SIAM J. SCI. COMPUT.
Vol. 46, No. 3, pp. A1423–A1440
© 2024 Society for Industrial and Applied Mathematics

Downloaded 07/01/24 to 134.100.220.139 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

A MATRIX-FREE EXACT NEWTON METHOD*

UWE NAUMANN†

Abstract. A modification of Newton's method for solving systems of $n > 0$ nonlinear equations is presented. The new matrix-free method is exact as opposed to a range of inexact Newton methods in the sense that both the Jacobians and the solutions to the linear Newton systems are computed without truncation. It relies on a given decomposition of a structurally dense invertible Jacobian of the residual into a product of $q > 0$ structurally sparse invertible elemental Jacobians according to the chain rule of differentiation. Inspired by the adjoint mode of algorithmic differentiation, explicit accumulation of the Jacobian of the residual is avoided. Prospective, generally applicable implementations of the new method can be based on similar ideas. Sparsity is exploited for the direct solution of the linear Newton systems. Optimal exploitation of sparsity yields various well-known computationally intractable combinatorial optimization problems in sparse linear algebra such as BANDWIDTH or DIRECTED ELIMINATION ORDERING. The method is motivated in the context of a decomposition into elemental Jacobians with bandwidth $2\mu + 1$ for $0 \leq \mu \ll n$. In the likely scenario of $q > n$, the computational cost of the standard Newton algorithm is dominated by the cost of accumulating the Jacobian of the residual. It can be estimated as $\mathcal{O}(q\mu n^2)$, thus exceeding the cost of $\mathcal{O}(n^3)$ for the direct solution of the linear Newton system. The new method reduces this cost to $\mathcal{O}(qn\mu^2)$, yielding a potential improvement by a factor of $\mathcal{O}\left(\frac{n}{\mu}\right)$. Supporting run time measurements are presented for the tridiagonal case showing a reduction of the computational cost by $\mathcal{O}(n)$. Generalization yields the combinatorial MATRIX-FREE EXACT NEWTON STEP problem. We prove NP-completeness, and we present algorithmic components for building methods for the approximate solution. Potential applications of the matrix-free exact Newton method in machine learning of surrogates for computationally expensive nonlinear residuals are touched on briefly as part of various conclusions to be drawn.

Key words. Newton method, matrix-free, algorithmic differentiation, adjoint, machine learning, surrogate model

MSC codes. 49M15, 47A05, 68N99

DOI. 10.1137/23M157017X

1. Introduction.

We revisit Newton's method [29, 36] for computing roots

$$x \in \mathbb{R}^n : F(x) = 0$$

of differentiable multivariate vector functions (residuals)

$$F : \mathbb{R}^n \to \mathbb{R}^n : y = F(x)$$

with invertible dense Jacobians

$$F'(x) \equiv \left(\frac{dy_j}{dx_i}\right)_{i,j=1,\ldots,n} \in \mathbb{R}^{n \times n},$$

and where $x = (x_i)$ and $y = (y_i)$ for $i = 1, \ldots, n$. Building on the first-order Taylor expansion

$$F(x + \Delta x) = F(x) + F'(x) \cdot \Delta x + \mathcal{O}(\Delta x^2)$$

*Submitted to the journal's Numerical Algorithms for Scientific Computing section May 2, 2023; accepted for publication (in revised form) January 31, 2024; published electronically May 2, 2024.
https://doi.org/10.1137/23M157017X

†Software and Tools for Computational Engineering, RWTH Aachen University, Aachen 52056, Germany (naumann@stce.rwth-aachen.de).

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/01/24 to 134.100.220.139 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

of the residual for $0 \leq \Delta x \ll 1$, the nonlinear problem $F(x) = 0$ is replaced by its linearization $F(x) + F'(x) \cdot \Delta x = 0$, immediately yielding

$$\Delta x = -F'(x)^{-1} \cdot F(x)$$

as the solution of the linear Newton system

$$F'(x) \cdot \Delta x = -F(x)\,.$$

Approximate solutions for which $F(x) \approx 0$ are computed iteratively as

$$\begin{aligned} \Delta x &:= -F'(x)^{-1} \cdot F(x), \\ x &:= x + \Delta x, \end{aligned} \tag{1.1}$$

for given starting points $x \in \mathbb{R}^n$. Convergence after $p \geq 0$ iterations is typically defined as the norm $\|F(x)\|$ of the residual falling below some given threshold $0 < \epsilon \ll 1$. It requires $\|G'(x)\| < 1$, where $G(x) = x - F'(x)^{-1} \cdot F(x)$ results from restating (1.1) as a fixed-point iteration. Line search methods can help in the not unlikely case of violation. See, for example, [6, 20] for further information on Newton's method.

We use $=$ to denote mathematical equality, $\equiv$ in the sense of "is defined as," and $:=$ to represent assignment according to imperative programming. Approximate equality is denoted as $\approx$. We distinguish between partial (∂) and total (d) derivatives. Multiplication is denoted by a dot. The dot may also be omitted in favor of a more compact notation.

Newton's method generalizes naturally to unconstrained convex nonlinear optimization (e.g., minimization) problems

$$\min_{x \in \mathbb{R}^n} f(x)$$

through numerical approximation of the first-order optimality condition

$$f'(x) = 0 \in \mathbb{R}^n\,.$$

Equation (1.1) becomes

$$\begin{aligned} \Delta x &:= -f''(x)^{-1} \cdot f'(x), \\ x &:= x + \Delta x\,. \end{aligned}$$

The type of stationary point is validated by evaluating the definiteness of the Hessian $f''(x) \in \mathbb{R}^{n \times n}$.

We consider exact Newton methods for both systems of nonlinear equations and unconstrained convex nonlinear optimization problems. All derivatives (Jacobians $F'(x)$, gradients $f'(x)$, Hessians $f''(x)$) are computed without truncation (with machine precision) by algorithmic differentiation (AD) [17, 27]. The linear Newton systems are solved by direct methods based, for example, on Gaussian, Cholesky, or orthogonal factorizations [12]. By contrast, inexact Newton methods rely on approximations of derivatives (see also quasi-Newton methods [3]) and/or of solutions to the linear systems (see also truncated Newton methods [5]).

The residual F is usually given as a differentiable program written in some high-level programming language. Equation (1.1) implies that the two main ingredients of Newton's method are the evaluation of the Jacobian $F'(x)$ and the solution of the linear Newton system $F'(x) \cdot \Delta x = -F(x)$. AD of F yields $F'(x)$ with machine

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/01/24 to 134.100.220.139 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

accuracy at a computational cost of $\mathcal{O}(n) \cdot \mathrm{Cost}(F)$. Subsequent direct solution of the linear Newton system yields the Newton steps $\Delta x \in \mathbb{R}^n$ at the computational cost of $\mathcal{O}(n^3)$. The overall computational cost can be dominated by either of the two parts depending on the ratio $\frac{\mathrm{Cost}(F)}{\mathcal{O}(n^2)}$. In many cases, $\mathrm{Cost}(F) > \mathcal{O}(n^2)$, making the computational cost of Newton's method for residuals with dense Jacobians dominated by the cost of differentiation.

Exploitation of special structure of F can yield a significant reduction in computational cost. Consequently, this article proposes a novel matrix-free exact Newton method motivated by savings obtained for relevant scenarios. We present the fundamental idea behind the method and we discuss further generalization. In section 2 we recall essential fundamentals of AD and draw conclusions for the computation of Newton steps. The potential for reduction in computational cost is illustrated with the help of the practically relevant special case of banded elemental Jacobians in section 3. Formal complexity analysis and further generalization are the subjects of sections 4 and 5. Conclusions are drawn in section 6 including comments on further exploitation of the exact matrix-free Newton method in the context of machine learning of surrogates for the residual.

2. A lesson from adjoint AD. First-order AD comes in two fundamental flavors. Tangent AD yields

$$\dot{y} = \dot{F}(x, \dot{x}) \equiv F'(x) \cdot \dot{x} \tag{2.1}$$

and, hence, the (dense) Jacobian with machine accuracy at $\mathcal{O}(n) \cdot \mathrm{Cost}(\dot{F})$ by letting $\dot{x}$ range over the Cartesian basis of $\mathbb{R}^n$. Adjoint AD computes

$$\bar{x} = \bar{F}(x, \bar{y}) \equiv F'(x)^T \cdot \bar{y}\,, \tag{2.2}$$

yielding the same Jacobian with up to machine accuracy at $\mathcal{O}(n) \cdot \mathrm{Cost}(\bar{F})$ by letting $\bar{y}$ range over the Cartesian basis of $\mathbb{R}^n$. Both tangent and adjoint AD are matrix-free methods in the sense that the Jacobian is not required explicitly in order to evaluate Equation (2.1) or (2.2). Both methods can be used to accumulate F'. Their costs differ according to the ratio

$$\mathcal{R} \equiv \frac{\mathrm{Cost}(\bar{F})}{\mathrm{Cost}(\dot{F})}\,,$$

where, typically, $\mathcal{R} \geq 1$. Moreover, $\mathrm{Cost}(\dot{F}) = \mathcal{O}(\mathrm{Cost}(F)) \approx 2 \cdot \mathcal{O}(\mathrm{Cost}(F))$ and $\mathrm{Cost}(\bar{F}) = \mathcal{O}(\mathrm{Cost}(F)) \approx 3 \cdot \mathcal{O}(\mathrm{Cost}(F))$. Actual costs depend on specifics of the given implementation of F as well as on the quality of the AD solution. The cost of adjoint AD in particular can easily exceed $10 \cdot \mathcal{O}(\mathrm{Cost}(F))$ in practice.

Tangent AD is usually the method of choice for computing Jacobians of the residual in the context of Newton's method. Sparsity of the Jacobian of the residual may change the picture [9]. Row or bidirectional compression can lead to superior computational cost of adjoint or hybrid (combining tangent and adjoint) methods.

Second-order tangent and adjoint AD follow naturally; see [17]. Hessians required by Newton's method for optimization of convex objectives $f : \mathbb{R}^n \to \mathbb{R}$ can be computed at $\mathcal{O}(n) \cdot \mathrm{Cost}(\dot{\bar{f}})$ using a second-order adjoint version

$$\dot{\bar{f}} = \dot{\bar{f}}(x, \bar{y}, \dot{x}) \equiv \bar{y} \cdot f''(x) \cdot \dot{x}$$

of f. Set $\bar{y} = 1 \in \mathbb{R}$, and let $\dot{x}$ range over the Cartesian basis vectors in $\mathbb{R}^n$. Corresponding matrix-free (inexact) Newton–Krylov methods [22] can be derived. They are based

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/01/24 to 134.100.220.139 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

on the observation that the Hessian-vector products required by Krylov-subspace methods (e.g., conjugate gradients [18]) can be computed by a second-order adjoint $\dot{\bar{f}}$ without prior accumulation of the Hessian. Similar remarks apply to the first-order scenario, where Jacobian-vector products can be obtained efficiently and matrix-free by tangent AD. The approach to be proposed in this article is different. It exploits special structure and local sparsity of F instead of global propagation of tangents or adjoints.

2.1. Terminology. AD requires the given implementation of the residual F as a computer program to be differentiable. The notation from [17] is modified only slightly.

DEFINITION 2.1 (differentiable program). *A* differentiable program

$$F : \mathbb{R}^n \to \mathbb{R}^m : \; y := F(x)$$

decomposes into a single assignment code

$$v_j := \varphi_j(v_i)_{i \in P_j} \quad \textit{for } j = 1, \ldots, p+m \,, \tag{2.3}$$

where $v_{i-n} = x_i$ *for* $i = 1, \ldots, n$ *and* $y_k = v_{p+k}$ *for* $k = 1, \ldots, m$ *and with differentiable* elemental functions φ_j, $j = 1, \ldots, p+m$, *featuring* elemental partial derivatives

$$\partial_{j,i} \equiv \frac{\partial \varphi_j}{\partial v_i} \quad \textit{for } i \in P_j \,.$$

The set of arguments of φ_j *(direct predecessors of* v_j*) is denoted by* P_j.

Elemental functions range from the built-in arithmetic operations (e.g., scalar floating-point multiplication) and intrinsic functions (e.g., sine) via basic linear algebra subprograms (e.g., level-2 and level-3 BLAS) to substantially more complex numerical methods (e.g., solvers for systems of linear or nonlinear equations) [10, 11]. Indeed, a given implementation of Newton's method could be regarded as an elemental function [28]. The main requirement for a function to qualify as elemental is the availability of its derivatives up to the necessary order. In this case it can be processed in the context of the chain rule of differentiation by AD.

DEFINITION 2.2 (directed acyclic graph). *Equation* (2.3) *induces a* directed acyclic graph (DAG) $G = (V, E)$, $V = (X, Z, Y)$, $X \cap Z \cap Y = \varnothing$,[1] $E \subseteq V \times V$ *such that* $X = \{1-n, \ldots, 0\}$, $Z = \{1, \ldots, p\}$, $Y = p+1, \ldots, p+m$, *and* $(i, j) \in E \Leftrightarrow i \in P_j$. *All edges* (i, j) *are labeled with* $\partial_{j,i}$.

DAGs of sample differentiable programs to be discussed further below are depicted in Figures 1 and 2.

DEFINITION 2.3 (layered DAG and elemental Jacobians). *A DAG of a matrix chain product*

$$F'_q \times \cdots \times F'_1 = F' \in \mathbb{R}^{m \times n}$$

of elemental Jacobians F'_i, $i = 1, \ldots, q$ *is called* layered DAG.

The set of vertices of a layered DAG can be partitioned such that all paths connecting vertices from one set with vertices from another have equal length. Paths connecting vertices i and j are denoted as $(i, \ldots, j)$. For example, the DAG in Figure 2(a)

[1] The trivial program x=x distinguishes x as an input ($\in X$) from x as an output ($\in Y$) while $Z = \varnothing$ in this case. Corresponding larger scenarios follow naturally.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

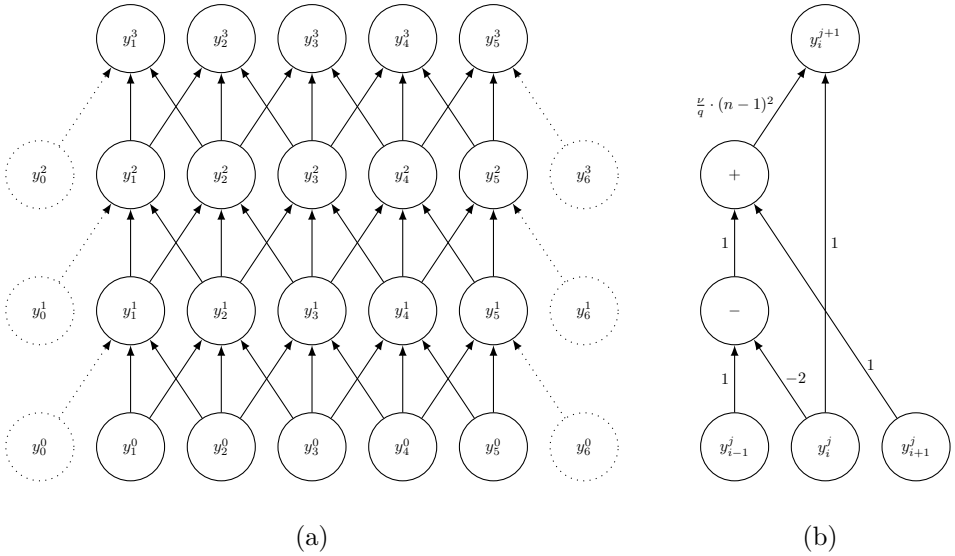

FIG. 1. *Initial-boundary value problem for the* 1*D diffusion equation.*

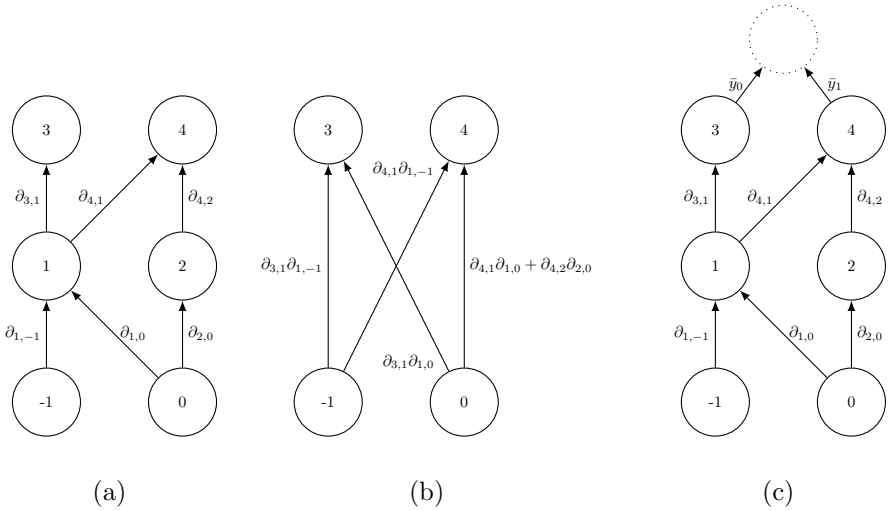

FIG. 2. *DAGs and AD.*

is layered. Its vertex set can be partitioned as $\{-1,0\} \cup \{1,2\} \cup \{3,4\}$ with all paths between partitions having length one, $(\{-1,0\},\ldots,\{1,2\})$ and $(\{1,2\},\ldots,\{3,4\})$, or two, $(\{-1,0\},\ldots,\{3,4\})$.

DEFINITION 2.4 (uniformly layered DAG). *A layered DAG of a matrix chain product*

$$F_q' \times \cdots \times F_1' = F' \in \mathbb{R}^{n\times n}$$

is called uniformly layered DAG *if* $F_i' \in \mathbb{R}^{n\times n}$ *for* $i = 1,\ldots,q$.

All partitions of the vertex set that correspond to layers as in Definition 2.3 need to have the same cardinality n. In particular, we require $n = m$, which is satisfied for

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/01/24 to 134.100.220.139 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

the residuals of systems of n nonlinear equations in n unknowns that were introduced in section 1. Obviously, the DAG in Figure 2(a) is uniformly layered with $n=2$.

DEFINITION 2.5 (invertible DAG). *We refer to a DAG of a differentiable program with invertible Jacobian $F' \in \mathbb{R}^{n\times n}$ as an* invertible DAG. *The* corresponding Newton step *is defined as $(F')^{-1}\cdot y$ for given $y \in \mathbb{R}^n$.*

According to [17] all differentiable programs can conceptually be represented as uniformly layered DAGs by letting elemental functions operate on the entire memory space $\{v_j : j = 1-n,\ldots,p+m\}$. The resulting elemental Jacobians turn out to be not invertible due to identically vanishing columns, which prevents naive application of this approach. Nevertheless, this perspective might offer additional insight into problem structure and combinatorics, which is why we take a closer look at it in section 5.

Let us relate the formalism developed so far to a simple, yet realistic differentiable numerical simulation. The initial-boundary value problem for the 1D diffusion equation

$$\frac{dy}{dt} = \nu \cdot \frac{d^2y}{dx^2}, \quad x \in (0,1),\ t \in (0,1]$$

defines $y = y(x,t)$ in the interior of its domain at final time $t=1$ for given diffusion coefficient $\nu > 0$, initial ($y(x,0) = y^0$), and (e.g., for simplicity, Dirichlet-type) boundary ($y(0,t) = y_0$, $y(1,t) = y_n$) conditions. Discretization in time using forward finite differences with $\Delta t = \frac{1}{q}$ and in space using central finite differences with $\Delta x = \frac{1}{n-1}$ yields

$$\frac{y_i^{j+1} - y_i^j}{\Delta t} = \nu \cdot \frac{y_{i+1}^j - 2\cdot y_i^j + y_{i-1}^j}{\Delta x^2}$$

and, hence, the explicit Euler integration scheme

$$y_i^{j+1} = y_i^j + \Delta t \cdot \nu \cdot \frac{y_{i+1}^j - 2\cdot y_i^j + y_{i-1}^j}{\Delta x^2} = y_i^j + \frac{\nu}{q}\cdot (n-1)^2 \cdot (y_{i+1}^j - 2\cdot y_i^j + y_{i-1}^j)$$

for $j = 0,\ldots,q-1$ and $i = 1,\ldots,n-1$. Implementations in arbitrary programming languages follow immediately. Their DAGs turn out to be uniformly layered as illustrated in Figure 1(a) for $n=6$ and $q=3$. Constant boundary values are added as dotted vertices (not actually belonging to the DAG). The chain rule of differentiation yields the dense Jacobian of the final state with respect to the initial condition as

$$\frac{dy^3}{dy^0} = \frac{dy^3}{dy^2}\cdot\frac{dy^2}{dy^1}\cdot\frac{dy^1}{dy^0}$$

with tridiagonal invertible elemental Jacobians

$$\frac{dy^i}{dy^{i-1}} = \begin{pmatrix} 1-2\cdot\alpha & \alpha & 0 & 0 & 0 \\ \alpha & 1-2\cdot\alpha & \alpha & 0 & 0 \\ 0 & \alpha & 1-2\cdot\alpha & \alpha & 0 \\ 0 & 0 & \alpha & 1-2\cdot\alpha & \alpha \\ 0 & 0 & 0 & \alpha & 1-2\cdot\alpha \end{pmatrix}$$

for $i = 1,\ldots,3$, and where $\alpha = \frac{\nu}{q}\cdot(n-1)^2$. The cost of computing $y^i \in \mathbb{R}^5$ can be estimated as $\mathcal{O}(3n) = \mathcal{O}((2\mu+1)n)$ for $\mu = 1$ and where $2\mu+1$ denotes the bandwidth of

Downloaded 07/01/24 to 134.100.220.139 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/01/24 to 134.100.220.139 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

the elemental Jacobians. Each scalar entry requires evaluation of three floating-point operations as illustrated by the DAG on the right of Figure 1 visualizing the (three-point) *stencil* of the finite difference discretization. Linearity of all elemental functions makes their local Jacobians independent of the values of y^{i-1}. This observation can naturally be exploited for the efficient evaluation of the single exact Newton step. As such, this very simple example is merely a structural illustration for a variety of other, generally nonlinear scenarios.

Adjoint AD can be implemented by using operator and function overloading in suitable programming languages such as C++ [15, 19, 32]. The given implementation of F as a differentiable program is run in overloaded arithmetic to augment the computation of the function value with the recording of the corresponding DAG (also referred to as *tape*). For example, for q = 2 the differentiable C++ program

```
for (int i=0;i<q;i++) {
  x[i%2]*=x[(i+1)%2];
  x[(i+1)%2]=sin(x[(i+1)%2]);
}
```

yields the DAG in Figure 2(a), where $\partial_{1,-1} = \mathrm{x}[1]$, $\partial_{3,1} = \cos(\mathrm{x}[0])$, and so forth. The Jacobian

$$F' = F_2' \cdot F_1' = \begin{pmatrix} \partial_{3,1} & 0 \\ \partial_{4,1} & \partial_{4,2} \end{pmatrix} \begin{pmatrix} \partial_{1,-1} & \partial_{1,0} \\ 0 & \partial_{2,0} \end{pmatrix} = \begin{pmatrix} \partial_{3,1}\partial_{1,-1} & \partial_{3,1}\partial_{1,0} \\ \partial_{4,1}\partial_{1,-1} & \partial_{4,1}\partial_{1,0} + \partial_{4,2}\partial_{2,0} \end{pmatrix}$$

is represented by the bipartite DAG in Figure 2(b). Both options for evaluating the adjoint

$$\left(\begin{pmatrix} \partial_{3,1} & 0 \\ \partial_{4,1} & \partial_{4,2} \end{pmatrix} \begin{pmatrix} \partial_{1,-1} & \partial_{1,0} \\ 0 & \partial_{2,0} \end{pmatrix} \right)^T \begin{pmatrix} \bar{y}_0 \\ \bar{y}_1 \end{pmatrix} = \begin{pmatrix} \partial_{1,-1} & 0 \\ \partial_{1,0} & \partial_{2,0} \end{pmatrix} \left(\begin{pmatrix} \partial_{3,1} & \partial_{4,1} \\ 0 & \partial_{4,2} \end{pmatrix} \begin{pmatrix} \bar{y}_0 \\ \bar{y}_1 \end{pmatrix} \right)$$

(with $\bar{y} = (1\ 0)^T$ and $\bar{y} = (0\ 1)^T$ giving the rows of the Jacobian) yield the same result. Note that the evaluation of the expression on the left-hand side requires 12 fused multiply-add floating-point operations (fma); $\mathcal{O}(qn^3)$; here $q = n = 2$), while the right-hand-side expression takes only 8 fma ($\mathcal{O}(qn^2)$). The adjoint can be visualized as in Figure 2(c). The same effect due to associativity of the chain rule of differentiation (equivalently, of matrix multiplication)[2] is exploited by *backpropagation* in the context of training of artificial neural networks [13].

A discussion of the numerous aspects of AD and of its implementation are beyond the scope of this article. The interested reader is referred to [17] for further information on the subject. Moreover, the AD community's Web portal `www.autodiff.org` contains a comprehensive bibliography as well as links to various AD research and tool development efforts.

2.2. Lesson. How can Newton's method benefit from lessons learned in adjoint AD? Working toward an answer to this question, the given implementation of F as a differentiable program is assumed to be composed of elemental functions

$$F_i : \mathbb{R}^n \to \mathbb{R}^n : x^i = F_i(x^{i-1})$$

for $i = 1, \ldots, q$ as

$$y = x^q = F(x^0) = F_q(\ldots F_1(x^0)\ldots), \tag{2.4}$$

[2] assuming infinite precision arithmetic.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/01/24 to 134.100.220.139 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

where $x^0 = x$. Application of the chain rule of differentiation to (2.4) yields

$$F'(x^0) = F'_q(x^{q-1}) \times \cdots \times F'_1(x^0),$$

implying

$$\begin{aligned} \Delta x &= -F'(x)^{-1} \cdot y \\ &= -(F'_q(x^{q-1}) \times \cdots \times F'_1(x^0))^{-1} \cdot y \\ &= -F'_1(x^0)^{-1} \times \cdots \times F'_q(x^{q-1})^{-1} \cdot y\,. \end{aligned} \tag{2.5}$$

$F'(x)$ needs to be invertible. Invertibility of all F'_i follows immediately. Their (without loss of generality) LU factorization [7] yields

$$F_i \equiv F_i(x^{i-1}) = L_i(x^{i-1}) \cdot U_i(x^{i-1}) \equiv L_i \cdot U_i \tag{2.6}$$

with lower triangular L_i and upper triangular U_i. In-place factorization yields $L_i - I_n + U_i$ at a computational cost of $\mathcal{O}(n^3)$. The matrix $I_n \in \mathbb{R}^{n \times n}$ denotes the identity in $\mathbb{R}^n$. From (2.5) it follows that

$$\begin{aligned} \Delta x &= -U_1^{-1} \cdot L_1^{-1} \times \cdots \times U_q^{-1} \cdot L_q^{-1} \cdot y \\ &= -U_1^{-1} \cdot \left(L_1^{-1} \times \cdots \times \left(U_q^{-1} \cdot \left(L_q^{-1} \cdot y\right)\right) \ldots\right), \end{aligned} \tag{2.7}$$

yielding $2q$ linear systems to be solved as efficiently as possible in order to undercut the computational cost of the standard Newton method. The new method is matrix-free in the sense that F' is not accumulated explicitly. For the example in Figure 2, the standard *"accumulate first, then factorize"* approach yields the Newton step

$$\underbrace{\underbrace{\underbrace{\left(\begin{pmatrix} \partial_{3,1} & 0 \\ \partial_{4,1} & \partial_{4,2} \end{pmatrix} \begin{pmatrix} \partial_{1,-1} & \partial_{1,0} \\ 0 & \partial_{2,0} \end{pmatrix}\right)^{-1}}_{\mathcal{O}(qn^3) \,\hat{=}\, 5} \begin{pmatrix} y_0 \\ y_1 \end{pmatrix}}_{\mathcal{O}(n^3) \,\hat{=}\, 3+3+1=7}}_{\Sigma=12}$$

at the expense of 12 fma. The alternative *"factorize first, then accumulate"* method

$$\underbrace{\begin{pmatrix} \partial_{1,-1} & \partial_{1,0} \\ 0 & \partial_{2,0} \end{pmatrix}^{-1} \underbrace{\left(\begin{pmatrix} \partial_{3,1} & 0 \\ \partial_{4,1} & \partial_{4,2} \end{pmatrix}^{-1} \begin{pmatrix} y_0 \\ y_1 \end{pmatrix}\right)}_{O(n^2) \,\hat{=}\, 3}}_{O(qn^2) \,\hat{=}\, 6}$$

performs the same task using only 6 fma. Admittedly, this example is extreme in the sense that all elemental Jacobians are already triangular. Factorization does not need to be performed at all. The entire computational cost is due to accumulation by forward and backward substitutions. More realistically, the individual factors need to be transformed into triangular form first. Ideally, the exploitation of sparsity of the elemental Jacobians is expected to keep the corresponding additional effort low. However, the usual challenges faced in the context of direct methods for sparse linear algebra [7] need to be addressed. A representative special case is discussed in the next section. It illustrates the potential of the new method and serves as motivation for further generalization in section 4 with the aim to set the stage for applicability to a wider range of practically relevant problems.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/01/24 to 134.100.220.139 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

3. Banded elemental Jacobians. Let all F_i' have (maximum) bandwidth $2\mu+1$ (μ off-diagonals) with $\mu \ll n$. The previously discussed initial-boundary value problem for the 1D diffusion equation provides illustration for $\mu = 1$. In-place LU factorization of the F_i' as in (2.6) yields $L_i - I_n + U_i$ with the same bandwidth at a computational cost of $\mathcal{O}(\mu^2 n)$. The resulting $2q$ triangular linear systems can be solved efficiently with a cost of $\mathcal{O}(\mu n)$ by simple substitution, respectively. The total computational cost of the matrix-free exact Newton method can hence be estimated as $\mathcal{O}(q\mu^2 n)$.

For F' to become dense we require $q \geq \underline{q} \equiv (n-\mu-1)/\mu$. For $q \gg \underline{q}$, the cost of computing F' can be estimated as $\mathcal{O}(q\mu n^2)$. A matrix chain product of length q needs to be evaluated, where one factor has bandwidth $2\mu+1$ and the other factor becomes dense for $q \geq \underline{q}$. Interpretation of the tape (DAG) in adjoint AD yields a similar approach. Superiority of the matrix-free exact Newton method follows immediately for $\mu \ll n$ as $\mathcal{O}(q\mu^2 n) < \mathcal{O}(q\mu n^2)$.

In most real-world scenarios, the accumulation of F' is likely to dominate the overall computational cost, yielding a reduction of the computational cost of the matrix-free exact Newton method over the cost of the standard Newton method by $\mathcal{O}(\frac{n}{\mu})$. Nevertheless, let the computational cost of a Newton step be dominated by the solution of the linear Newton system. Superiority of the matrix-free exact Newton method requires $\mathcal{O}(q\mu^2 n) < \mathcal{O}(n^3)$, which will only be violated for $q > n$ assuming $n = \mathcal{O}(\mu^2)$.

Let all F_i' be tridiagonal as in the 1D diffusion example. In-place LU factorization using the Thomas algorithm [34] yields the tridiagonal $L_i - I_n - U_i$ at a computational cost of $\mathcal{O}(n)$. The $2q$ linear systems in (2.7) can be solved with a cost of the same order, respectively. The total computational cost of the matrix-free exact Newton method adds up to $\mathcal{O}(qn) < \max(\mathcal{O}(qn^2), \mathcal{O}(n^3))$. The speedup of $\mathcal{O}(\frac{n}{\mu}) = \mathcal{O}(n)$ is illustrated in Figure 3, showing the results of experiments for a single Newton step and $\frac{q}{n} = 0.5, 4$ with $0 < n \leq 10^3$. Similar plots are obtained for $\frac{q}{n} = 1, 2$. The *"factorize first, then accumulate"* approach implements the new matrix-free exact Newton method. It yields the bottom line in both subfigures of Figure 3. The *"accumulate first, then factorize"* approach uses OpenBLAS' `gbmv` method (`www.openblas.net`) for multiplying banded matrices with dense vectors. It exceeds the computational cost of the new method by roughly a factor of n. Our search for a dedicated method for multiplying banded matrices turned out unsuccessful. We would expect such an algorithm to be more efficient for $q \ll n$. Our basic reference implementation did not outperform `gbmv` though. Little effort went into its optimization, as it would not be our method of choice for the more common scenario of $q \gg n$, or in other words, if the number of operations far exceeds the input dimension of the program.

4. Toward generalization. DAGs recorded by adjoint AD are typically neither layered nor uniform. According to Definition 2.3, the term "layered" refers to a decomposition of the DAG into a sequence of bipartite sub-DAGs representing the elemental Jacobians F_i'. For a DAG to be uniformly layered, all layers must consist of the same number (n) of vertices as formally stated in Definition 2.4.

In the following we present ideas on how to make invertible DAGs uniformly layered. We formalize the corresponding methods and prove their numerical correctness, respectively. All proofs rely on the following formulation of the chain rule of differentiation.

LEMMA 4.1 (chain rule of differentiation on DAG). *Let G be the DAG of a differentiable program $F : \mathbb{R}^n \to \mathbb{R}^m$ as in Definition 2.2 with Jacobian*

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

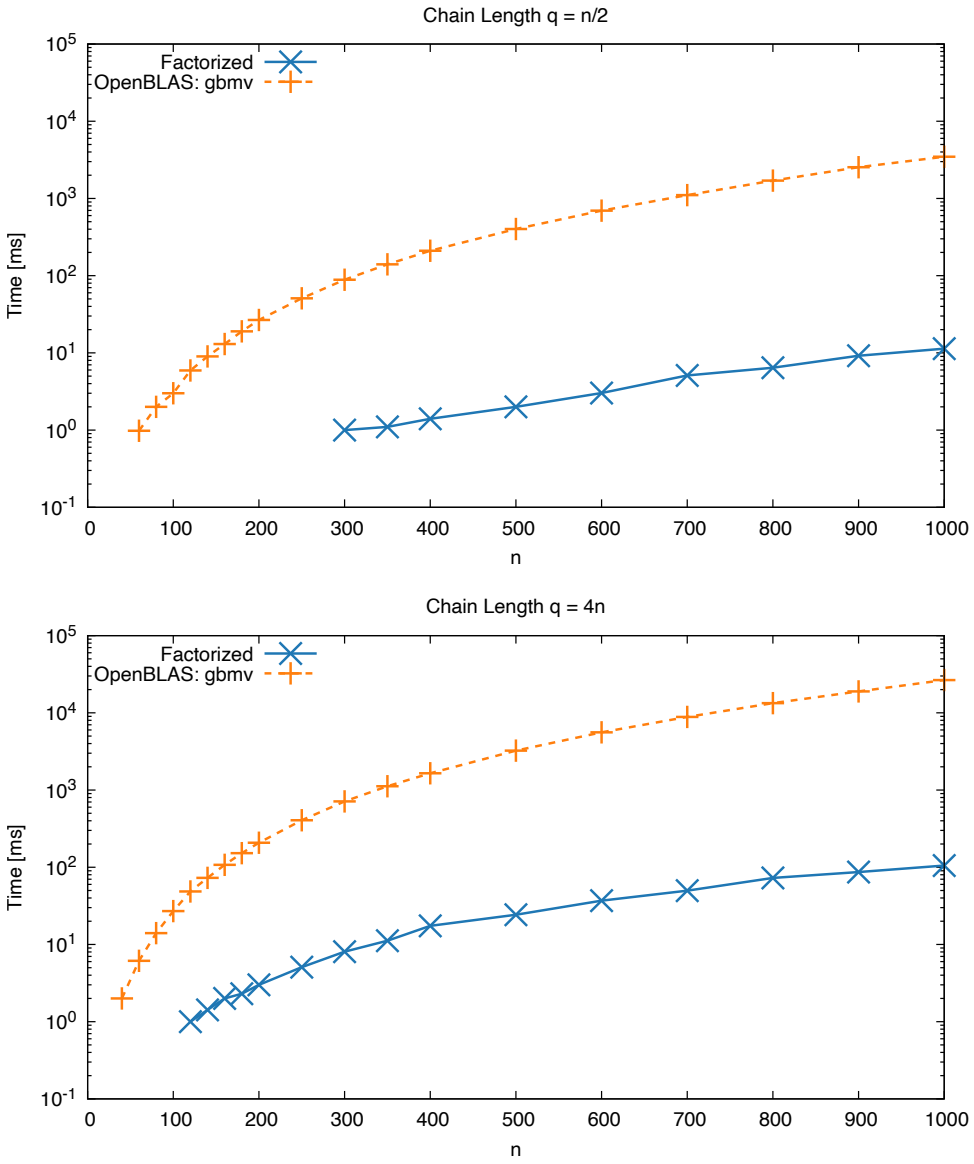

FIG. 3. *Tridiagonal elemental Jacobians.*

$$F' \equiv \left(\frac{dy_k}{dx_i}\right)_{i=1,\ldots,n}^{k=1,\ldots,m} = \left(\frac{dv_{p+k}}{dv_{i-n}}\right)_{i=1,\ldots,n}^{k=1,\ldots,m} \in \mathbb{R}^{m\times n}.$$

Then

$$\frac{dv_{p+k}}{dv_{i-n}} = \sum_{(i-n,\ldots,p+k)} \prod_{(s,t)\in(i-n,\ldots,p+k)} \partial_{t,s} \tag{4.1}$$

for $i = 1,\ldots,n$ *and* $k = 1,\ldots,m$.

Proof. See [1]. ∎

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/01/24 to 134.100.220.139 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

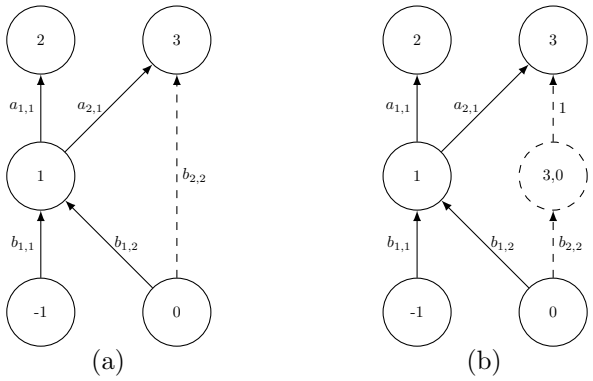

FIG. 4. *Edge splitting.*

An illustrative explanation of each method is followed by its formalization and the proof of numerical correctness.

4.1. Edge splitting. Consider a minor modification of the sample program from section 2 as.

```
for (int i=0;i<q;i++) {
  x[i%2]*=x[(i+1)%2];
  if (!(i%2)) x[(i+1)%2]=sin(x[(i+1)%2]);
}
```

For q = 2, the DAG in Figure 4(a) is recorded. The edge labeled with the partial derivative $b_{2,2}$ (=x[0] for i = 1) spans two layers, making the DAG not layered. We aim to transform it into a layered tripartite DAG representing the product $A \cdot B$ of the elemental Jacobians A and B, where $A \equiv (a_{j,i})$ as well as $B \equiv (b_{j,i})$. The $\partial_{j',i'}$ are replaced by $a_{j,i}$ or $b_{j,i}$. This slight modification in the notation is expected to make the upcoming examples easier to follow.

Edge splitting makes the DAG layered by inserting $l - 1 = j - i - 1$ dummy vertices for all edges connecting vertices in layers i and j (here, $l = 2 - 0 = 2$ yields one additional vertex labeled $3, 0$ to mark it as split vertex of edge $(0, 3)$). One of the resulting new edges keeps the original label while the others are labeled with ones as in Figure 4(b) ($l - 1 = 1$ dummy vertex). The Jacobian F' turns out to be invariant under edge splitting as an immediate consequence of the chain rule of differentiation. For example,

$$F' = \begin{pmatrix} a_{1,1} & 0 \\ a_{2,1} & 1 \end{pmatrix} \begin{pmatrix} b_{1,1} & b_{1,2} \\ 0 & b_{2,2} \end{pmatrix} = \begin{pmatrix} a_{1,1}b_{1,1} & a_{1,1}b_{1,2} \\ a_{2,1}b_{1,1} & a_{2,1}b_{1,2} + b_{2,2} \end{pmatrix}.$$

Invertibility of F' implies invertibility of all elemental Jacobians in the resulting uniformly layered DAG.

DEFINITION 4.2 (edge splitting). *Let $G = (V, E)$ be a DAG as in Definition* 2.2. *An* edge (i, j) *with label λ is* split *by replacing it with two new edges (i, k) and (k, j), $k \notin V$, and labeling them with $\partial_{k,i} = \lambda$ and $\partial_{j,k} = 1$ (or vice versa), respectively.*

Consequences of the various choices for labeling the new edges are the subject of ongoing investigations. Both alternatives are regarded as equivalent for the purpose of the upcoming discussions. Note that one might also be tempted to replace (i, j) with edges (i, k) and (k, j) carrying labels $\partial_{k,i} = \partial_{j,k} = \sqrt{\lambda}$ or any other feasible

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/01/24 to 134.100.220.139 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Downloaded 07/01/24 to 134.100.220.139 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

factorization of λ. We consider benefits from similar strategies as unlikely since unit edge labels are easier to exploit in the context of structural modifications of the DAG, including preaccumulation to be discussed in section 4.2.

LEMMA 4.3. *Let a DAG G induce an invertible Jacobian $F' \in \mathbb{R}^{n\times n}$ under the chain rule of differentiation as in Lemma* 4.1*.*

1. *The Jacobian is invariant under edge splitting as in Definition* 4.2*.*
2. *If repeated edge splitting yields a uniformly layered DAG as in Definition* 2.4*, then the elemental Jacobians of each layer are invertible.*

Proof.

1. Equation (4.1) is invariant under edge splitting.
2. Let the uniformly layered DAG have depth q. Equation (4.1) yields

$$F' = F'_q \times \cdots \times F'_1$$

and hence

$$(F')^{-1} = (F'_q \times \cdots \times F'_1)^{-1} = (F'_1)^{-1} \times \cdots \times (F'_q)^{-1},$$

implying invertibility of all F'_j, $j = 1, \ldots, q$. ∎

Obviously, edge splitting terminates as soon as the DAG becomes layered.

4.2. Preaccumulation. Making DAGs layered by edge splitting turned out to be rather straightforward. Unfortunately, the resulting layered DAGs are rarely uniform. Moreover, invertibility of the Jacobian implies that the sizes of all layers are at least equal to n. We propose *preaccumulation* of elemental sub-Jacobians followed by the decomposition of the resulting bipartite sub-DAGs into layered tripartite sub-DAGs as a method for making such DAGs uniformly layered.

Figure 5(a) shows a nonuniformly layered tripartite DAG G with $n = m = 3$ sources, respectively, sinks, and $p = 4$ intermediate vertices. Connected components of the bipartite sub-DAGs induce pure layered tripartite sub-DAGs of G as formally described in Definition 4.4. For example, the bipartite component spanned by the three vertices $1, 2, 5$ induces a pure layered tripartite sub-DAG $\hat{G}$ of G including -2 and -1. (Similarly, the bipartite component spanned by $-2, 1, 3$ yields the pure layered tripartite sub-DAG of G including 5 and 6.) Application of the chain rule of differentiation as in Lemma 4.1 to $\hat{G}$ yields an elemental Jacobian $A \in \mathbb{R}^{m\times n}$ (here, $A \in \mathbb{R}^{1\times 2}$ is the gradient of v_5 with respect to v_{-2} and v_{-1}).

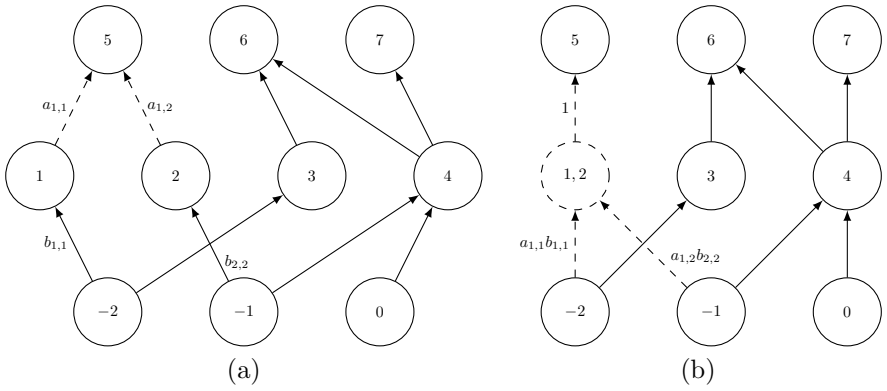

FIG. 5. *Preaccumulation.*

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/01/24 to 134.100.220.139 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Replacing $\hat{G}$ with the bipartite DAG of A violates the requirement for G to be layered. Decomposition of A as $A = I_m \cdot A$ (or $A = A \cdot I_n$) takes care of this violation by substitution of a modified pure layered tripartite sub-DAG $\tilde{G}$ as shown in Figure 5(b). The number of intermediate vertices in $\tilde{G}$ is equal to $\tilde{p} = \min(n, m)$, implying a reduction of the number of intermediate vertices in G if $\tilde{p} < p$. Figure 5 yields $1 = \tilde{p} < p = 2$, making the resulting DAG uniformly layered.

Ultimately, preaccumulation always yields the uniformly layered DAG preserving all intermediate layers of size n from the original DAG. In the worst case we get the bipartite DAG of F'. Our objective is to apply preaccumulation selectively as illustrated above. The resulting uniformly layered DAG is meant to preserve the sparsity of all elemental Jacobians as well as possible.

Decomposition of a bipartite graph (similarly, DAG) $G = (V, E)$ into its connected components turns out to be straightforward. Each vertex belongs to exactly one connected component. Repeated (e.g., depth-first) searches starting from unassigned vertices yield a decomposition into connected components at the cost of $\mathcal{O}(|V| + |E|)$.

The following definitions formalize the above followed by the proof of the numerical correctness of preaccumulation in combination with bipartite DAG splitting.

DEFINITION 4.4 (pure layered tripartite sub-DAG). *A layered tripartite sub-DAG $G = (V, E)$, $V = (X, Z, Y)$, as in Definition* 2.3*, is* pure *if $P(Z) = X$ and $S(Z) = Y$.*

DEFINITION 4.5 (preaccumulation). *Let G be a pure layered tripartite sub-DAG as in Definition* 4.4 *representing an elemental Jacobian product $F' = F'_2 \cdot F'_1$.* Preaccumulation *replaces G by the bipartite sub-DAG representing F'.*

LEMMA 4.6. *Let a DAG G induce an invertible Jacobian $F' \in \mathbb{R}^{n \times n}$ under the chain rule of differentiation as in Lemma* 4.1*.*

1. *The Jacobian is invariant under preaccumulation as in Definition* 4.5*.*
2. *Preaccumulation terminates.*

Proof.

1. Equation (4.1) is invariant under preaccumulation.
2. The cumulative sum of all paths connecting sources with sinks decreases monotonically under preaccumulation. Hence, full preaccumulation yields the bipartite DAG that represents F'. ∎

DEFINITION 4.7 (bipartite DAG splitting). *A* bipartite DAG *G representing an elemental Jacobian $F'_j \in \mathbb{R}^{m \times n}$ is* split *by replacing it with a layered tripartite DAG representing the matrix products $F' \cdot I_n$ if $n \leq m$ or $I_m \cdot F'$ otherwise.*

The condition $n \leq m$ could be omitted if numerical correctness was our sole objective. However, aiming for maximum sparsity we keep the number of newly generated edges as low as possible. Further implications of the two choices are the subject of ongoing research.

LEMMA 4.8. *Let a DAG G induce an invertible Jacobian $F' \in \mathbb{R}^{n \times n}$ under the chain rule of differentiation as in Lemma* 4.1*.*

1. *The Jacobian is invariant under bipartite DAG splitting as in Definition* 4.7*.*
2. *If repeated bipartite DAG splitting yields a uniformly layered DAG as in Definition* 2.4*, then the elemental Jacobians of each layer are invertible.*

Proof.

1. Equation (4.1) is invariant under bipartite DAG splitting.
2. Let the uniformly layered DAG have depth q. Equation (4.1) yields

$$F' = F'_q \times \cdots \times F'_1$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/01/24 to 134.100.220.139 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

and hence

$$(F')^{-1} = (F'_1)^{-1} \times \cdots \times (F'_q)^{-1},$$

implying invertibility of all F'_j, $j = 1, \ldots, q$. ∎

4.3. Combinatorics. We aim for a transformation of the given DAG into uniformly layered shape with the lowest possible number of edge splits and preaccumulations. Put differently, the fill-in (newly introduced edges) due to the necessary structural modifications including the subsequent factorization of all sparse elemental Jacobians should be minimized. This purely structural problem formulation turns out to be computationally intractable due to the known hardness of Directed Elimination Ordering [31]. Greedy heuristics for the selection of targets for preaccumulation according to the resulting reduction in the width of the DAG are easy to derive. It remains unclear if less naive approaches can lead to lower fill-in. Moreover, the interplay of these choices with known heuristics for minimizing fill-in [23] due to sparse matrix factorizations remains to be investigated.

An alternative perspective on the costs of the matrix-free exact Newton method takes potential algebraic dependences (e.g., equality) among scalar entries of the elemental Jacobians into account. The corresponding Matrix-Free Exact Newton Step problem to be formulated in its decision version next turns out to be computationally intractable.

Problem 1 (Matrix-Free Exact Newton Step).
instance: *An invertible DAG and an integer* $K \geq 0$.
question: *Can the corresponding Newton step be evaluated with at most* K *flops*[3] *?*

Theorem 4.9. Matrix-Free Exact Newton Step *is NP-complete.*

Proof. The proof of NP-completeness of the Adjoint Computation problem presented in [25] reduces Ensemble Computation [8] to uniformly layered DAGs representing matrix chain products over diagonal Jacobians $\in \mathbb{R}^{n \times n}$ as

$$F' = F'_q \times \cdots \times F'_1 \in \mathbb{R}^{n \times n}\,.$$

A corresponding Newton step becomes equal to

$$(F')^{-1} \cdot y = (F'_q \times \cdots \times F'_1)^{-1} \cdot y = (F'_1)^{-1} \times \cdots \times (F'_q)^{-1} \cdot y\,.$$

The "accumulate first, then factorize" method turns out to be superior as F' remains a diagonal matrix. Its inversion adds a constant offset of n (reciprocals) to the flop count of any given instance of Adjoint Computation. A solution for Matrix-Free Exact Newton Step would hence solve Adjoint Computation implying NP-hardness of the former. Moreover, a proposed solution is easily validated efficiently by counting the at most $(q+1)n$ flops. ∎

The proof of NP-completeness of the Matrix-Free Exact Newton Step problem relies entirely on algebraic dependences (equality in particular) among the elemental partial derivatives. The structure of the underlying DAGs turns out to be trivial. On the other hand, the methods proposed for making DAGs uniformly layered are motivated by purely structural reasoning about the DAGs. All elemental partial derivatives were assumed to be mutually independent. At this stage, the reduction used in the proof of Theorem 4.9 has no consequences on algorithms for the

[3] floating-point operations.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/01/24 to 134.100.220.139 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

approximate solution of instances of Matrix-Free Exact Newton Step. In fact, similar statements apply to the related proof presented in [25]. Future research is expected to fill this gap.

5. Extended Jacobian chains. According to [17] all differentiable programs can conceptually be represented as uniformly layered DAGs. All elemental functions are simply regarded as operations on the entire memory space

$$\{v_j : j = 1-n, \dots, p+m\}$$

yielding potentially extremely sparse elemental Jacobians. For example, the Jacobian $F' \in \mathbb{R}^{2\times 2}$ in Figure 2(b) can be computed as

$$F' = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & \partial_{4,1} & \partial_{4,2} & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & \partial_{3,1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$
$$\cdot \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & \partial_{2,0} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ \partial_{1,-1} & \partial_{1,0} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} .$$

The elemental (6×6)-Jacobians induce a uniformly layered DAG. However, they are not invertible due to identically vanishing columns. *Pruning* as proposed in [16] propagates the zero columns of the left-most factor (extracting the last two rows from F') and the zero rows of the right-most factor (extracting the first two columns from F') through the elemental Jacobian chain product yielding the pruned elemental Jacobian chain product

$$F' = \begin{pmatrix} 0 & 0 & 1 \\ \partial_{4,1} & \partial_{4,2} & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ \partial_{3,1} & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ \partial_{2,0} & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ \partial_{1,-1} & \partial_{1,0} \end{pmatrix} .$$

The corresponding DAG is no longer uniformly layered. Multiplication of two left-most factors at vanishing cost[4] yields

$$F'_2 = \begin{pmatrix} \partial_{3,1} & 0 \\ \partial_{4,1} & \partial_{4,2} \end{pmatrix}$$

in the original example from section 2.1. Similarly, multiplication of two right-most factors at vanishing cost yields

$$F'_1 = \begin{pmatrix} \partial_{1,-1} & \partial_{1,0} \\ 0 & \partial_{2,0} \end{pmatrix}$$

and hence the original elemental Jacobian chain product $F' = F'_2 \cdot F'_1$ from section 2.1. This perspective on the Matrix-Free Exact Newton Step problem does not seem

[4]Multiplications by structural zeros and ones can typically be avoided in AD.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 07/01/24 to 134.100.220.139 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

to offer any additional insight for the given simple problem. Nevertheless it should be included into investigations of potential combinatorial optimization methods. A combination of pruning of subchains with pivoting resulting in the permutation of columns and/or rows to reduce bandwidth may result in sparser uniformly layered invertible DAGs.

6. Conclusion. The new matrix-free exact Newton method promises a significant reduction in the computational cost of solutions for systems of nonlinear equations. This claim is supported by run time measurements for problems with tridiagonal elemental Jacobians. Further challenges need to be addressed for elemental Jacobians with irregular sparsity patterns as well as for computationally expensive residuals. Conditioning ($\kappa(F') = \|(F')^{-1}\|\|F'\|$) must not be ignored as $\kappa(F') = \kappa(\prod_{i=0}^{q} F_i') \leq \prod_{i=0}^{q} \kappa(F_i')$. Implications for relevant applications are the subject of ongoing investigations.

The matrix-free exact Newton method relies on the reversal of the DAG. For very large problems, the size of the DAG may exceed the available memory resources. The combinatorial DAG Reversal problem asks for a distribution of the available storage such that the overall computational cost is minimized. It is known to be NP-complete [26], as is the related Call Tree Reversal problem [24]. Checkpointing methods [14, 33, 35] offer solutions for a variety of relevant special cases.

Factorization of elemental Jacobians with irregular sparsity patterns yields several well-known combinatorial problems in sparse linear algebra such as Bandwidth [30] or Directed Elimination Ordering [31]. Fill-in needs to be kept low for our method to outperform other state-of-the-art solutions. A rich set of results from past and ongoing efforts in this highly active area of research can be built on.

Tapes recorded by software tools for AD that rely on operator and function overloading (e.g., in C++) provide the ideal basis for implementing the exact matrix-free Newton method. Applicability to a wide range of computationally expensive residuals could be enabled. Corresponding activities including the technically nontrivial transformation of tapes into uniformly layered form are ongoing.

Another class of potential targets are surrogates for computationally expensive nonlinear residuals obtained by machine learning. While the training of such models can be challenging, the inversion of their Jacobian can be guaranteed to benefit from the new method. Ongoing research aims to explore a potential extension to data-driven models with layers exhibiting suitable sparsity patterns, e.g., triangular elemental Jacobians. Further investigations target Sobolev training [4] and pruning [21] based on first- and/or second-directional derivatives obtained by applying AD to the given residuals.

Acknowledgments. We thank the anonymous referees for their productive comments on the initial version of the manuscript. They triggered further related thoughts as well as helping to improve the overall presentation.

The experiments reported on in Figure 3 were performed by Gero Kauerauf as part of his M.Sc. thesis project at RWTH Aachen University.

REFERENCES

[1] W. Baur and V. Strassen, *The complexity of partial derivatives*, Theoret. Comput. Sci., 22 (1983), pp. 317–330.

[2] C. Bischof, M. Bücker, P. Hovland, U. Naumann, and J. Utke, eds., *Advances in Automatic Differentiation*, Lect. Notes Comput. Sci. Eng. 64, Springer, Cham, Switzerland, 2008.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

[3] C. Broyden, *The convergence of a class of Double-rank minimization algorithms* 1. *General considerations*, IMA J. Appl. Math., 6 (1970), pp. 76–90.
[4] W. Czarnecki, S. Osindero, M. Jaderberg, G. Swirszcz, and R. Pascanu, *Sobolev training for neural networks*, in Proceedings of 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA, 2017.
[5] R. Dembo and T. Steihaug, *Truncated Newton algorithms for large-scale unconstrained optimization*, Math. Program., 26 (1983), pp. 190–212.
[6] P. Deuflhard, *Newton Methods for Nonlinear Problems. Affine Invariance and Adaptive Algorithms*, Springer Ser. Comput. Math. 35, Springer International, Cham, Switzerland, 2004.
[7] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*, Clarendon Press, Oxford, 1986.
[8] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, 1st ed., A Series of Books in the Mathematical Sciences, W. H. Freeman, 1979.
[9] A. H. Gebremedhin, F. Manne, and A. Pothen, *What color is your Jacobian? Graph coloring for computing derivatives*, SIAM Rev., 47 (2005), pp. 629–705.
[10] J. Gilbert, *Automatic differentiation and iterative processes*, Optim. Methods Softw., 1 (1992), pp. 13–21.
[11] M. B. Giles, *Collected matrix derivative results for forward and reverse mode algorithmic differentiation*, in Advances in Automatic Differentiation, Lect. Notes Comput. Sci. Eng. 64, Springer, Cham, Switzerland, pp. 35–44.
[12] G. Golub and C. Van Loan, *Matrix Computations*, 3rd ed., The Johns Hopkins University Press, Baltimore, MD, 1996.
[13] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, Cambridge, MA, 2016, http://www.deeplearningbook.org.
[14] A. Griewank, *Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation*, Optim. Methods Softw., 1 (1992), pp. 35–54.
[15] A. Griewank, D. Juedes, and J. Utke, *Algorithm* 755*: ADOL-C: A package for the automatic differentiation of algorithms written in C/C++*, ACM Trans. Math. Software, 22 (1996), pp. 131–167.
[16] A. Griewank and U. Naumann, *Accumulating Jacobians as chained sparse matrix products*, Math. Program., 95 (2003), pp. 555–571.
[17] A. Griewank and A. Walther, *Evaluating Derivatives, Principles and Techniques of Algorithmic Differentiation*, 2nd ed., Other Titles in Applied Mathematics OT105, SIAM, Philadelphia, PA, 2008.
[18] M. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*, J. Res. Natl. Bur. Stand., 42 (1952), pp. 409–435.
[19] R. Hogan, *Fast reverse-mode automatic differentiation using expression templates in C++*, ACM Trans. Math. Software, 40 (2014), pp. 26:1–26:24.
[20] C. T. Kelley, *Solving Nonlinear Equations with Newton's Methods*, SIAM, Philadelphia, PA, 2003.
[21] N. Kichler, S. Afghan, and U. Naumann, *Towards Sobolev Pruning*, preprint, https://arxiv.org/abs/2312.03510, 2023.
[22] D. Knoll and D. Keyes, *Jacobian-free Newton-Krylov methods: A survey of* 548 *approaches and applications*, J. Comput. Phys., 193 (2004), pp. 357–397.
[23] H. Markowitz, *The elimination form of the inverse and its application*, Manag. Sci., 3 (1957), pp. 257–269.
[24] U. Naumann, *Call tree reversal is NP-complete*, in Advances in Automatic Differentiation, Lect. Notes Comput. Sci. Eng. 64, Springer, Cham, Switzerland, 2008, pp. 13–22.
[25] U. Naumann, *Optimal Jacobian accumulation is NP-complete*, Math. Program., 112 (2008), pp. 427–441.
[26] U. Naumann, *DAG reversal is NP-complete*, J. Discrete Algorithms, 7 (2009), pp. 402–410.
[27] U. Naumann, *The Art of Differentiating Computer Programs, An Introduction to Algorithmic Differentiation*, Software Enviro Tools, SIAM, Philadelphia, PA, 2012.
[28] U. Naumann, J. Lotz, K. Leppkes, and M. Towara, *Algorithmic differentiation of numerical methods: Tangent and adjoint solvers for parameterized systems of nonlinear equations*, ACM Trans. Math. Software, 41 (2015), 26.
[29] I. Newton, *Philosophiae Naturalis Principia Mathematica*, Colonia Allobrogum: Sumptibus Cl. et Ant. Philibert, 1760.
[30] C. H. Papadimitriou, *The NP-completeness of the bandwidth minimization problem*, Computing, 16 (1976), pp. 263–270.

Downloaded 07/01/24 to 134.100.220.139 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

[31] D. J. Rose and R. E. Tarjan, *Algorithmic aspects of vertex elimination on directed graphs*, SIAM J. Appl. Math., 34 (1978), pp. 176–197.
[32] M. Sagebaum, T. Albring, and N. Gauger, *High-performance derivative computations using CoDiPack*, ACM Trans. Math. Software, 45 (2019), pp. 1–26, https://doi.org/10.1145/3356900.
[33] P. Stumm and A. Walther, *Multistage approaches for optimal offline checkpointing*, SIAM J. Sci. Comput., 31 (2009), pp. 1946–1967.
[34] L. H. Thomas, *Elliptic Problems in Linear Differential Equations Over a Network*, Technical report, Watson Scientific Computing Laboratory, Columbia University, New York, 1949.
[35] Q. Wang, P. Moin, and G. Iaccarino, *Minimal repetition dynamic checkpointing algorithm for unsteady adjoint calculation*, SIAM J. Sci. Comput., 31 (2009), pp. 2549–2567.
[36] D. T. Whiteside, *The Mathematical Papers of Isaac Newton,* 7 *vols.*, Cambridge University Press, Cambridge, MA, 1967–1976.

Downloaded 07/01/24 to 134.100.220.139 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.