in the Real-Time Transport Protocol
2013-20	Jacob Palczynski: Time-Continuous Behaviour Comparison Based on Abstract Models
2014-01 *	Fachgruppe Informatik: Annual Report 2014
2014-02	Daniel Merschen: Integration und Analyse von Artefakten in der modellbasierten Entwicklung eingebetteter Software
2014-04	Namit Chaturvedi: Languages of Infinite Traces and Deterministic Asynchronous Automata
2014-05	Thomas Ströder, Jürgen Giesl, Marc Brockschmidt, Florian Frohn, Carsten Fuhs, Jera Hensel, and Peter Schneider-Kamp: Automated Termination Analysis for Programs with Pointer Arithmetic

2014-06	Esther Horbert, Germán Martín García, Simone Frintrop, and Bastian Leibe: Sequence Level Salient Object Proposals for Generic Object Detection in Video

* These reports are only available as a printed version.
Please contact `biblio@informatik.rwth-aachen.de` to obtain copies.