# Higher-order derivatives of the QR and of the real symmetric eigenvalue decomposition in forward and reverse mode algorithmic differentiation

Sebastian F. Walter[*1], Lutz Lehmann[1], and René Lamour[1]

[1]Department of Mathematics, Faculty of Mathematics and Natural Sciences II,,
Humboldt-Universität zu Berlin, Rudower Chaussee 25, Adlershof, 12489 Berlin,, mail address:
Unter den Linden 6, 10099 Berlin, Germany ,

October 1, 2010

## Abstract

We address the task of higher-order derivative evaluation of computer programs that contain QR decompositions and real symmetric eigenvalue decompositions. The approach is a combination of univariate Taylor polynomial arithmetic and matrix calculus in the (combined) forward/reverse mode of Algorithmic Differentiation (AD). Explicit algorithms are derived and presented in an accessible form. The approach is illustrated via examples.

## 1  Introduction and related work

This paper is concerned with the efficient evaluation of higher-order derivatives of functions $F : \mathbb{R}^N \to \mathbb{R}^M$ which are implemented as computer programs that contain numerical linear algebra functions like the QR or the real symmetric eigenvalue decomposition.

Traditionally, Algorithmic Differentiation (AD) tools like ADOL-C [GJM+99] or CppAD [Bel10] regard the functions defined in the C header file math.h as elementary functions. In the forward mode of AD, their approach to compute higher-order derivatives is to generalize from real arithmetic to univariate Taylor polynomial (UTP) arithmetic [GJM+99, GUW00, GW08]. For the reverse mode of AD, the program evaluation is traced and stored in a computational graph or on a sequential tape. During the so-called reverse sweep the stored intermediate values are retrieved and used to compute derivatives (c.f. Section 4).

---

[*]sebastian.walter@gmail.com

As explained in Section 3, the functions in math.h suffice since all computable functions are a concatenation of these functions. However, working only at the expression level has also disadvantages since no global knowledge of the function's structure can be used. A particularly important class of algorithms in science and engineering are numerical linear algebra (NLA) functions. Though NLA functions are typically locally smooth, their implementations often contain non-differentiable operations and program branches. If no special care is taken, this may result in incorrect computations of derivatives. Also, many NLA functions on $\mathbb{R}^{N \times N}$ matrices require $\mathcal{O}(N^3)$ arithmetic operations. Since during the reverse mode intermediate results are required, this would yield an $\mathcal{O}(N^3)$ memory requirement. Though it may be possible to adapt codes to yield reduced memory requirements, as for instance reported for the LU decomposition [Gri03], in practice it can be a cumbersome and error-prone process. Also one would like to reuse existing, high-performance implementations of NLA algorithms. Adding the NLA functions to the list of elementary functions circumvents this problem. This has been realized before [Büc02, BH96] and also UTP algorithms for some NLA functions (e.g. the solution of linear equations) have been implemented in software [Eri03].

The contribution of this paper is to provide explicit algorithms for UTP arithmetic applied to the QR decomposition and the real symmetric eigenvalue decomposition. Note that our approach to the real symmetric eigenvalue decomposition is similar to [AT98, vdAMM07] but our algorithmic result differs. In addition, we also treat the reverse mode of AD.

The document is structured as follows: In Section 2 we give two application examples for the algorithms presented in this document, followed by a brief review of the underlying computational model in Section 3. We shortly describe the basics of AD in Section 4 where we make use of the results from 3. In Section 5 we describe the general approach of NLA functions. After that, we apply the results from Section 4 to find extended functions for the QR and eigenvalue decomposition in Section 6 and 7 and also provide pullback algorithms that are necessary in in the reverse mode of AD. Finally, we present some numerical results in Section 8.

## 2   Application examples for the proposed algorithms

The purpose of this section is to show two application examples where higher-order derivatives of computer programs that contain the QR and the real symmetric eigenvalue decomposition are necessary.

### 2.1   Optimum experimental design

The goal in optimum experimental design (OED) is to minimize some cost function representing the size of the confidence region of parameters of interest. We consider here a popular formulation where the objective function $\Phi(C) \in \mathbb{R}$ depends on the covariance matrix $C \in \mathbb{R}^{N_p \times N_p}$ of a constrained parameter estimation problem, where the covariance matrix is computed by

$$ C \;=\; (\mathbf{I}, 0) \begin{pmatrix} J_1^T J_1 & J_2^T \\ J_2 & 0 \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{I} \\ 0 \end{pmatrix}, \tag{2.1} $$

and we assume that $J_1 \equiv J_1(q) \in \mathbb{R}^{N_m \times N_p}$, $J_2 \equiv J_2(q) \in \mathbb{R}^{N_r \times N_p}$, $\mathbf{I} \in \mathbb{R}^{N_p \times N_p}$ and $q \in \mathbb{R}^{N_q}$. The notation is motivated as follows: $p \in \mathbb{R}^{N_p}$ are model (pseudo-)parameters, $q \in \mathbb{R}^{N_q}$ are control variables and $J_1$ and $J_2$ are

Jacobians of the residuals resp. of the constraint function with respect to the parameters $p$. Typical choices for cost function $\Phi$ are the trace, the determinant or the maximum eigenvalue of the covariance matrix $C$. Though Eqn. (2.1) correctly describes the covariance matrix $C$, the actual algorithmic implementation is often a code like

$$C = Q_2^T \left( Q_2 J_1^T J_1 Q_2^T \right)^{-1} Q_2, \tag{2.2}$$

where $Q_2$ results from a QR-like decomposition of $J_2$, i.e. $J_2^T = (Q_1^T, Q_2^T)(L,0)^T$. The matrices $J_1$ and $J_2$ are assumed to satisfy the constraint qualification $\mathrm{rank}\,(J_2)) = N_\mathrm{r}$ and the condition $\mathrm{rank}\,(J) = N_p$, where $J = (J_1^T, J_2^T)^T$. The matrix $Q_2$ spans the nullspace of $J_2$. For a detailed discussion we refer to Körkel [Kör02] and to Bock and Kostina [KKSB07].

Newton-type optimization algorithms require at least the gradient $\nabla_q \Phi(q)$ of the objective function $\Phi$. To obtain good convergence near the local minimizer, it is often advantageous if exact second-order derivatives are available. Since the number of controls $N_q$ can be large, one would like to have the possibility to compute these derivatives in the reverse mode of AD. Robust objective functions are often formulated in a way that require third and even higher-order derivatives, so it is necessary to have algorithms that scale easily to arbitrary order.

Thus, this example requires the differentiation of the nullspace of a matrix, the matrix product, matrix inversion, the QR decomposition and the objective function evaluation, e.g. the eigenvalue decomposition.

## 2.2 Index determination of differential algebraic equations

Many dynamical problems in chemical engineering, rigid body mechanics, circuit simulation and control theory are described by Differential Algebraic Equations (DAEs) of the form

$$0 = f\left( \tfrac{\mathrm{d}}{\mathrm{d}x} d(y,x), y, x \right), \; x \in I = [a,b] \subset \mathbb{R}, \tag{2.3}$$

where $y : \mathbb{R} \to \mathbb{R}^m$ lives in suitable function space, $f : \mathbb{R}^n \times \mathbb{R}^m \times I \to \mathbb{R}^m$, $d : \mathbb{R}^m \times I \to \mathbb{R}^n$ are sufficiently smooth and typically $n$ is smaller than $m$.

Using higher–order derivatives of the functions in the DAE one can, in general, transform the DAE system into an ODE system of order one. The *differentiation index* is the highest derivative order required in this process, that is, derivatives of up to this order of the original equations are part of any solution of the DAE. The knowledge of the index allows to estimate the difficulty to solve the DAE.

There are many different index definitions. Here we consider the tractability index. To compute it, the DAE is linearized along a given function $\bar{y}(x)$ as

$$\underbrace{\tfrac{\partial}{\partial z} f(\bar{w}(x), \bar{y}(x), x)}_{=A(x) \in \mathbb{R}^{m \times n}} \tfrac{\mathrm{d}}{\mathrm{d}x} \left( \underbrace{\tfrac{\partial}{\partial y} d(\bar{y}(x), x)}_{=D(x) \in \mathbb{R}^{n \times m}} z(x) \right) + \underbrace{\tfrac{\partial}{\partial y} f(\bar{w}(x), \bar{y}(x), x)}_{=B(x) \in \mathbb{R}^{m \times m}} z(x) = \underbrace{-f(\bar{w}(x), \bar{y}(x), x)}_{q(x)}$$

with $\bar{w}(x) = \tfrac{\mathrm{d}}{\mathrm{d}x} d(\bar{y}(x), x)$. The coefficient functions $A = A(x)$, $D = D(x)$ and $B = B(x)$ give rise to a matrix sequence

$$
\begin{aligned}
G_0 &= AD, & B_0 &= B, \\
G_{i+1} &= G_i + B_i Q_i, & B_{i+1} &= B_i P_i - G_{i+1} D^- \tfrac{\mathrm{d}}{\mathrm{d}x} (D P_0 \cdots P_{i+1} D^-) D P_0 \cdots P_i,
\end{aligned}
\tag{2.4}
$$

where $Q_i$ describes a projector onto $\ker G_i$, $P_i = I - Q_i$ and $D^-$ is a generalized reflexive inverse of $D$. Now, the tractability index is the smallest number $\mu \in \mathbb{N}$ where $G_\mu$ is nonsingular. The projectors $Q_i$ can be determined mainly by use of a QR decomposition.

A QR decomposition of the potentially singular matrix $G \in \mathbb{R}^{M \times M}$ with rank $G = r$ results in

$$G\Pi = Q \begin{pmatrix} R_1 & R_2 \\ 0 & 0 \end{pmatrix},$$

where $\Pi$ describes a column permutation matrix, $Q$ an orthogonal matrix and $R_1 \in \mathbb{R}^{r \times r}$ an upper triangular matrix. Then a nullspace projector $Q_G$ onto $\ker G$ is given by

$$Q_G = \Pi \begin{pmatrix} 0 & -R_1^{-1} R_2 \\ 0 & I \end{pmatrix} \Pi^T.$$

The computation of $B_{i+1}$ via (2.4) needs the differentiation of $DP_0 \cdots P_{i+1} D^-$ with respect to $x$. Thus, higher–order derivatives of a function that contains the QR decomposition are necessary. For a in-depth discussion of index definition of DAEs see März [Mär02, Mär03].

## 3   Computational model

We consider functions

$$\begin{aligned} F : \mathbb{R}^N &\rightarrow \mathbb{R}^M \\ x &\mapsto y = F(x) \,, \end{aligned}$$

that can be described by the *three-part form*

$$\begin{aligned} v_{n-N} &= x_n & n &= 1, \ldots, N \\ v_l &= \phi_l(v_{j \prec l}) & l &= 1, \ldots, L \\ y_{M-m} &= v_{L-m} & m &= M-1, \ldots, 0 \,, \end{aligned}$$

where $\phi_l \in \{+, -, \cdot, /, \sin, \exp, \ldots\}$ are called *elementary functions*, $v_l$ are intermediate values and $v_{i \prec l}$ denote the tuples of input arguments of $\phi_l$. For instance the function $F : \mathbb{R}^2 \rightarrow \mathbb{R}$, $x \mapsto y = F(x) = \sin(x_1 + \cos(x_2) * x_1)$ is described by

| independent | $v_{-1}$ | $=$ | $x_1$ | $=$ | 3 |
| independent | $v_0$ | $=$ | $x_2$ | $=$ | 7 |
| | $v_1$ | $=$ | $\phi_1(v_0)$ | $=$ | $\cos(v_0)$ |
| | $v_2$ | $=$ | $\phi_2(v_1, v_{-1})$ | $=$ | $v_1 v_{-1}$ |
| | $v_3$ | $=$ | $\phi_3(v_{-1}, v_2)$ | $=$ | $v_{-1} + v_2$ |
| | $v_4$ | $=$ | $\phi_4(v_3)$ | $=$ | $\sin(v_3)$ |
| dependent | $y$ | $=$ | $v_4$ | | |

It shows a sequential representation of the computation. Alternatively, one can describe the function evaluation as composite function

$$F(x) \quad = \quad P_y \circ \Phi_L \circ \Phi_{L-1} \circ \cdots \circ \Phi_1 \circ P_x^T(x) \,, \tag{3.1}$$

where $\Phi_l : \mathscr{H} \to \mathscr{H}$, $s^{(l-1)} \mapsto s^{(l)} = \Phi_l(s^{(l-1)})$ are called *elementary transitions* that operate the *state space* $\mathscr{H}$. Each elementary transition can be written as

$$\Phi_l = P_l \circ \phi_l \circ Q_l + (\mathbf{I} - (1 - \sigma_l)P_l \circ P_l^T) \,. \tag{3.2}$$

where the functions $\phi_l : \mathscr{D}_l \subseteq \mathscr{H}_l \to \mathscr{H}_l \in \{+, -, *, /, \sin, \exp, \dots\}$ are the elementary functions. The $Q_l : \mathscr{H} \to \mathscr{H}_l$ map to the domains of the elementary functions and the $P_l : \mathscr{H}_l \to \mathscr{H}$ map back to the overall state space. The functions $P_x^T$ and $P_y$ are used to map the independent variables $x$ to the state $s^{(0)}$ and $s^{(L)}$ to $y$. The case $\sigma_l = 1$ corresponds to an augmented assignment $s_l = s_l + \phi_l(s_l)$ and $\sigma_l = 0$ to the usual assignment $s_l = \phi_l(s_l)$. For our purposes it suffices to consider a real vector space as state space, i.e., the mappings $P_l$ and $Q_l$ can be written as matrices. For a more detailed discussion see Griewank [Gri03].

# 4 Algorithmic differentiation

In this section we briefly review some key results from the theory of AD that will be necessary in Section 6 and 7. For a detailed discussion we refer to the standard reference "Evaluating Derivatives" by Griewank and Walther [GW08].

## 4.1 The forward mode

One can use univariate Taylor series expansions to compute higher-order (directional) derivatives. The basic observation is that given a smooth curve $x(t) = x_0 + x_1 t$ with $t \in (-\varepsilon, \varepsilon)$, $\varepsilon > 0$, and a smooth function $F$ one obtains a smooth curve $y(t) = F(x(t))$ with the Taylor series expansion

$$y(t) = \sum_{d=0}^{D-1} y_d t^d + \mathcal{O}(t^D) = \sum_{d=0}^{D-1} \frac{1}{d!} \frac{\mathrm{d}^d}{\mathrm{d}t^d} F(x(t)) \bigg|_{t=0} t^d + \mathcal{O}(t^D) \,. \tag{4.1}$$

By application of the chain rule one can interpet the terms of the expansion. The zeroth derivative is the normal function evaluation $y_0 = F(x_0)$ and the first coefficient $y_1 = \frac{\mathrm{d}}{\mathrm{d}t} F(x(t))\big|_{t=0} = \frac{\mathrm{d}}{\mathrm{d}x} F(x_0) \cdot x_1$ is a directional derivative.

In the context of AD it is advantageous to generalize the notion of Taylor series expansions to a purely algebraic task. In other words, for arithmetic with univariate Taylor polynomials (UTP) one extends functions $F : \mathbb{R}^N \to \mathbb{R}^N$ to functions $E_D(F) : \mathbb{R}^N[T]/(T^D) \to \mathbb{R}^M[T]/(T^D)$. We denote representing elements of the polynomial factor ring $\mathbb{R}^N[T]/(T^D)$ as

$$[x]_D := [x_1, \dots, x_{D-1}] := \sum_{d=0}^{D-1} x_d T^d \,, \tag{4.2}$$

where $x_d \in \mathbb{R}^N$ is called *Taylor coefficient*. The quantity $T$ is an indeterminate, i.e., a formal variable. The *extended function* $E_D(F)$ is defined by its action

$$[y]_D \quad = \quad E_D(F)([x]_D) = \sum_{d=0}^{D-1} y_d T^d = \sum_{d=0}^{D-1} \frac{1}{d!} \frac{\mathrm{d}^d}{\mathrm{d}t^d} F\Big(\sum_{d=0}^{D-1} x_d t^d\Big) \bigg|_{t=0} T^d \,. \tag{4.3}$$

The notation $[x]_D \equiv [x]_{:D-1}$ and $[x]_{d+1:D-1} \equiv [x]_{d+1:} \equiv [x_{d+1}, \ldots, x_{D-1}]$ will be useful later on. One can show that this definition is compatible with the usual polynomial addition and multiplication. Furthermore, any composite function $F(x) = (H \circ G)(x) = H(G(x))$ satisfies

$$E_D(F) \quad = \quad E_D(H) \circ E_D(G) \, . \tag{4.4}$$

I.e., the extension function $E_D$ is a homomorphism which preserves function composition. An immediate consequence is that it is necessary to find algorithms only for the very limited set of elementary functions $\phi \in \{+,-,*,/,\sin,\cos,\exp,\ldots\}$. Explicitly, one performs the program transformation $E_D(F) = E_D(\Phi_L) \circ \cdots \circ E_D(\Phi_1)([x]_D)$. We call the action of computing $[y]_D = E_D(F)([x]_D)$, i.e., the resolution of the symbolic dependence to obtain the numerical value $[y]_D$, the *pushforward* of the function $E_D(F)$.

Many functions are implicitly defined by equations of the type $0 = F(x,y) \in \mathbb{R}^M$, where $x \in \mathbb{R}^N$ are the inputs and $y \in \mathbb{R}^M$ the outputs. The idea is to demand that the *defining equations of order D*

$$0 \stackrel{D}{=} E_D(F)([x]_D, [y]_D) \tag{4.5}$$

should be satisfied. By $\stackrel{D}{=}$ it is meant that $[x] \stackrel{D}{=} [y]$ if $x_d = y_d$ for $d = 0, \ldots, D-1$. This is also often written either as $[x] = [y] + \mathcal{O}(T^D)$ or $[x] = [y] \mod T^D$. The defining equations lead directly to an algorithmic approach to compute $[y]_D$, the so-called *Newton-Hensel lifting*. In the literature it is often also just called Hensel-lifting or Newton's method [GW08]. Assuming $[y]_D$ is already known and satisfies $0 \stackrel{D}{=} E_D(F)([x], [y]_D)$, one can lift the computation to a higher degree. Explicitly, one tries to solve $0 \stackrel{D+E}{=} E_{D+E}(F)([x], [y]_{D+E})$. Splitting $[y]_{D+E} = [y]_D + [\Delta y]_E T^D$ and performing a first order Taylor expansion of $F$ about $[y]_D$ yields after a short calculation

$$[\Delta y]_E \quad \stackrel{E}{=} \quad -[F_y]_E^{-1}[\Delta F]_E \, , \tag{4.6}$$

where $E_{D+E}(F)([x], [y]_D) \stackrel{D+E}{=} [\Delta F]_E T^D$ and $[F_y]_E := E_E(\frac{\mathrm{d}F}{\mathrm{d}y})([x], [y]_E)$. Setting $E = D$ means that at each step the number of correct coefficients is doubled. In this case we call it *Newton's method*. In the case $E = 1$ only the next coefficient is computed. We call the special case $E = 1$ *sequential Hensel lifting* which is also the formula that is often given as part of the implicit function theorem. The difference is that Newton-Hensel lifting is a purely algebraic task. For a discussion on how to obtain asymptotically fast algorithms and for the nomenclature see e.g. Bernstein [Ber01, Ber08].

## 4.2   The reverse mode

The basic idea of the reverse mode of AD is to pullback linear forms $\alpha$ to obtain an explict mapping $\bar{y} \mapsto \bar{x}$. I.e., given $F : \mathbb{R}^N \to \mathbb{R}^M$, $y = F(x)$ one has

$$\alpha(\bar{y}, y) \quad = \quad \sum_{m=1}^{M} \bar{y}_m \mathrm{d}y_m = \sum_{m=1}^{M} \bar{y}_m \sum_{n=1}^{N} \frac{\partial F_m}{\partial x_n} \mathrm{d}x_n = \sum_{n=1}^{N} \bar{x}_n \mathrm{d}x_n = \alpha(\bar{x}, x) \, , \tag{4.7}$$

where $\bar{x}_n = \sum_{m=1}^{M} \bar{y}_m \frac{\partial F_m}{\partial x_n}$. For notational reasons one uses $\sum_{n=1}^{N} \bar{x}_n \mathrm{d}x_n \equiv \bar{x}^T \mathrm{d}x$. We call the action of going back one level of the symbolic dependence the *pullback* of the linear form $\alpha(\bar{y}, y)$. For a more detailed discussion on calculations with differentials see Magnus and Neudecker [MN99].

It is also possible to compute higher-order derivatives by combining UTP arithmetic and the reverse mode of AD. For that, the UTP algorithms are interpreted as functions mapping $D$ coefficients $x_d, 0 \leq d < D$ to $D$ coefficients $y_d, 0 \leq d < D$, i.e., a mapping from $\mathbb{R}^{N \times D} \to \mathbb{R}^{M \times D}$ with a special structure. One can formally define a linear form by

$$E_D(\alpha)([\bar{y}]_D, [y]_D) := [y]_D^T \mathrm{d}[y]_D . \tag{4.8}$$

Here, $\mathrm{d}[y]_D = \sum_{d=0}^{D-1} \mathrm{d}y_d T^d$ is a formal polynomial where each coefficient is a differential and $[\bar{y}]_D^T \mathrm{d}[y]_D = \sum_{m=1}^{M} [\bar{y}_m]_D \mathrm{d}[y_m]_D$ computes the polynomial multiplication of formal polynomials. One can show that

$$E_D(\alpha)([\bar{y}]_D, [y]_D) \;\overset{D}{=}\; [\bar{y}]_D^T E_D(\frac{\partial F}{\partial x})([x]_D)\mathrm{d}[x]_D \overset{D}{=} [\bar{x}_n]_D^T \mathrm{d}[x_n]_D = E_D(\alpha)([\bar{x}]_D, [x]_D) \tag{4.9}$$

holds [Chr91]. One can interpret this result as follows: If $[\bar{y}]_D = w \in \mathbb{R}^M$ then $[\bar{x}]_D = E_D(w^T \frac{\partial F}{\partial x})([x]_D)$. Setting $w = e_i$ a Cartesian basis vector would yield the Taylor expansion of the $i$'th row of the Jacobian. The interpretation of the Taylor coefficients as derivatives yields higher-order derivatives. If $M = 1$ and $w = 1$ one obtains the Taylor expansion of the gradient $[\bar{x}]_D = E_D(\nabla F)([x]_D)$. E.g., propagating the UTP $[x]_2 = x_0 + x_1 T$ would yield $[\bar{x}]_2 = \bar{x}_0 + \bar{x}_1 T$ where $\bar{x}_0 = \nabla_x F(x)$ and $\bar{x}_1 = \nabla_x^2 F(x) \cdot x_1$, i.e., a Hessian-vector product.

# 5 Defining equations of numerical linear algebra functions

As briefly mentioned in the introduction, Numerical Linear Algebra (NLA) functions can be viewed as algorithms representing a concatenation of functions like $+, -, *, /, \sin, \cos, \ldots$ and thus it is possible to apply the AD techniques described in the previous section directly to the algorithm. However, there is also the possibility to regard the problem from a more abstract point of view. Many NLA functions are implicitly defined by a system of equations.

For instance the QR decomposition is defined by the defining equations

$$\begin{align} 0 &= QR - A \tag{5.1} \\ 0 &= Q^T Q - \mathbf{I} \tag{5.2} \\ 0 &= P_L \circ R , \tag{5.3} \end{align}$$

where $A, R \in \mathbb{R}^{M \times N}$ with $M \geq N$ and $Q \in \mathbb{R}^{M \times M}$. The functional dependence of the defining equations is denoted

$$Q, R = \mathrm{qr}(A) . \tag{5.4}$$

Only the first $N$ rows $R_{:N,:} \in \mathbb{R}^{N \times N}$ of $R$ are nonzero. For convenience reasons we use the slicing notation $i : j = (i, i+1, \ldots, j)$.

The defining equations of the symmetric eigenvalue decomposition are given by

$$\begin{align} 0 &= Q^T A Q - \Lambda \tag{5.5} \\ 0 &= Q^T Q - \mathbf{I} \tag{5.6} \\ 0 &= (P_L + P_R) \circ \Lambda , \tag{5.7} \end{align}$$

where $A \in \mathbb{R}^{M \times M}$ is symmetric. The functional dependence is denoted $\Lambda, Q = \mathrm{eigh}(A)$. We call the matrices $(P_L)_{ij} = \delta_{j<i}$ and $(P_R)_{ij} = \delta_{i<j}$ skeletal projectors since their elementwise product with a matrix returns strictly lower resp. strictly upper triangular matrices.

## 6   The QR decomposition

Before we derive algorithms based on the defining equations, we briefly investigate what can go wrong if a typical implementation of the QR decomposition using Householder reflections is evaluated in UTP arithmetic. Consider Algorithm 5.1.1 from the book "Matrix Computations" by Golub and Van Loan [GVL96] which we adapted to our notation in Algorithm 1. From the AD point of view, the problematic part in the code is the check $\sigma = 0$. Since a paradigm of AD tools is that the control flow must remain unchanged, the check $\sigma = 0$ only considers the zeroth coefficient $x_0$ of a UTP. Hence, if $[x]_2 = e_1 + x_1 T$ is given as input and $x_1 \neq 0$, the algorithm will simply evaluate $\beta = 0$ and return. As final result, one obtains a matrix $[R]_2$ where $R_1$ is not upper triangular. The LAPACK implementation (LAPACK-3.2.2) of DGEQRFP.f calls the subroutine DLARFGP.f which contains a similar check. Hence, automatic augmentation based on AD principles can go wrong in such cases.

As a side remark, note that additionally the function realized by this algorithm has a pole at $\sigma = 0$, producing numerical overflow for $\sigma \approx 0$.

---

**input** : $x \in \mathbb{R}^N$
**output**: $v \in \mathbb{R}^N$ with $v_1 = 1$
**output**: $\beta \in \mathbb{R}$

$\sigma = x_{2:}^T x_{2:}$
$v = \begin{pmatrix} 1 \\ x_{2:} \end{pmatrix}$
**if** $\sigma = 0$ **then**
$\quad | \quad \beta = 0$
**else**
$\quad \quad \mu = \sqrt{x_1^2 + \sigma}$
$\quad \quad$ **if** $x_1 \leq 0$ **then**
$\quad \quad \quad | \quad v_1 = x_1 - \mu$
$\quad \quad$ **else**
$\quad \quad \quad | \quad v_1 = -\sigma/(x_1 + \mu)$
$\quad \quad$ **end**
$\quad \quad \beta = 2v_1^2/(\sigma + v_1^2)$
$\quad \quad v = v/v_1$
**end**

**Algorithm 1**: Householder Vector. The reflector is $P_v = I - \beta v v^\top$, with $v_1 = 1$.

## 6.1   Pushforward in Taylor arithmetic

We now come to the higher–level approach that is based on the defining equations given in Section 5. To compute $[Q]_D, [R]_D = E_D(\mathrm{qr})([A]_D)$ one can apply Newton-Hensel lifting to solve

$$0 \overset{D}{=} [Q]_D[R]_D - [A]_D \tag{6.1}$$

$$0 \overset{D}{=} [Q]_D^T[Q]_D - \mathbf{I} \tag{6.2}$$

$$0 \overset{D}{=} P_L \circ [R]_D . \tag{6.3}$$

The direct application of Eqn. (4.6) should be avoided since $F_y$ is sparse and has a lot of structure. Rather, one assumes that one has already computed $[Q]_D$ and $[R]_D$ and computes the next $1 \le E \le D$ coefficients by performing a first order Taylor expansion $[Q]_{D+E} = [Q]_D + [\Delta Q]_E T^D$ and $[R]_{D+E} = [R]_D + [\Delta R]_E T^D$ and tries to solve for the yet unknown $[\Delta R]_E$ and $[\Delta Q]_E$. As result one obtains Proposition 1. For convenience, we use the convention that $R_{d;i,j}$ is the $i$'th row and $j$'th column of the $d$'th coefficient of $[R]_D$.

**Proposition 1.** *Let $[A]_{D+E} \in \mathbb{R}^{M \times N}[T]/(T^{D+E})$ with $M \ge N$ and $1 \le E \le D$, $[R]_D \in \mathbb{R}^{M \times N}[T]/(T^D)$ where $[R_{:N,:}]_D$ is upper triangular with nonsingular $R_{0;:N,:}$ and $[Q]_D \in \mathbb{R}^{M \times M}[T]/(T^D)$ orthogonal be given and satisfy the defining equations of order $D$. Then $[\Delta R_{:N,:}]_E \equiv [R_{:N,:}]_{D:D+E-1}$ and $[\Delta Q]_E \equiv [Q]_{D:D+E-1}$ are given by*

$$[\Delta F]_E T^D \overset{D+E}{=} -[Q]_D[R]_D + [A]_{D+E} \tag{6.4}$$

$$[\Delta G]_E T^D \overset{D+E}{=} -[Q]_D^T[Q]_D + \mathbf{I} \tag{6.5}$$

$$[S]_E \overset{E}{=} \frac{1}{2}[\Delta G]_E \tag{6.6}$$

$$P_L \circ ([X_{:,:N}]_E) \overset{E}{=} P_L \circ \left([Q]_E^T[\Delta F]_E[R_{:N,:}]_E^{-1}\right) - P_L \circ [S_{:,:N}]_E \tag{6.7}$$

$$[\Delta R]_E \overset{E}{=} [Q]_E^T[\Delta F]_E - ([S]_E + [X]_E)[R]_E \tag{6.8}$$

$$[\Delta Q]_E \overset{E}{=} [Q]_E ([S]_E + [X]_E) , \tag{6.9}$$

*where $P_L \in \mathbb{R}^{M \times N}$ with $(P_L)_{ij} = \delta_{j<i}$.*

*Proof.* In the Appendix A.1.1. □

## 6.2   Pullback

**Proposition 2.** *Let $A, R, \bar{R} \in \mathbb{R}^{M \times N}$ resp. $Q, \bar{Q} \in \mathbb{R}^{M \times M}$ be given and it holds $M \ge N$, $\mathrm{rank}(A) = N$, $Q, R = \mathrm{qr}(A)$. Then $\bar{A} \in \mathbb{R}^{M \times N}$ can be computed by*

$$\bar{A} = Q \left(\bar{R} + \left(P_L \circ \left(R\bar{R}^T - \bar{R}R^T + Q^T\bar{Q} - \bar{Q}^TQ\right)\right) R^{+T}\right) . \tag{6.10}$$

*Here, $R^+$ denotes the Moore-Penrose pseudoinverse of $R$. That means it satisfies $RR^+R = R$ and since $R$ has full column rank also $R^+R = \mathbf{I}$.*

*Proof.* In Appendix A.1.2. □

## 6.3  Explicit algorithms

One can use Proposition 1 to derive an explicit algorithm as shown in Algorithm 2, where at each step $E = 1$ is used.

---

**input** : $[A]_D = [A_0, \ldots, A_{D-1}]$, where $A_d \in \mathbb{R}^{M \times N}$, $d = 0, \ldots, D - 1$ and rank $(A_0) = N$, $M \geq N$.
**output**: $[Q]_D = [Q_0, \ldots, Q_{D-1}]$ matrix with orthonormal column vectors, where $Q_d \in \mathbb{R}^{M \times N}$,
$\qquad d = 0, \ldots, D - 1$
**output**: $[R]_D = [R_0, \ldots, R_{D-1}]$ upper triangular, where $R_d \in \mathbb{R}^{N \times N}$, $d = 0, \ldots, D - 1$

$Q_0, R_0 = \mathrm{qr}(A_0)$

**for** $d = 1$ **to** $D - 1$ **do**
$\quad \Delta F = A_d - \sum_{k=1}^{d-1} Q_{d-k} R_k$
$\quad S = -\frac{1}{2} \sum_{k=1}^{d-1} Q_{d-k}^T Q_k$
$\quad X_{:,:N} = P_L \circ (Q_0^T \Delta F R_{0;:N,:N}^{-1} - S_{:,:N})$
$\quad X_{:,N+1:} = 0$
$\quad X = X - X^T$
$\quad R_d = Q_0^T \Delta F - (S + X) R_0$
$\quad Q_d = Q_0(S + X)$
**end**

**Algorithm 2**: Sequential Hensel lifting for the QR decomposition.

---

The pullback can be computed in Taylor arithmetic. In the global derivative accumulation it is necessary to update the value of $[\bar{A}]_D$. This happens if $[A]_D$ is input of more than one function. The algorithm for the pullback takes this into consideration.

---

**input** $\qquad$ : $[A]_D = [A_0, \ldots, A_{D-1}]$, where $A_d \in \mathbb{R}^{M \times N}$, $d = 0, \ldots, D - 1$, $M \geq N$.
**input** $\qquad$ : $[Q]_D = [Q_0, \ldots, Q_{D-1}]$, where $Q_d \in \mathbb{R}^{M \times M}$, $d = 0, \ldots, D - 1$
**input** $\qquad$ : $[R]_D = [R_0, \ldots, R_{D-1}]$, where $R_d \in \mathbb{R}^{M \times N}$, $d = 0, \ldots, D - 1$
**input/output**: $[\bar{A}]_D = [\bar{A}_0, \ldots, \bar{A}_{D-1}]$, where $\bar{A}_d \in \mathbb{R}^{M \times N}$, $d = 0, \ldots, D - 1$, $M \geq N$.
**input** $\qquad$ : $[\bar{Q}]_D = [\bar{Q}_0, \ldots, \bar{Q}_{D-1}]$, where $\bar{Q}_d \in \mathbb{R}^{M \times M}$, $d = 0, \ldots, D - 1$
**input** $\qquad$ : $[\bar{R}]_D = [\bar{R}_0, \ldots, \bar{R}_{D-1}]$, where $\bar{R}_d \in \mathbb{R}^{M \times N}$, $d = 0, \ldots, D - 1$

$$\begin{aligned} [\bar{A}]_D \; = \; & [\bar{A}]_D + [Q]_D \cdot \\ & \cdot \left( [\bar{R}]_D + \left( P_L \circ \left( [R]_D [\bar{R}]_D^T - [\bar{R}]_D [R]_D^T + [Q]_D^T [\bar{Q}]_D - [\bar{Q}]_D^T [Q]_D \right) \right) [R]_D^{+T} \right) \end{aligned}$$

**Algorithm 3**: Pullback of the QR decomposition in Taylor arithmetic. The inputs $[A]_D, [Q]_D, [R]_D$ must satisfy the defining equations.

# 7 The real symmetric eigenvalue decomposition

The problem of finding eigenvalues and eigenvectors arises in a wide variety of practical applications. As for the QR decomposition, we want to have algorithms that compute the real symmetric eigenvalue decomposition in UTP arithmetic as well as pullback algorithms. The symmetric eigenvalue decomposition is also important since the Singular Value Decomposition (SVD) of real matrices is closely related to it. More explicitly, one can compute the SVD of a matrix $A \in \mathbb{R}^{M \times N}$ of rank $r$., i.e., $A = U\Sigma V^T$, where $\Sigma = \mathrm{diag}(\Sigma_1, 0)$, $U = (U_1, U_2)$, $U_1 \in \mathbb{R}^{M \times r}$, $V = (V_1, V_2)$, $V_1 \in \mathbb{R}^{N \times r}$ as

$$C = \begin{pmatrix} 0 & A \\ A^T & 0 \end{pmatrix} = P^T \begin{pmatrix} \Sigma_1 & 0 & 0 \\ 0 & -\Sigma_1 & 0 \\ 0 & 0 & 0 \end{pmatrix} P \,,$$

where

$$P = \frac{1}{\sqrt{2}} \begin{pmatrix} U_1 & U_1 & \sqrt{2}U_2 & 0 \\ V_1 & -V_1 & 0 & \sqrt{2}V_2 \end{pmatrix}^T$$

is orthogonal [Bjö96].

## 7.1 Pushforward in Taylor arithmetic

Given the symmetric polynomial matrix $[A]_D \in \mathbb{R}^{N \times N}[T]/(T^D)$. The eigenvalue decomposition is the solution $[\Lambda]_D, [Q]_D \in \mathbb{R}^{N \times N}[T]/(T^D)$ of the implicit system

$$0 \overset{D}{=} [Q]_D^T [A]_D [Q]_D - [\Lambda]_D \tag{7.1}$$

$$0 \overset{D}{=} [Q]_D^T [Q]_D - \mathbf{I} \tag{7.2}$$

$$0 \overset{D}{=} (P_L + P_R) \circ [\Lambda]_D \,, \tag{7.3}$$

which is called the *defining equations of order $D$*. We also assume that the eigenvalues are sorted as $[\Lambda_{11}]_D \leq [\Lambda_{22}]_D \leq \cdots \leq [\Lambda_{NN}]_D$. The functional dependence is denoted

$$[\Lambda]_D, [Q]_D = \mathrm{eigh}\,([A]_D) \,. \tag{7.4}$$

Let $\Lambda, Q = \mathrm{eigh}\,(A)$ be the usual symmetric eigenvalue decomposition. We denote the diagonal of $[\Lambda]_D$ as $[\lambda]_D = \mathrm{diag}([\Lambda]_D)$. If eigenvalues are repeated, i.e., multiple, the eigenvectors generalize to eigenspaces and the columns of $Q$, that are associated to such a multiple eigenvalue, are not unique. Rather, any orthonormal basis could be the result. This has consequences for the Hensel-Newton lifting approach, because given $[Q]_D$ and $[R]_D$ that satisfy the defining equations of order $D$ it is generally not possible to find a $[\Delta Q]_E$ and $[\Delta R]_E$ such that $[Q]_{D+E} = [Q]_D + [\Delta Q]_E T^D$ and $[R]_{D+E} = [R]_D + [\Delta R]_E T^D$ satisfy the defining equations of order $D + E$. The higher-order coefficients $[\Delta A]_E$ enforce additional conditions on the chosen basis of the eigenspaces. A wrong choice of $[Q]_D$ means that $0 \overset{D+E}{=} (P_L + P_R) \circ [\Lambda]_{D+E}$ cannot be satisfied. However, $0 \overset{D}{=} P_b^D \circ [\Lambda]_{D+E}$ can be satisfied. The matrix $P_b^D$ is a skeletal projector with zero blocks on the main diagonal

11

whose size corresponds to the multiplicity of an eigenvalue $[\lambda]_D$ and all other entries are ones. The *multiplicity* $m^d([\lambda_j]_D)$ of an eigenvalue $[\lambda_j]_D$ of *level d* is defined to be the number of $i \in \mathbb{N}$ s.t. $[\lambda_j]_D \overset{d}{=} [\lambda_i]_D$. I.e.,

$$\mathrm{diag}([\Lambda]_d) \quad = \quad (\underbrace{[\lambda_1]_d,\ldots,[\lambda_1]_d}_{m^d([\lambda_1]_D)\ \text{times}},\ldots,\underbrace{[\lambda_{N_b^d}]_d,\ldots,[\lambda_{N_b^d}]_d}_{m^d([\lambda_{N_b^d}]_D)\ \text{times}}),$$

where $N_b^d$ is the number of different eigenvalues at level $d$. We define $b^d \in \mathbb{N}^{N_b^d+1}$ to be a vector satisfying $m^d([\lambda_{n_b}]_D) = b_{n_b+1}^d - b_{n_b}^d$. The symbol $b$ is used because it relates to blocks in the matrix. The elements of $P_b^d$ satisfy $(P_b^d)_{ij} = 1 - \sum_{n_b=1}^{N_b^d+1} \delta_{b_{n_b}^d \leq i < b_{n_b+1}^d}\, \delta_{b_{n_b}^d \leq j < b_{n_b+1}^d}$. This notation is a little cumbersome but turns out to be helpful. One defines $b^0 = [0, N+1]$. The vector $b^1$ represents the multiplicities of the usual symmetric eigenvalue decomposition. E.g., for $N = 3$ and $b^d = [1, 3, 4]$ one has

$$P_b^d = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}.$$

We reformulate the overall problem as a sequence of subproblems. We call the implicit system

$$0 \quad \overset{D}{=} \quad [Q^d]_D^T [A]_D [Q^d]_D - [\Lambda^d]_D \tag{7.5}$$

$$0 \quad \overset{D}{=} \quad [Q^d]_D^T [Q^d]_D - \mathbf{I} \tag{7.6}$$

$$0 \quad \overset{d}{=} \quad (P_L + P_R) \circ [\Lambda]_d \tag{7.7}$$

$$0 \quad \overset{D}{=} \quad P_b^d \circ [\Lambda^d]_D , \tag{7.8}$$

the *relaxed problem of level d and order D*. I.e., it is assumed that up to order $d$ the original problem is solved but only block diagonalized for the higher order coefficients.

To give an illustrative example consider this relaxed problem of order 3 and level 2. At this point of the algorithm, one has potentially obtained a matrix polynomial $[\Lambda]_3 = \sum_{d=0} \Lambda_d T^d$ with coefficients of the form

$$\Lambda_0 = \begin{pmatrix} 1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & & 2 & & \\ & & & & 2 & \\ & & & & & 3 \end{pmatrix}, \quad \Lambda_1 = \begin{pmatrix} 2 & & & & & \\ & 2 & & & & \\ & & 3 & & & \\ & & & 2 & & \\ & & & & 2 & \\ & & & & & 2 \end{pmatrix}, \quad \Lambda_2 = \begin{pmatrix} 1 & 3 & & & & \\ 3 & 5 & & & & \\ & & 7 & & & \\ & & & 1 & 2 & \\ & & & 2 & 3 & \\ & & & & & 7 \end{pmatrix}.$$

I.e., $\Lambda_0$ and $\Lambda_1$ are already diagonal. Since there are two eigenvalues with multiplicity $m^2([\lambda]_3) = 2$ it follows that $\Lambda_2$ is only block diagonal. Note that the eigenvalues are not globally sorted by value in the higher coefficients but only in the subblocks defined by the lower order coefficients. In this example, the repeated eigenvalues in the first block split at the lift from $d = 0$ to $d = 1$. The blocks are defined by $b^1 = [1, 4, 6, 7]$ and $b^2 = [1, 3, 4, 6, 7]$. The blocks in $\Lambda_2$ are defined by $b^2$.

The function that solves the relaxed problem of order $D$ and level $d$ is denoted

$$[\Lambda^d]_D, [Q^d]_D \quad = \quad \mathrm{eigh}_d([A]_D) . \tag{7.9}$$

The idea is to implement an algorithm that successively increases $d$ by one. For convenience we define $[Q^0]_D := \mathbf{I}$ and $[\Lambda^0]_D := [A]_D$.

**Theorem 3.** *Let $[A]_D$ be given. Then the solution of*

$$[Q^{d+1}]_D, [\Lambda^{d+1}]_D \overset{D}{=} \mathrm{eigh}_{d+1}([A]_D) \tag{7.10}$$

*can be computed from the solution $[Q^d]_D, [\Lambda^d]_D \overset{D}{=} \mathrm{eigh}_d([A]_D)$ by computing*

$$[\hat{\Lambda}_{s,s}]_{D-d}, [\hat{Q}_{s,s}]_{D-d} \overset{D-d}{=} \mathrm{eigh}_1([\Lambda^d_{s,s}]_{d:}), \tag{7.11}$$

*where $s = b^d_{n_b} : b^d_{n_b+1} - 1$ are slice indices and $n_b = 1, \ldots, N^d_b$. All other elements of $[\hat{Q}]_{D-d}$ and $[\hat{\Lambda}]_{D-d}$ are zero. I.e., $[\hat{Q}]_{D-d}$ and $[\hat{\Lambda}]_{D-d}$ are block diagonal. It holds that*

$$[\Lambda^{d+1}]_D \overset{D}{=} [\Lambda^d]_d + [\hat{\Lambda}]_{D-d} T^d \tag{7.12}$$

$$[Q^{d+1}]_D \overset{D}{=} [Q^d]_D [Q]_D, \tag{7.13}$$

*where $[Q]_D = [\hat{Q}]_{D-d} + [\Delta Q]_d T^{D-d}$ for some $[\Delta Q]_{D-d}$ that satisfies*

$$0 \overset{D}{=} [Q]^T_D [Q]_D - \mathbf{I}. \tag{7.14}$$

*Proof.* We need to show that $[\Lambda^{d+1}]_D$, $[Q^{d+1}]_D$ is a solution to the relaxed equations of level $d+1$ and order $D$. From the definition of $\mathrm{eigh}_1$ it follows that $0 = (P_L + P_R) \circ [\Lambda^{d+1}]_{d+1}$ and $0 = P^{d+1}_b \circ [\Lambda^{d+1}]_D$ is satisfied. We also know that $0 \overset{D}{=} [Q^{d+1}]^T_D [Q^{d+1}]_D - \mathbf{I} \overset{D}{=} [Q]^T_D [Q^d]^T_D [Q^d]_D [Q]_D - \mathbf{I}$ is fulfilled because $0 \overset{D}{=} [Q^d]^T_D [Q^d]_D - \mathbf{I}$ and $0 \overset{D}{=} [Q]^T_D [Q]^T_D - \mathbf{I}$. Hence, it only remains to show that the third defining equation is satisfied which is shown by the following straight-forward calculation:

$$
\begin{aligned}
0 &\overset{D}{=} [Q]^T_D [Q^d]^T_D [A]_D [Q^d]_D [Q]_D - [\Lambda^{d+1}]_D \\
&\overset{D}{=} [Q]^T_D [\Lambda^d]_D [Q]_D - [\Lambda^{d+1}]_D \\
&\overset{D}{=} [Q]^T_D ([\Lambda^d]_d + [\Lambda^d]_{d:} T^d) [Q]_D - [\Lambda^d]_d - [\hat{\Lambda}]_{D-d} T^d \\
&\overset{D}{=} [Q]^T_D [\Lambda^d]_d [Q]_D + [Q]^T_D [\Lambda^d]_{d:} [Q]_D T^d - [\Lambda^d]_d - [\hat{\Lambda}]_{D-d} T^d \\
&\overset{D}{=} [\Lambda^d]_d [Q]^T_D [Q]_D + [\hat{Q}]^T_{D-d} [\Lambda^d]_{d:} [\hat{Q}]_{D-d} T^d - [\Lambda^d]_d - [\hat{\Lambda}]_{D-d} T^d \\
&\overset{D}{=} [\hat{Q}]^T_{D-d} [\Lambda^d]_{d:} [\hat{Q}]_{D-d} T^d - [\hat{\Lambda}]_{D-d} T^d \\
&\overset{D-d}{=} [\hat{Q}]^T_{D-d} [\Lambda^d]_{d:} [\hat{Q}]_{D-d} - [\hat{\Lambda}]_{D-d}.
\end{aligned}
$$

In the fifth line it has been used that the diagonalization has only to be performed for block diagonal matrices. If the eigenvalues are already distinct there is nothing to diagonalize and the step can be skipped. It also means that one may interchange $[\Lambda^d]_d$ with $[Q]_D$. $\qquad \square$

The following proposition gives us the means to diagonalize a matrix in the zeroth degree and block diagonalize w.r.t. the blocks defined by the repeated eigenvalues. I.e., it gives the justification that the solution of Eqn. (7.11) can be found. In the case of distinct eigenvalues the application of this algorithm already solves the original problem.

**Proposition 4.** *Let* $[A]_{D+E} = [A]_D + [\Delta A]_E T^D$ *be given and* $[\Lambda^d]_D$, $[Q^d]_D$ *be a solution of the relaxed problem of level* $d = 1$ *and order* $D$. *Then it exist* $[\Delta\Lambda^d]_E$ *and* $[\Delta Q^d]_E$ *such that* $[\Lambda^d]_{D+E} = [\Lambda^d]_D + [\Delta\Lambda^d]_E T^D$ *and* $[\Delta Q^d]_{D+E} = [\Delta Q^d]_D + [\Delta Q^d]_E T^D$ *are a solution of the relaxed problem of level* $d = 1$ *and order* $D+E$. *A closed form solution is*

$$[\Delta\Lambda^d]_E \overset{E}{=} \bar{P}_b^d \circ [K]_E \tag{7.15}$$

$$[\Delta Q^d]_E \overset{E}{=} [Q^d]_E \left([\Delta G]_E + P_b^d \circ ([K]_E/[E]_E)\right) \tag{7.16}$$

*where* $[\Delta F]_E T^D \overset{D+E}{=} [Q^d]_D^T [A]_D [Q^d]_D - [\Lambda^d]_D$ *and* $[\Delta G]_E T^D \overset{D+E}{=} -\frac{1}{2}\left([Q^d]_D^T[Q^d]_D - \mathbf{I}\right)$, $[K]_E \overset{E}{=} [\Delta F]_E + ([\Lambda]_E[\Delta G]_E + [\Delta G]_E[\Lambda]_E) + [Q^d]_E^T[\Delta A]_E[Q^d]_E$ *and* $[E_{ij}]_E \overset{E}{=} [\Lambda^d_{jj}]_E - [\Lambda^d_{ii}]_E$. *The expression* $[K]_E/[E]_E$ *denotes an element-wise division.* $P_b^d$ *is a matrix with only ones on the diagonal blocks defined by the multiplicity of eigenvalues in* $\Lambda_0$. $\bar{P}_b^d$ *is defined s.t.* $\bar{P}_b^d + P_b^d$ *is a matrix full of ones. One can see here that if the eigenvalues are distinct, then* $\bar{P}_b^d$ *is the identity matrix* $\mathbf{I}$.

*Proof.* We set $Q^d \equiv Q$ etc. for notational simplicity. Applying Newton-Hensel lifting to the defining equations yields

$$
\begin{aligned}
0 \;&\overset{D+E}{=}\; ([Q]_D + [\Delta Q]_E T^D)^T([Q]_D + [\Delta Q]_E T^D) - \mathbf{I} \\
&\overset{E}{=}\; -2[\Delta G]_E + [\Delta Q]_E^T[Q]_E + [Q]_E^T[\Delta Q]_E \\
&\overset{E}{=}\; -2[\Delta G]_E + 2[S]_E \;, \\
0 \;&\overset{D+E}{=}\; ([Q]_D + [\Delta Q]_E T^D)^T([A]_D + [\Delta A]_E T^D)([Q]_D + [\Delta Q]_E T^D) - ([\Lambda]_D + [\Delta\Lambda]_E T^D) \\
&\overset{E}{=}\; [\Delta F]_E + [Q]_E^T[\Delta A]_E[Q]_E + [\Delta Q]_E^T[Q]_E[\Lambda]_E + [\Lambda]_E[Q]_E^T[\Delta Q]_E - [\Delta\Lambda]_E \\
&\overset{E}{=}\; [K]_E + [X]_E[\Lambda]_E - [\Lambda]_E[X]_E - [\Delta\Lambda]_E \\
&\overset{E}{=}\; [K]_E + [E]_E \circ [X]_E - [\Delta\Lambda]_E \;. \tag{7.17}
\end{aligned}
$$

Thus $[\Delta\Lambda]_E \overset{E}{=} \bar{P}_b^d \circ [K]_E$ and $[X]_E^T \overset{E}{=} P_b^d \circ ([K]_E/[E]_E)$. Above, $[\Delta Q]_E^T[Q]_E \overset{E}{=} [S]_E + [X]_E$, $[S]_E$ symmetric and $[X]_E$ antisymmetric (Lemma 15) has been used. □

It remains to show that Eqn. (7.14) can be satisfied.

**Lemma 5.** *Let* $[Q]_D$ *be given and it satisfies the defining equation* $0 \overset{D}{=} [Q]_D^T[Q]_D - \mathbf{I}$. *Then the solution can be lifted to* $D+E$ *with* $E \leq D$. *I.e., it is possible to find* $[Q]_{D+E} := [Q]_D + [\Delta Q]_E T^D$ *s.t.* $0 \overset{D+E}{=} [Q]_{D+E}^T[Q]_{D+E} - \mathbf{I}$. *A closed form solution for* $[\Delta Q]_E$ *is given by*

$$[\Delta Q]_E \overset{E}{=} [Q]_E[S]_E \;, \tag{7.18}$$

*where* $[S]_E T^D \overset{D+E}{=} -\frac{1}{2}\left([Q]_D^T[Q]_D - \mathbf{I}\right)$.

*Proof.* In Appendix A.1.3. □

## 7.2  Pullback

The eigenvalue decomposition is non-differentiable at points where eigenvalues are repeated and hence the defining equations do not define a *well behaved implicit mapping* as described by Christianson [Chr98]. However, the eigenvalue decomposition is typically used within a global context where the non-uniqueness and non-differentiability can be worked around. Here, we give only the pullback algorithm that is correct for unique eigenvalues.

**Proposition 6** (Pullback of the Symmetric Eigenvalue Decomposition with Distinct Eigenvalues:). *Given* $A, Q, \Lambda, \bar{Q}, \bar{\Lambda}$, *where all eigenvalues are distinct, one can compute* $\bar{A}$ *by*

$$H_{ij} \;=\; (\lambda_j - \lambda_i)^{-1} \quad if \quad i \neq j, \quad 0 \quad else \tag{7.19}$$

$$\bar{A} \;=\; Q \left( \bar{\Lambda} + H \circ (Q^T \bar{Q}) \right) Q^T \tag{7.20}$$

*Proof.* In Appendix A.1.4. □

## 7.3  Explicit algorithms

---

**input** : $[Q]_d = [Q_0, \ldots, Q_{d-1}]$ with $0 \overset{d}{=} [Q]_d^T [Q]_d - \mathbf{I}$
**input** : $D \in \mathbb{N}$
**output**: $[Q]_D = [Q_0, \ldots, Q_{D-1}]$, where $0 \overset{D}{=} [Q]_D^T [Q]_D - \mathbf{I}$

**for** $k = d$ **to** $D - 1$ **do**
$\quad \left| \quad Q_k = -\frac{1}{2} Q_0 \sum_{i=1}^{k-1} Q_i^T Q_{k-i} \right.$
**end**

---

**Algorithm 4**: This algorithm computes $[Q]_D = \text{qlift}([Q]_d, D)$ as described in Proposition 5 using sequential Hensel-lifting ($E = 1$).

**input** : $[A]_D = [A_0, \ldots, A_{D-1}]$, where $A_d \in \mathbb{R}^{N \times N}$ symmetric positive definite, $d = 0, \ldots, D-1$
**output**: $[\Lambda]_D = [\Lambda_0, \ldots, \Lambda_{D-1}]$, where $\Lambda_0 \in \mathbb{R}^{N \times N}$ diagonal and $\Lambda_d \in \mathbb{R}^{N \times N}$ block diagonal
　　　　$d = 1, \ldots, D-1$.
**output**: $[Q]_D = [Q_0, \ldots, Q_{D-1}]$ orthogonal, where $Q_d \in \mathbb{R}^{N \times N}$
**output**: $b \in \mathbb{N}^{N_b+1}$, array of integers defining the blocks. The integer $N_b$ is the number of blocks. Each
　　　　block has the size of the multiplicity of an eigenvalue $\lambda_{n_b}$ of $\Lambda_0$ s.t. for $s = b_{n_b} : b_{n_b+1} - 1$ one has
　　　　$(Q_{0;:,s})^T A_0 Q_{0;:,s} = \lambda_{n_b} \mathbf{I}$.

$\Lambda_0, Q_0 = \text{eigh}(A_0)$

compute $b \in \mathbb{R}^{N_b+1}$

$E_{ij} = \Lambda_{0;jj} - \Lambda_{0;ii}$
$H = P_b \circ (1/E)$

**for** $d = 1$ **to** $D-1$ **do**
　　$\Delta F = \sum_{|i|=d} Q_{i_1}^T A_{i_2} Q_{i_3}$
　　$S = -\frac{1}{2} \sum_{k=1}^{d-1} Q_{d-k}^T Q_k$
　　$K = \Delta F + Q_0^T A_d Q_0 + S\Lambda_0 + \Lambda_0 S$
　　$Q_d = Q_0(S + H \circ K)$
　　$\Lambda_d = \bar{P}_b \circ K$
**end**

**Algorithm 5**: This algorithm computes $[\Lambda]_D, [Q]_D, b = \text{eigh}_1([A]_D)$ as specified by 4 using sequential Hensel-lifting ($E = 1$). I.e., the zeroth coefficient is diagonalized and the higher order coefficients are block diagonalized. The symbol $i \in \mathbb{N}_0^3$ denotes a multi-index, i.e., the summation $\sum_{|i|=d}$ goes over all possible $i$ such that $|i| \equiv \sum_{k=1}^3 i_k = d$.

---

**input** : $[A]_D = [A_0, \ldots, A_{D-1}]$ symmetric with $A_d \in \mathbb{R}^{N \times N}$
**output**: $[\Lambda]_D = [\Lambda_0, \ldots, \Lambda_{D-1}]$, where $\Lambda_d \in \mathbb{R}^{N \times N}$ diagonal for $d = 0, \ldots, D-1$.
**output**: $[Q]_D = [Q_0, \ldots, Q_{D-1}]$ orthogonal, where $Q_d \in \mathbb{R}^{N \times N}$

$[\Lambda^0]_D = [A]_D$, $[Q^0]_D = \mathbf{I}$ and $b^0 = [1, N+1]$
**for** $d = 0$ **to** $D-1$ **do**
　　**for** $n_b = 1$ **to** $N_b^d$ **do**
　　　　$s = b_{n_b}^d : b_{n_b+1}^d - 1$ 　　(slice index)
　　　　$[\hat{\Lambda}_{s,s}]_{D-d}, [\hat{Q}_{s,s}]_{D-d}, b^{d+1} = \text{eigh}_1([\Lambda_{s,s}^d]_{d:})$
　　　　$[Q_{s,s}]_D = \text{qlift}([\hat{Q}_{s,s}]_{D-d}, D)$
　　**end**
　　$[\Lambda^{d+1}]_D = [\Lambda^d]_d + [\hat{\Lambda}]_{D-d} T^d$
　　$[Q^{d+1}]_D = [Q^d]_D [Q]_D$
**end**

**Algorithm 6**: This algorithm computes $[\Lambda]_D, [Q]_D = \text{eigh}([A]_D)$ as described in Theorem 3. The algorithm uses internally Algorithm 5 and 4.

# 8 Numerical tests and examples

## 8.1 Taylor polynomial arithmetic on real symmetric eigenvalue problem

As an example to test the validity of the pushforward in UTP arithmetic we consider the following system [AT98]:

$$Q(t) = \frac{1}{\sqrt{3}} \begin{pmatrix} \cos(x(t)) & 1 & \sin(x(t)) & -1 \\ -\sin(x(t)) & -1 & \cos(x(t)) & -1 \\ 1 & -\sin(x(t)) & 1 & \cos(x(t)) \\ -1 & \cos(x(t)) & 1 & \sin(x(t)) \end{pmatrix}$$

$$\Lambda(t) = \mathrm{diag}(x^2 - x + \frac{1}{2}, 4x^2 - 3x, \delta(-\frac{1}{2}x^3 + 2x^2 - \frac{3}{2}x + 1) + (x^3 + x^2 - 1), 3x - 1),$$

where $x \equiv x(t) := 1 + t$. The constant $\delta$ is some predefined constant. In Taylor arithmetic one obtains

$$\begin{aligned}
\Lambda_0 &= \mathrm{diag}(1/2, 1, 1 + \delta, 2) \\
\Lambda_1 &= \mathrm{diag}(1, 5, 5 + \delta, 3) \\
\Lambda_2 &= \mathrm{diag}(2, 8, 8 + \delta, 0) \\
\Lambda_3 &= \mathrm{diag}(0, 0, 6 - 3\delta, 0) \\
\Lambda_d &= \mathrm{diag}(0, 0, 0, 0), \quad \forall d \geq 4 .
\end{aligned}$$

One can see that in the case $\delta = 0$ one obtains one repeated eigenvalue that splits at $d = 3$. We apply Algorithm 6 to reconstruct $[\Lambda]_D$. The reconstructed values are denoted $[\tilde{\Lambda}]_D$. The numerical results are shown in Fig. (8.1).
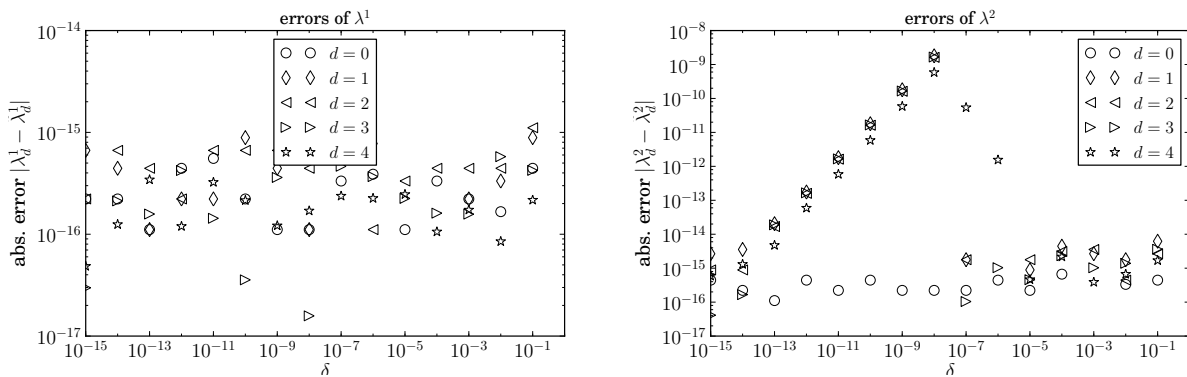


Figure 8.1: One left side one can see that error between the true $\lambda_1$ and reconstructed $\tilde{\lambda}_1$ is close to machine precision. On the right side one can see that the absolute error $|\tilde{\lambda}_2 - \lambda_2|$ has a jump at $\delta \approx 10^{-7}$. This is due to the fact that that the algorithm treats eigenvalues $|\lambda_i - \lambda_j| < 10^{-7}$ as repeated eigenvalues. One can see that when $\delta$ approaches $10^{-16}$ the error gets smaller. The eigenvalue $\lambda_4$ shows the same qualitative behavior as $\lambda_1$ and $\lambda_3$ the same as $\lambda_2$.

## 8.2 Covariance matrix computation

To test the validity of the covariance matrix computation of Eqn. (2.1) we first check that the first directional derivatives of the covariance matrix $C$ w.r.t. $J_1$ and $J_2$ coincide with the results from the complex-step derivative approximation, abbreviated here for convenience as CSDA. The CSDA computes directional derivatives of a real-valued function $y = f(x)$ as $\dot{y} \approx \frac{\Im(f(x+i\varepsilon\dot{x}))}{\varepsilon} = \frac{f(x+i\varepsilon\dot{x})-f(x-i\varepsilon\dot{x})}{2i\varepsilon}$, i.e., $\Im$ extracts the imaginary part and $i \equiv \sqrt{-1}$. The number $\varepsilon \in \mathbb{R}$ can be made very small. For a detailed discussion that also shows the relation to AD see Martins et al. [MSA01, MSA03]. Having verified the first order derivatives by UTP arithmetic we can check if the UTP arithmetic on Eqn. (2.2) yields the same result. Unfortunately, it is not possible to use the CSDA in a straight-forward fashion since for complex matrices the QR decomposition does not yield an orthogonal but a unitary $Q$. For reproducibility we define $J_1$ and $J_2$ rather arbitrarily as

$$J_1(x) = \begin{pmatrix} \sin(x_1)x_2 & \cos(x_2) \\ \exp(x_1) & x_1 x_2 \\ x_1 \log(x_2) & \log(1+\exp(\cos(x_1))) \\ x_2 + x_1 & x_1(x_2 + \cos(x_1)) \end{pmatrix}, \quad J_2(x)^T = \begin{pmatrix} x_1 \log(x_2 + 3\sin(x_1 x_2)) \\ x_2 \exp(\sin(x_1) + \cos(x_1 x_2)) \end{pmatrix}.$$

The numerical results are shown in Figure 8.2. Note that $x_1$ and $x_2$ are here elements of the vector $x$ and not coefficients.
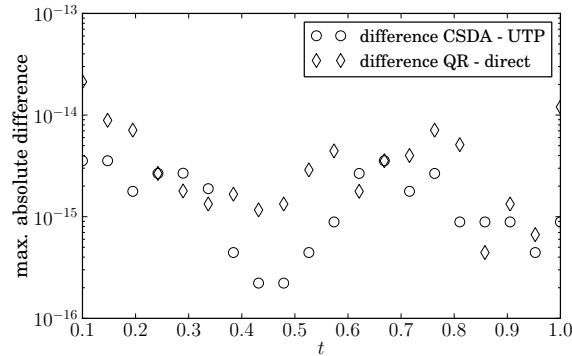


Figure 8.2: This plot shows the maximum absolute differences of the directional derivatives at $x = t(3,1)^T$, where $t \in [0.1, 1]$ in direction $\dot{x} = (5,7)^T$. The circles show the difference between the CSDA solution and the first order UTP solution using Eqn. (2.1). The diamonds show the difference between the UTP solution of Eqn. (2.1) and Eqn. (2.2). One can see that the difference is close to machine precision of IEEE 754 float64, which is approximately $10^{-16}$.

# 9  Summary and outlook

We have shown how computer codes containing real symmetric eigenvalue decompositions and QR decompositions can be evaluated in univariate Taylor polynomial arithmetic. Furthermore, the reverse mode of AD has been treated. Explicit algorithms have been presented that can be used in combination with existing AD software, e.g. general purpose AD tools like ADOL-C [GJM+99] or CppAD [Bel10] but also differentiated DAE solvers like SolvIND [AK]. Numerical tests have been used to check the algorithms.

Other algorithms that contain the the QR decomposition and the real symmetric eigenvalue decompostion can be differentiated using the shown algorithms. In particular, we think of the differentiation of the SVD or the computation of pseudoinverses. We believe that these algorithms, in modified form, may also be valuable for the eigenvalue optimization problem where eigenvalues are repeated in the solution point.

## Acknowledgements

## A   Additional proofs

### A.1   Proofs of QR decomposition

#### A.1.1   Proof of Proposition 1

*Proof.* We look at the first defining equation and try to separate the known from the unknown quantities:

$$
\begin{aligned}
0 \; &\stackrel{D+E}{=} \; [Q]_{D+E}[R]_{D+E} - [A]_{D+E} \\
&\stackrel{D+E}{=} \; ([Q]_D + [\Delta Q]_E T^D)([R]_D + [\Delta R]_E T^D) - [A]_{D+E} \\
&\stackrel{D+E}{=} \; [Q]_D[R]_D - [A]_{D+E} + ([\Delta Q]_E[R]_D + [Q]_D[\Delta R]_E)T^D \\
&\stackrel{D+E}{=} \; -[\Delta F]_E T^D + ([\Delta Q]_E[R]_D + [Q]_D[\Delta R]_E)T^D \\
&\stackrel{E}{=} \; -[\Delta F]_E + [\Delta Q]_E[R]_E + [Q]_E[\Delta R]_E \; .
\end{aligned}
\tag{A.1}
$$

Similarly for the second defining equation

$$
\begin{aligned}
0 \; &\stackrel{D+E}{=} \; [Q]_{D+E}^T[Q]_{D+E} - \mathbf{I} \\
&\stackrel{D+E}{=} \; [Q]_D^T[Q]_D - \mathbf{I} + ([Q]_D^T[\Delta Q]_E + [\Delta Q]_E^T[Q]_D)T^D \\
\Rightarrow 0 \; &\stackrel{E}{=} \; -[\Delta G]_E + [Q]_E^T[\Delta Q]_E + [\Delta Q]_E^T[Q]_E \\
&\stackrel{E}{=} \; -[\Delta G]_E + [S]_E + [X]_E + [S]_E - [X]_E \\
\Rightarrow \quad S \; &= \; \frac{1}{2}[\Delta G]_E \; ,
\end{aligned}
$$

where $[S]_E + [X]_E = [Q]_E^T[\Delta Q]_E$ and it has been used that every matrix can be written as the sum of a symmetric and an antisymmetric matrix. Now multiply (A.1) by $[Q]_E^T$ from the left to obtain

$$0 \overset{E}{=} -[Q]_E^T[\Delta F]_E + [Q]_E^T[\Delta Q]_E[R]_E + [\Delta R]_E \tag{A.2}$$
$$\overset{E}{=} -[Q]_E^T[\Delta F]_E + ([S]_E + [X]_E)[R]_E + [\Delta R]_E \tag{A.3}$$
$$\overset{E}{=} -[Q]_E^T[\Delta F]_E + [S]_E[R]_E + [X]_E[R]_E + [\Delta R]_E .$$

Multiplication of $[R_{:N,:}]_E^{-1}$ from the right yields

$$0 \overset{E}{=} -[Q]_E^T[\Delta F]_E[R_{:N,:}]_E^{-1} + [S]_E[R]_E[R_{:N,:}]_E^{-1} + [X]_E[R]_E[R_{:N,:}]_E^{-1} + [\Delta R]_E[R_{:N,:}]_E^{-1}$$
$$\overset{E}{=} -[Q]_E^T[\Delta F]_E[R_{:N,:}]_E^{-1} + [S_{:,:N}]_E + [X_{:,:N}]_E + [\Delta R]_E[R_{:N,:}]_E$$
$$\Rightarrow P_L \circ ([X_{:,:N}]_E) \overset{E}{=} P_L \circ \left([Q]_E^T[\Delta F]_E[R_{:N,:}]_E^{-1} - [S_{:,:N}]_E\right) .$$

The coefficients of $X_{:,N+1:}$ are not specified and can for instance be set to zero. Since $X$ is antisymmetric it is already defined by the above equation. Since $[S]_E + [X]_E \overset{E}{=} [Q]_E^T[\Delta Q]_E$ one can obtain $[\Delta Q]_E$ as

$$[\Delta Q]_E = [Q]_E([S]_E + [X]_E)$$

because for quadratic $Q$ one has the identity $QQ^T = \mathbf{I}$. □

### A.1.2   Proof of Proposition 2

*Proof.* We differentiate the implicit system

$$0 = A - QR$$
$$0 = Q^TQ - \mathbf{I}$$
$$0 = P_L \circ R$$

and obtain

$$0 = dA - dQR - QdR \quad (*)$$
$$0 = dQ^TQ + Q^TdQ \quad (**) .$$

We define the antisymmetric "matrix" $X := Q^TdQ$. Multiplication of Eqn. (*) from the left with $Q^T$ yields

$$0 = Q^TdA - XR - dR$$
$$\text{hence} \quad dR = Q^TdA - XR .$$

The multipication of this last equation from the right with the Moore-Penrose pseudoinverse $R^+ = (R_{:N,:}^{-1}, 0)$ yields the equivalent equation

$$0 = Q^TdAR^+ - XRR^+ - dRR^+$$
$$\text{and thus} \quad P_L \circ X = P_L \circ (Q^TdAR^+) ,$$

where we have chosen arbitrarily that $X_{:,N+1:} = 0$. Since $X$ is antisymmetric we have

$$X = (P_L \circ X) - (P_L \circ X)^T .$$

We can use these results to compute the pullback:

$$
\begin{aligned}
\operatorname{tr}(\bar{R}^T \, \mathrm{d}R) + \operatorname{tr}(\bar{Q}^T \, \mathrm{d}Q) &= \operatorname{tr}(Q\bar{R}\mathrm{d}A^T) - \operatorname{tr}(R\bar{R}^T X) + \operatorname{tr}(\bar{Q}^T Q Q^T \, \mathrm{d}Q) \\
&= \operatorname{tr}(Q\bar{R}\mathrm{d}A^T) + \operatorname{tr}(\underbrace{(\bar{Q}^T Q - R\bar{R}^T)}_{=:K} X) \\
&= \operatorname{tr}(Q\bar{R}\mathrm{d}A^T) + \operatorname{tr}((K - K^T)(P_L \circ X)) \\
&= \operatorname{tr}(Q\bar{R}\mathrm{d}A^T) + \operatorname{tr}(R^{+T}\mathrm{d}A^T Q(P_L \circ (K^T - K))) \\
&= \operatorname{tr}(Q[\bar{R} + \{P_L \circ (Q^T \bar{Q} - \bar{Q}^T Q + R\bar{R}^T - \bar{R}R^T)\}R^{+T}]\mathrm{d}A^T) \\
&= \operatorname{tr}(\bar{A}\mathrm{d}A^T) .
\end{aligned}
$$

In the above derivation we have used Lemmas 7, 8 and 9.

$\square$

### A.1.3  Proof of Lemma 5

*Proof.*

$$
\begin{aligned}
0 \quad &\overset{D+E}{=} \quad ([Q]_D + [\Delta Q]_E T^D)^T([Q]_D + [\Delta Q]_E T^D) - \mathbf{I} \\
&\overset{D+E}{=} \quad ([Q]_D^T [Q]_D - \mathbf{I}) + ([Q]_D^T [\Delta Q]_E + [\Delta Q]_E^T [Q]_D)T^D \\
&\overset{E}{=} \quad [\Delta G]_E + [Q]_E^T [\Delta Q]_E + [\Delta Q]_E^T [Q]_E \\
&\overset{E}{=} \quad [\Delta G]_E + 2[S]_E \\
[\Delta Q]_E \quad &\overset{E}{=} \quad -\frac{1}{2}[Q]_E[\Delta G]_E ,
\end{aligned}
$$

where $[\Delta Q]_E^T [Q]_E = [S]_E + [X]_E$, $[S]_E$ symmetric and $[X]_E$ antisymmetric and $[\Delta G_E]_E T^D \overset{D+E}{=} (Q^T Q - \mathbf{I})$.
Since no condition defines constraints on $[X]_E$ it has been set to zero. $\square$

### A.1.4  Proof of Proposition 6

*Proof.* We want to compute $\operatorname{tr}(\bar{A}^T \, \mathrm{d}A) = \operatorname{tr}(\bar{\Lambda}^T \, \mathrm{d}\Lambda) + \operatorname{tr}(\bar{Q}^T \, \mathrm{d}Q)$. We differentiate the implicit system

$$
\begin{aligned}
0 &= Q^T A Q - \Lambda \\
0 &= Q^T Q - \mathbf{I} \\
0 &= (P_L + P_R) \circ \Lambda
\end{aligned}
$$

and obtain

$$
\begin{aligned}
\mathrm{d}\Lambda &= Q^T \, \mathrm{d}A Q + \mathrm{d}Q^T A Q + Q^T A \, \mathrm{d}Q \\
&= Q^T \, \mathrm{d}A Q + \mathrm{d}Q^T Q\Lambda + \Lambda Q^T \, \mathrm{d}Q \\
0 &= \mathrm{d}Q^T Q + Q^T \, \mathrm{d}Q .
\end{aligned}
$$

A straight forward calculation shows:

$$
\begin{aligned}
\text{tr}\,(\bar{\Lambda}^T d\Lambda) &= \text{tr}\,(Q\bar{\Lambda}Q^T dA) + \text{tr}\,(\Lambda\bar{\Lambda}dQ^T Q) + \text{tr}\,(\bar{\Lambda}\Lambda Q^T dQ) \\
&= \text{tr}\,(Q\bar{\Lambda}Q^T dA)\,, \\
\text{tr}\,(\bar{Q}^T dQ) &= \text{tr}\,(\bar{Q}^T Q(H \circ (Q^T dAQ))) \\
&= \text{tr}\,(Q(H^T \circ (\bar{Q}^T Q))Q^T dA)\,, \\
\text{tr}\,(\bar{A}^T dA) &= \text{tr}\,\big((Q(\bar{\Lambda}+H^T \circ (\bar{Q}^T Q))Q^T)dA\big)
\end{aligned}
$$

where we have used

$$
\begin{aligned}
d\Lambda &= Q^T dAQ - (Q^T dQ)^T \Lambda + \Lambda Q^T dQ \\
&= Q^T dAQ - K \circ (Q^T dQ) \\
\implies Q^T dQ &= H \circ (Q^T dAQ - d\Lambda) \\
&= H \circ (Q^T dAQ)
\end{aligned}
$$

where we have defined $K_{ij} := \Lambda_{jj} - \Lambda_{ii}$ and $H_{ij} = (K_{ij})^{-1}$ for $i \neq j$ and $H_{ij} = 0$ otherwise and used the property $X\Lambda - \Lambda X = K \circ X$ for all $X \in \mathbb{R}^{N \times N}$ and diagonal $\Lambda \in \mathbb{R}^{N \times N}$. $\qquad\square$

## A.2   Basic results used in the proofs

**Lemma 7.** *Let $X \in \mathbb{R}^{N \times N}$ be an antisymmetric matrix, i.e., $X^T = -X$ and $P_L$ defined as above. We then can write*

$$
X = P_L \circ X - (P_L \circ X)^T\,. \tag{A.4}
$$

*Proof.* $X = P_L \circ X + P_R \circ X = P_L \circ X + (P_L \circ X^T)^T = P_L \circ X - (P_L \circ X)^T$ $\qquad\square$

**Lemma 8.** *Let $A \in \mathbb{R}^{N \times N}$ and $P_L$ resp. $P_R$ defined as above. Then*

$$
(P_L \circ A)^T = P_R \circ A^T\,. \tag{A.5}
$$

*Proof.* $B_{ij} := (P_L \circ A)_{ij} = A_{ij}(i > j)$ and $B_{ij}^T = B_{ji} = A_{ji}(j > i) = A_{ij}^T P_R = P_R \circ A$ $\qquad\square$

**Lemma 9.** *Let $A, B, C \in \mathbb{R}^{M \times N}$. We then have*

$$
\text{tr}\,\big(A^T(B \circ C)\big) = \text{tr}\,\big(C^T(B \circ A)\big) \tag{A.6}
$$

*Proof.* $\text{tr}\,(A^T(B \circ C)) = \sum_{i=1}^{N} \sum_{j=1}^{M} A_{ij} B_{ij} C_{ij} = \text{tr}\,(C^T(B \circ A))$ $\qquad\square$

**Lemma 10.** *Let $A, B$ be lower triangular matrices. Then the following expression holds:*

$$
P_D \circ (AB) = (P_D \circ A)(P_D \circ B)\,. \tag{A.7}
$$

*Proof.* $AB$ is also lower trinangular and thus $P_D \circ (AB) = diag(a_{ii}b_{ii}) = diag(a_{ii})diag(b_{ii}) = (P_D \circ A)(P_D \circ B)$ $\qquad\square$

**Lemma 11.** *The formula*

$$P_D \circ (A^T) \quad = \quad P_D \circ A \tag{A.8}$$

*holds for all quadratic matrices A.*

*Proof.* $(P_D \circ (A^T))_{ij} = \delta_{ij} A_{ji} = \delta_{ij} A_{ij} = (P_d \circ A)_{ij}$ □

**Lemma 12.** *Let A be a nonsingular quadratic lower triangular matrix. Then the formula*

$$P_D \circ (A^{-1}) \quad = \quad (P_D \circ A)^{-1} \tag{A.9}$$

*holds.*

*Proof.* Using Lemma 10 we obtain $(P_D \circ (A^{-1}))(P_D \circ A) = P_D \circ \mathbf{I} = \mathbf{I}$. Since the quadratic matrices form a group, the inverse is unique. Therefore, equality between $(P_D \circ (A^{-1})) = (P_D \circ (A))^{-1}$ must hold. □

**Lemma 13.** *Let $A \in \mathbb{R}^{N \times N}$ be strictly lower triangular and $B \in \mathbb{R}^{N \times N}$ lower triangular. Then their product $C = AB$ is strictly lower triangular.*

*Proof.* C is lower triangular and the diagonal entries are $C_{ii} = A_{ii} B_{ii} = 0$ since B has a zero diagonal. □

**Corollary 14.** *Let $A \in \mathbb{R}^{N \times N}$ be strictly lower triangular and $D \in \mathbb{R}^{N \times N}$ diagonal. Then their product $C = AD$ is strictly lower triagonal.*

**Lemma 15.** *Every quadratic matrix A can be written as the sum of a symmetric matrix $S = \frac{1}{2}(A + A^T)$ and an antisymmetric matrix $X = \frac{1}{2}(A - A^T)$, i.e.*

$$A \quad = \quad S + X \tag{A.10}$$

*Proof.* $A = \frac{1}{2}(A + A^T + A - A^T) = \frac{1}{2}(A + A^T) + \frac{1}{2}(A - A^T) = S + X.$ □

# References

[AK]      Jan Albersmeyer and C. Kirches. The SolvIND webpage. http://www.solvind.org.

[AT98]    Alan L. Andrew and Roger C. E. Tan. Computation of derivatives of repeated eigenvalues and the corresponding eigenvectors of symmetric matrix pencils. *SIAM Journal on Matrix Analysis and Applications*, 20(1):78–100, 1998.

[Bel10]   B. Bell. Cppad: a package for c++ algorithmic differentiation. http://www.coin-or.org/CppAD, 2010.

[Ber01]   Daniel J. Bernstein. Multidigit multiplication for mathematicians, 2001.

[Ber08]   Daniel J. Bernstein. Fast multiplication and its applications. *Algorithmic Number Theory, Lattices, Number Fields, Curves and Cryptography*, 2008.

[BH96]     Christian H. Bischof and Mohammad R. Haghighat. Hierarchical approaches to automatic differentiation. In Martin Berz, Christian Bischof, George Corliss, and Andreas Griewank, editors, *Computational Differentiation: Techniques, Applications, and Tools*, pages 83–94. SIAM, Philadelphia, PA, 1996.

[Bjö96]    Å. Björck. *Numerical Methods for Least Squares Problems*. SIAM, Philadelphia, 1996.

[Büc02]    H. M. Bücker. *Hierarchical Algorithms for Automatic Differentiation*. Habilitationsschrift, Aachen, nov 2002.

[Chr91]    Bruce Christianson. Reverse accumulation and accurate rounding error estimates for Taylor series coefficients. *Optimization Methods and Software*, 1(1):81–94, 1991. Also appeared as Tech. Report No. NOC TR239, The Numerical Optimisation Centre, University of Hertfordshire, U.K., July 1991.

[Chr98]    Bruce Christianson. Reverse accumulation and implicit functions. *Optimization Methods and Software*, 9(4):307–322, 1998.

[Eri03]    Eric Todd Phipps. *Taylor Series Integration of Differential-Algebraic Equations: Automatic Differentation as a Tool For Simulationg Rigid Body Mechanical Systems*. PhD thesis, Cornell University, February 2003.

[GJM⁺99]   Andreas Griewank, David Juedes, H. Mitev, Jean Utke, Olaf Vogel, and Andrea Walther. ADOL-C: A package for the automatic differentiation of algorithms written in C/C++. Technical report, Institute of Scientific Computing, Technical University Dresden, 1999. Updated version of the paper published in *ACM Trans. Math. Software* 22, 1996, 131–167.

[Gri03]    Andreas Griewank. A mathematical view of automatic differentiation. In *Acta Numerica*, volume 12, pages 321–398. Cambridge University Press, 2003.

[GUW00]    Andreas Griewank, Jean Utke, and Andrea Walther. Evaluating higher derivative tensors by forward propagation of univariate Taylor series. *Mathematics of Computation*, 69:1117–1130, 2000.

[GVL96]    Gene H. Golub and Charles F. Van Loan. *Matrix computations (3rd ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996.

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

[KKSB07]   S. Körkel, E. Kostina, J. P. Schlöder, and H. G. Bock. *Robustness Aspects in Parameter Estimation, Optimal Design of Experiments and Optimal Control*. Springer Berlin Heidelberg, 2007.

[Kör02]    Stefan Körkel. *Numerische Methoden für Optimale Versuchsplanungsprobleme bei nichtlinearen DAE-Modellen*. PhD thesis, Universitat Heidelberg, 2002.

[Mär02]    R. März. The index of linear differential algebraic equations with properly stated leading terms. *Result. Math.*, 42:308–338, 2002.

[Mär03]      R. März. Differential Algebraic Systems with Properly Stated Leading Term and MNA Equations. In *Modeling, Simulation and Optimization of Integrated Circuits, International Series of Numerical Mathematics*, volume 146, pages 135–151, Basel, 2003. Birkhauser Verlag.

[MN99]       Jan R. Magnus and Heinz Neudecker. *Matrix differential calculus with applications in statistics and econometrics*. John Wiley & Sons, 2nd edition, 1999.

[MSA01]      Joaquim R. R. A. Martins, Peter Sturdza, and Juan J. Alonso. The connection between the complex-step derivative approximation and algorithmic differentiation, 2001.

[MSA03]      Joaquim R. R. A. Martins, Peter Sturdza, and Juan J. Alonso. The complex-step derivative approximation. *ACM Trans. Math. Softw.*, 29(3):245–262, 2003.

[vdAMM07]  N.P. van der Aa, H.G. Ter Morsche, and R.R.M. Mattheij. Computation of eigenvalue and eigenvector derivatives for a general complex-valued eigensystem. *Electronic Journal of Linear Algebra ISSN 1081-3810*, 16:300–314, 2007.