ROBUST MULTISECANT QUASI-NEWTON VARIANTS FOR PARALLEL FLUID-STRUCTURE SIMULATIONS—AND OTHER MULTIPHYSICS APPLICATIONS

KLAUDIUS SCHEUFELE† AND MIRIAM MEHL†

Abstract. Multisecant quasi-Newton methods have been shown to be particularly suited to solve nonlinear fixed-point equations that arise from partitioned multiphysics simulations where the exact Jacobian is inaccessible. In all these methods, the underdetermined multisecant equation for the approximate (inverse) Jacobian is enhanced by a norm minimization condition. The standard choice is the minimization of the Frobenius norm of the approximate inverse Jacobian. In this setting, it is well known that transient fluid-structure simulations typically require the use of secant information also from previous time steps to achieve a small enough number of iterations per implicit time step. The number of these time steps highly depends on the application, the physical parameters, the used solvers, and the mesh resolution. Using too few leads to a relatively high number of iterations, while using too many leads not only to a computational overhead but also to an increase in the number of iterations as well. Determining the optimal number requires a costly trial-and-error process. In this paper, we present results for two different approaches to overcome this issue: The first approach is to use a modified method (presented in [F. Lindner, M. Mehl, K. Scheufele, and B. Uekermann, “A Comparison of Various Quasi-Newton Schemes for Partitioned Fluid-Structure Interaction,” in Proceedings of ECCOMAS Coupled Problems, Venice, 2015, pp. 1–12]) that minimizes the Frobenius norm of the difference between the current approximate (inverse) Jacobian and that of the previous time step. Thus, previous time step information is taken into account in an implicit and automated way without magic parameters. The second approach is to use an unrestricted number of previous time steps in combination with a suitable filtering algorithm automatically removing secant information that is outdated (thus, slowing down convergence) or contradicting newer information (deteriorating the condition of the multisecant equation system). We present a novel algorithm to realize the first idea with linear complexity in the number of coupling surface unknowns (note that already storing the approximate inverse Jacobian would induce quadratic complexity) and the efficient parallel implementation for both approaches. This results in highly efficient, parallelizable, and robust iterative solvers applicable for surface coupling in many types of multiphysics simulations using black-box solver software. In addition, our numerical results for the fluid-structure benchmark (FSI3) from Turek and Hron [“Proposal for Numerical Benchmarking of Fluid-Structure Interaction between an Elastic Object and Laminar Incompressible Flow,” in Fluid-Structure Interaction, Springer, Berlin, 2006, pp. 371–385] and for a flow through a flexible tube with a large variety of parameter settings prove the robustness and numerical efficiency of the first approach in particular. The second approach can be shown to be highly sensitive to the choice of the filtering method for the secant information and not always robust for the filters we currently use.

Key words. partitioned multiphysics, nonlinear fixed-point solver, quasi-Newton, fluid-structure interactions

AMS subject classifications. 65B99, 65H10, 65M99, 68W10, 74B99, 76D05

DOI. 10.1137/16M1082020

1. Introduction. Solving nonlinear systems of equations that are given only implicitly or even as a black-box functionality has become a common issue in multiphysics simulations, where interface equations depend on the discretization details of the single-physics domains, or in coupled optimization problems, where the task function is evaluated by several software components together. Let’s assume that we have two (logically coupled) solver components involved that provide routines to

---

*Received by the editors June 30, 2016; accepted for publication (in revised form) February 22, 2017; published electronically October 26, 2017.
†Institute for Parallel and Distributed Systems, Universität Stuttgart, Stuttgart, Germany (klaudius.scheufele@ipvs.uni-stuttgart.de, miriam.mehl@ipvs.uni-stuttgart.de).
evaluate a nonlinear problem in their respective field (i.e., evaluation of a single time step for a single-physics problem). These coupled problems induce mappings $S_1$ and $S_2$ which map elements of vector spaces $X_1$ and $X_2$ that describe data at the coupling interface to output values at the coupling interface. Hereby, $S_2$ requires as input the output computed by $S_1$, and vice versa, i.e., we have

\[
S_1: X_1 \rightarrow X_2 \quad \text{and} \quad S_2: X_2 \rightarrow X_1.
\]

Implicit interface coupling is realized via fixed-point equations at the interface. Depending on whether we execute the two solvers simultaneously or one after the other, this yields two different types of fixed-point equations that have to be fulfilled in each (implicit) time step:

\[
\begin{pmatrix}
0 & S_2 \\
S_1 & 0
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2
\end{pmatrix} =
\begin{pmatrix}
x_1 \\
x_2
\end{pmatrix}
\quad \text{or} \quad
S_2 \circ S_1(x_1) = x_1
\]

(Jacobi-type system) \quad (Gauß–Seidel-type system)

The Jacobi-type system may easily be ill-conditioned, as the coupling variables $x_1 \in X_1$ and $x_2 \in X_2$, concatenated as $x = (x_1, x_2)^T$, are likely to live on very different scales. In order to get a good conditioned problem, we have to apply a proper weighting and solve the resulting scaled system:

\[
\begin{pmatrix}
0 & \lambda^{S_2} S_2 \\
\lambda^{S_1} S_1 & 0
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2
\end{pmatrix} =
\begin{pmatrix}
\lambda^{S_2} x_1 \\
\lambda^{S_1} x_2
\end{pmatrix}.
\]

The weights $\lambda^{S_1}$ and $\lambda^{S_2}$ are chosen such that the subvectors $x_1$ and $x_2$ are normalized with respect to the previous time step.\footnote{Uekermann showed in [29] through exhaustive testing that the per-subvector normalization outperforms the per-entry normalization that would result in a weighting factor for each entry in $x$.} Uekermann [29] proposes a series of normalization ideas to obtain the weighting factors, where the so-called residual-sum approach shows the best results. Here, the subvectors $x_1$ and $x_2$ are normalized with respect to the sum of the residuals from $k$ previous quasi-Newton iterations; i.e., we get weighting factors

\[
\lambda^{S_1} = \left( \sum_{j=1}^{k} \frac{\|S_2(x_1^j) - x_1^j\|_2}{\|H(x^j) - x^j\|_2} \right)^{-1} \quad \text{and} \quad
\lambda^{S_2} = \left( \sum_{j=1}^{k} \frac{\|S_1(x_2^j) - x_2^j\|_2}{\|H(x^j) - x^j\|_2} \right)^{-1}.
\]

In the following, we focus on the Jacobi-type system (J-system) as it allows for an efficient use of massively parallel systems [23]. Nonetheless, all methods and concepts can be applied to basically any type of fixed-point equation. Thus, all methods are described for a general fixed-point equation

\[
H(x) = x \quad \text{with the residual} \quad R(x) := H(x) - x = 0,
\]

where $H : \mathbb{R}^N \rightarrow \mathbb{R}^N$ is an operator that we are able to apply; i.e., we assume that we have a simulation environment that allows us to evaluate $H(x)$ for a given input $x$.

In the literature, two basic solution approaches for (3) are discussed: (accelerated) fixed-point iterations (also referred to as Anderson mixing [1, 14, 22, 27, 32]) and quasi-Newton methods [12, 15, 19, 24, 25, 30, 31]. Both approaches have been
developed independently from each other but can be shown to be in large part equivalent. Basically, they are generalizations of GMRES for nonlinear problems. The common ingredient of all methods is the approximation of the (inverse) Jacobian matrix based on (1) a multisecant approach using input and output data of the involved modules over several solver iterations, and (2) a norm minimization condition. In particular for time-dependent problems requiring the solution of a nonlinear equation in each time step, using information from previous time steps can substantially improve the convergence. This can be done in two fundamentally different ways: The first possibility is to explicitly include input and output data from previous time steps in the multisecant equation. This induces the unknown parameter number of reused time steps $\xi$ which is highly problem-dependent and can only be determined in a costly trial-and-error process. Both too large and too small $\xi$ lead to an unacceptable convergence behavior. The other possibility is to minimize the norm of the difference of subsequent (inverse) Jacobian approximations. This implicit reuse of old information results in very robust and fast converging solvers, but requires the explicit calculation and storage of the Jacobian matrix, which is not necessary otherwise. The costs for this are prohibitively large in most applications. Therefore, we present novel ideas to overcome the drawbacks of each of the two alternatives:

1. We enhanced the standard approach with explicit reuse of previous time step information in the secant equation by a filtering approach that should filter out previous time step information that is either outdated—and thus causing slow convergence of the quasi-Newton iterations—or contradicting with newer information—and thus leading to ill-conditioned secant equation systems. In contrast to the usually used combination of reuse and filtering (e.g., in [18]), our idea is to choose $\xi$ arbitrarily large and to let the filtering take care of choosing a suitable selection from the collected secant information. If this works, it eliminates the costly trial-and-error determination of the optimal number of reused time steps.

2. We developed and implemented a linear complexity algorithm for the variant using norm minimization for the difference between the current and the previous time step’s approximate inverse Jacobian. It is based on storing the approximate inverse Jacobian in a decomposed form, low-rank updates for singular value decomposition (SVD), and truncation of SVDs. The novelty here is the substantial reduction of complexity while maintaining the robustness of the original method.

In section 2, we give an overview of multisecant quasi-Newton methods for general nonlinear fixed-point problems and present a few numerical results showing the effects of reusing previous time step information for two fluid-structure interaction scenarios. The new variants based on filtering and difference norm minimization are presented in section 3, together with their efficient algorithmic realization and parallelization. Section 4 shows numerical and scalability results for further fluid-structure interaction scenarios and a detailed comparison of the two new methods.

2. Quasi-Newton and Anderson acceleration. A large range of problems boils down to the solution of a nonlinear fixed-point equation (3) where we are able to evaluate $H(x)$ for a given input $x$, yet no further details of $H$ are known. We additionally assume that the evaluation of $H$ is very expensive such that minimizing the number of iterations is crucial. In recent years, two different communities have developed methods for this problem: so-called quasi-Newton methods have been used in particular in the context of fluid-structure interactions [12, 19, 20, 24, 25], whereas
variants of Anderson mixing [1] have been developed in a more general context [14, 22, 27, 32]. We introduce the general idea from a quasi-Newton point of view. A
Newton step for (3) reads
\begin{equation}
x^{k+1} = x^k - J_{\tilde{R}}^{-1}R(x^k) = x^k - (J_{\tilde{R}}^{-1} - I)R(x^k),
\end{equation}
with \( \tilde{R} := I - H^{-1} = R \circ H^{-1} \) and the reasoning that \( H - R = I \) and, thus, \( R^{-1} = (H - R) \circ R^{-1} = \tilde{R}^{-1} - I \). In the following subsection, we give an overview of quasi-Newton variants discussed in literature to approximate the Jacobian \( J_{\tilde{R}} \) or its inverse \( J_{\tilde{R}}^{-1} \), respectively. We focus on the approximation of \( J_{\tilde{R}}^{-1} \), as this avoids the additional solution of a linear system per quasi-Newton iteration and can be shown to be more stable [26]. The overview is certainly not complete and presents no details but only the basic underlying ideas of the various method classes.

2.1. Overview quasi-Newton methods for general fixed-point equations.
All methods are based on matrices storing the response in output differences depending on input differences of the function \( \tilde{R}(\tilde{x}^k) := \tilde{x}^k - H^{-1}(\tilde{x}^k) = R(x^k) = \tilde{x}^k \), with \( \tilde{x}^k := H(x^k) \):
\begin{align*}
W_k &= [\Delta\tilde{x}_j, \Delta\tilde{x}_{j+1}, \ldots, \Delta\tilde{x}_{k-1}], \quad \text{with} \quad \Delta\tilde{x}_i = \tilde{x}^k - \tilde{x}^i,
V_k &= [\Delta\tilde{r}_j, \Delta\tilde{r}_{j+1}, \ldots, \Delta\tilde{r}_{k-1}], \quad \text{with} \quad \Delta\tilde{r}_i = R(x^k) - R(x^i),
\end{align*}
where \( k \) is the number of iterations done so far, and \( j \) is a method-dependent number between zero and \( k - 1 \). In the following, we use \( J^{-1} \) to denote the approximate inverse Jacobian of \( \tilde{R} \). The multisecant equation
\begin{equation}
J^{-1}V_k = W_k
\end{equation}
has to be fulfilled. This underdetermined system for the \( N^2 \) entries of \( J^{-1} \) needs suitable further restrictions formulated as norm minimization conditions for the Jacobian estimate or its difference to previous Jacobians. The concrete choice of these minimization conditions leads to different methods that we briefly outline:

Least squares (LS). The least-squares approach typically chooses \( j = 0 \), i.e., uses all previous iterations to build \( V_k \) and \( W_k \), and it completes (5) to a problem with a unique solution by the norm minimization condition
\begin{equation}
\|J^{-1}\| \rightarrow \min,
\end{equation}
where \( \| \cdot \| \) denotes the Frobenius norm. This leads to the approximation
\begin{equation}
J^{-1} = W_k V_k^+ \quad \text{with} \quad V_k^+ := (V_k^T V_k)^{-1}V_k^T
\end{equation}
of the inverse Jacobian with the pseudoinverse \( V_k^+ \) of \( V_k \) [20, 26, 31].

Broyden. Broyden’s methods [6] always uses \( j = k - 1 \), i.e., only information from a single iteration is used in the secant equation (5), which leads, together with the norm minimization
\begin{equation}
\|J^{-1,k} - J^{-1,k-1}\| \rightarrow \min
\end{equation}
for the difference between the updated Jacobian estimate \( J^{-1,k} \) and the previous iteration’s estimate \( J^{-1,k-1} \), to an iterative improvement of the inverse Jacobian approximation according to the formula
\begin{equation}
J^{-1,k} = J^{-1,k-1} + \frac{(W_k - J^{-1,k-1}V_k)V_k^T}{V_k^T V_k}.
\end{equation}
This method is known as bad Broyden’s method in contrast to the original good Broyden’s method, where both the secant equation and the norm minimization are formulated in terms of the Jacobian instead of its inverse. The natural choice for the initial approximation $J^{-1,0}$ is the zero matrix. With this, the first iteration according to (4) corresponds to a plain fixed-point iteration.

**Multivector update (MVJ).** The multivector-update method [4, 20, 26] can be interpreted as a generalization of Broyden’s method that replaces the rank-one update in (8) based on the previous iterate $J^{-1,k-1}$ of the Jacobian by a rank-$k$ update of an “initial” Jacobian approximation $J^{-1}_{prev}$. This is realized by choosing $j = 0$ and the norm minimization

$$\|J^{-1} - J^{-1}_{prev}\| \rightarrow \text{min}.$$  

Often, this method is used for time-dependent problems with a nonlinear fixed-point equation in every time step, which typically yields similar Jacobians in subsequent time steps. In the latter case, $J^{-1}_{prev}$ is the inverse Jacobian approximation from the previous time step. The inverse Jacobian is computed by a rank-$k$ update of $J^{-1}_{prev}$:

$$J^{-1} = J^{-1}_{prev} + (W_k - J^{-1}_{prev}V_k)V_k^\dagger.$$  

For the first time step, we use $J^{-1}_{prev} = 0$. Thus, the first time step actually uses the LS method.

**Successive rank-one Jacobian updates.** Haelterman showed in [17] that the approximate inverse Jacobian in (7) can be written as a sequence of rank-one updates,

$$J^{-1,k+1} - J^{-1,k} = A_MV_{k+1}V_{k+1}^\dagger - A_MV_kV_k^\dagger = A_M\bar{L}_{k+1}\bar{L}_k^T,$$

with $J^{-1,0} = 0$, with a linear mapping $A_M$ that fulfills $A_MV_{k+1} = W_{k+1}$, and with $\bar{L}_{k+1}$ denoting the last column of $L_{k+1}$, the matrix containing the orthonormalization of the columns of $V_{k+1}$. Without knowing the actual mapping $A_M$, one can compute

$$\bar{L}_{k+1} = \frac{(I - L_kL_k^T)\Delta r_{k+1}}{\|(I - L_kL_k^T)\Delta r_{k+1}\|} \text{ and } A_M\bar{L}_{k+1} = \frac{(J^{-1,k}-I)\Delta r_{k+1}}{\|(I - L_kL_k^T)\Delta r_{k+1}\|}.$$  

Generalizing (12) to the multivector-update formula yields

$$J^{-1,k+1} - J^{-1,k} = (A_M - J^{-1}_{prev})\bar{L}_{k+1}\bar{L}_k^T.$$  

As an initial guess for $J^{-1}$, $J^{-1,0} = J^{-1}_{prev}$ is used. Haelterman [17] proposes an improved least-squares method with old time step recovery using the least-squares method in rank-one update formulation (7). It also uses $J^{-1,0} = J^{-1}_{prev}$ and is similar, but not identical, to (13) as it lacks the term $J^{-1}_{prev}\bar{L}_{k+1}\bar{L}_k^T$ in the update.

**Anderson mixing.** Comparing [14, 32] with [12, 20, 25] shows that the quasi-Newton step executed with the Jacobian computed from the least-squares condition (7) is equivalent to type II Anderson acceleration, whereas type I Anderson acceleration is equivalent to a quasi-Newton method using an approximation of the Jacobian instead of its inverse. [20, 26] show that for fluid-structure interactions, our application example, the inverse Jacobian approximation is stable, which is not the case for the Jacobian approximation for nontrivial problems [26].
2.2. Convergence acceleration for transient problems by reusing old time steps. In a transient context, i.e., when we solve a fixed-point problem at every implicit time step, we approximate the Jacobian matrix at every time step. Since these Jacobian matrices usually do not differ much from one time step to the next, the convergence properties of the quasi-Newton methods improve considerably if input-output pairs of previous time steps are taken into account for the Jacobian approximation of the current time step [12]. This previous time step information enlarges the space spanned by the input vectors.2

To reuse previous time step information, the multisecant equation (5) is enhanced by reusing columns from previous time levels in the matrices $V_k$ and $W_k$, i.e.,

$$
V_k^\xi = [V_k, V_{k-\xi}, \ldots, V_{k-\xi+1}] \quad \text{and} \\
W_k^\xi = [W_k, W_{k-\xi}, \ldots, W_{k-\xi+1}],
$$

where $V_{k-i}$ and $W_{k-i}$, $(i = 1, \ldots, \xi)$ are the input and output matrices from the converged time step $n - i$ and $\xi$ determines the number of retained time levels. In practice, $\xi$ appears to be highly problem dependent and needs to be tuned carefully, as a nonoptimal choice may lead to either stagnation or numerical breakdown due to stability issues. As opposed to this explicit reuse of information within the multisecant equation, the Broyden and the MVJ methods implicitly incorporate previous information in the approximation of the system Jacobian via an initial guess $J_{i}^{-1}$ or $J_{prev}$, respectively, in the minimization condition (10). In other words, while fulfilling the multisecant equation (5) (with the latest information) exactly, these methods stay as close as possible to the Jacobian approximation from the previous iteration (Broyden) or the previous time step (MVJ). Thus, already well-resolved features are retained, and linearly dependent or even contradicting information is implicitly overwritten. Eventually, this ought to render the costly tuning step of the reuse parameter $\xi$ unnecessary, which, however, comes at the expense of storing an explicit representation of the inverse Jacobian matrix as it appears explicitly in the update formulas (8) or (11), respectively. We denote the LS method with explicit reuse of $\xi$ time steps in $V_k^\xi$ and $W_k^\xi$ by LS($\xi$).

3. Filtered LS and MVJ restart approach. The results presented in section 4.1 show the potential of LS and MVJ to efficiently solve fixed-point equations in coupled multiphysics simulations. However, each of the methods as presented in section 2 has a major drawback: The efficiency of LS strongly depends on the correct (problem- and solver-dependent) choice of the number of reused time steps. The only known way to determine this parameter is a very costly trial-and-error approach. For MVJ, we have shown robustness and good convergence without explicit reuse of previous time steps. This, however, has to be seen alongside the higher computational costs of the MVJ method compared to the LS approach. In this section, we propose ways to eliminate these drawbacks with the aim of constructing parameter-free and linear complexity quasi-Newton variants. To begin, we briefly summarize the computational components of both methods. The quasi-Newton iteration for the LS and

---

2In contrast to this reuse idea, creating additional input-output pairs within a time step implies a full time step execution in both involved solvers, i.e., more or less the costs of a full further coupling iteration, and is thus prohibitively costly.
MVJ methods, respectively, read

\begin{align}
    x^{k+1} &= H(x^k) - W_k V_k^r, \\
    x^{k+1} &= H(x^k) - (J_{\text{prev}}^{-1} + \tilde{W}_k V_k^r) r_k,
\end{align}

with the pseudoinverse \( V_k^\dagger = (V_k^T V_k)^{-1} V_k^T \). For both methods, we calculate a vector \( \alpha = V_k^\dagger (-r^k) \) and finally get the next quasi-Newton iterate from

\[ x^{k+1} = \tilde{x}^k + W_k \alpha \quad \text{and} \quad x^{k+1} = \tilde{x}^k - J_{\text{prev}}^{-1} r^k + (W_k - J_{\text{prev}}^{-1} V_k) \alpha \]

for the LS and MVJ methods, respectively. The pseudoinverse and the vector \( \alpha \), respectively, are computed efficiently from a QR-decomposition of \( V_k \) via backward substitution. See section 3.3 for details. The MVJ method in addition requires the calculation of \( \tilde{W}_k \) and, at the end of a time step, the computation and storage of the resulting Jacobian, to be used as \( J_{\text{prev}}^{-1} \) in the next time step. As \( J^{-1} \in \mathbb{R}^{N \times N} \), we have to carefully design the algorithms for these two components in order to avoid any \( \mathcal{O}(N^2) \) costs in terms of both storage and operations.

With a view to a robust and computationally cheap and efficient quasi-Newton acceleration method which in particular requires a minimum of parameter tuning, we propose two different, novel approaches:

1. We use the LS(\( \xi \)) method with a very large \( \xi \) and combine it with proper filtering. The hope is that filtering automatically extracts relevant and linearly independent columns from \( V_k \) and \( W_k \), thus making an optimal choice for \( \xi \) obsolete. Filtering is a way to automatically remove linear dependencies in the columns of \( V_k \) by removing columns during the QR-decomposition of \( V_k \) and has been proposed, e.g., in [18] to ensure robustness of multisection quasi-Newton with previous time step reuse. Our idea to use \( \xi \approx \infty \) pushes filtering to its limits in terms of eliminating columns in \( V_k \) and \( W_k \) that are no longer useful in the approximation of \( J^{-1} \).

2. We enhance the MVJ method with an information-retaining restart approach and efficient storage of the Jacobian. The basic idea of this approach is to avoid \( \mathcal{O}(N^2) \) complexity by explicitly storing all \( \tilde{W}_{k_i} \) and \( V_{k_i}^\dagger \) for a certain small number of time steps and to restart after this period by replacing the resulting sum representation of \( J^{-1} \) by a truncated (low-rank) singular-value decomposition. A sophisticated SVD-update algorithm allows us to do this with \( \mathcal{O}(N) \) complexity, as the rank of the truncated SVD does not increase with \( N \) (see section 4).

In the following, we recapitulate filtering methods known from the literature for the LS system (section 3.1) and propose a novel method for the efficient representation of the Jacobian based on a restart technique to reduce the complexity of the MVJ method (section 3.2) and the efficient parallel implementation of the algorithmic building blocks of both approaches (section 3.3).
occur at convergence of an iteration or for stagnating iterations. In particular, if reuse of information from previous time steps is considered, the risk of linearly dependent columns or even contradicting information grows drastically. To maintain a good condition of $V_k$ and, thus, improve the robustness of the quasi-Newton methods, potentially dangerous columns are deleted from $V_k$ and, accordingly, also in $W_k$. This process is called filtering. We do not introduce new filtering methods here, but recapitulate two different filtering approaches for the QR system proposed in [18] that we will analyze later to evaluate their potential in an LS($\xi$) with very large $\xi$. Both filters are based on the QR-decomposition of $V_k$ to indicate (nearly) linearly dependent columns. The first approach (QR1) was introduced in [12] and eliminates the $i$th column of $V_k$ and $W_k$ if
\begin{equation}
|R_{ii}| < \varepsilon_F \| R \|_2.
\end{equation}
Here, $R$ denotes the upper triangular matrix from the economy size QR-decomposition. This filtering is realized using an update scheme for the QR-decomposition; i.e., whenever a column is inserted or deleted, the already existing QR-decomposition is updated based on Givens rotations. While this approach is very efficient, it is probably less stable, as it tends to eliminate new iterates first, which contradicts the idea of filtering out too old information. However, in [29] Ueckermann showed that the difference between the update scheme and a version that recomputes the decomposition in each iteration, starting with the most recent information and thus eliminating old rather than new information, is negligible for an optimally tuned threshold $\varepsilon_F$ for the considered test problem.

The second filtering technique (QR2) compares and judges columns already during construction of the QR-decomposition with respect to their amount of new information incorporated into the least-squares system. This can be measured by the norm of a column of the orthogonalized (with respect to already computed columns of $Q$) columns of $V_k$. Here, it is crucial to start the orthogonalization with the most recent information, which, in turn, requires a recomputation of the QR-decomposition in each iteration. The completely filtered modified Gram–Schmidt orthogonalization for the QR2 filtering is given in Algorithm 1. Within this work, we do not present an exhaustive parameter study and comparison of the effects of the different filters but focus on their suitability to stabilize a least-squares system that is enriched by a lot of previous information. A more detailed description and examination of the different filtering techniques can be found in [18, 29].

Filtering is definitely required for LS($\xi$) with large values for $\xi$, but may also be required for MVJ, though it wasn’t for our examples in section 4. Several aspects, however, make filtering a much easier task for MVJ than for LS($\xi$): (1) the number of iterations is typically low, which drastically reduces the chance of linear dependencies in $V_k$ and $W_k$; (2) for MVJ, filtering is required only to remove linear dependencies from previous steps, and not also to judge implicitly on the relevance of information from those steps; (3) filtering and the resulting implication of a complete recalculation of the QR-decomposition are much cheaper for the low number of columns in $V_k$ for MVJ than for the “thicker” $V_k$ of the LS($\xi$) method.

3.2. Efficient representation of the inverse Jacobian $J^{-1}$—restart. $J^{-1}$ has a rank that is substantially smaller than $N$. This can be exploited to reduce the storage requirements and the computational costs of the MVJ method. We avoid building and storing the entire matrix but rather recompute and store the submatrices $\tilde{W}_k \in \mathbb{R}^{N \times k}$ and $V_k^\dagger \in \mathbb{R}^{k \times N}$ over several time steps. If $k_n$ is the number of iterations
**Algorithm 1** QR2-filtered modified Gram–Schmidt orthogonalization.

Data: $V_k = [\bar{v}_1, \bar{v}_2, \ldots, \bar{v}_k]$, threshold value $\varepsilon_F$  

Result: filtered $\hat{Q}$, $\hat{V}_k$, $\hat{W}_k \in \mathbb{R}^{N \times k}$, $\hat{R} \in \mathbb{R}^{k \times k}$ s. t. $\hat{V}_k = \hat{Q} \hat{R}$

set $\hat{R}_{11} = \|\bar{v}_1\|_2$ and $\hat{Q}_1 = \bar{v}_1/\hat{R}_{11}$

for $i = 2, \ldots, k$ do
  for $j = 1, 2, \ldots, i - 1$ do
    $R_{ji} = \hat{Q}^T_j \cdot v$
    $v = v - R_{ji} \cdot \hat{Q}_j$
    if $\|v\|_2 < \varepsilon_F \|\bar{v}_i\|_2$ then
      delete column $i$ from $\hat{V}_k$, $\hat{W}_k$, restart procedure
    else
      $\hat{R}_{ii} = \|v\|_2$ and $\hat{Q}_i = v/\hat{R}_{ii}$
    end
  end
end

in time step $n$, the inverse Jacobian approximation after $M$ time steps can be written as

$$J^{-1} = \hat{W}_k^T V_\xi^\dagger + \hat{W}_k^T V_{\xi - k}^\dagger + \cdots + \hat{W}_k^T V_{\kappa M}^\dagger .$$

Obviously, this approach is efficient only for $M$ substantially smaller than $N$. Therefore, we investigate different restart alternatives. At each restart, the inverse Jacobian estimation is restarted with new data. In other words, we divide the overall simulation into chunks of time steps that represent one era of the Jacobian estimation. Whether or not information from previous chunks can be retained in newer chunks depends on the restart policy. In the following, $M$ denotes an upper bound for the number of time steps per chunk, and $k$ an upper bound for the number of iterations per time step. We consider three different restart approaches:

**RS-0.** Clear all. This obviously results in $O(N \times k \times M)$ costs both for the storage of $J^{-1}$ in the form (17) and for the $M$ pairs of matrix-vector multiplications $y := V_{\kappa m}^\dagger x$ and $\hat{W}_{k m} y$ in $J^{-1} x$ for any vector $x \in \mathbb{R}^N$.

**RS-LS.** Clear $J^{-1}$, but keep columns in $V_{\xi RS} := [V_{\kappa n}, V_{\kappa n - 1}, \ldots, V_{\kappa n - \xi RS}]$ and $W_{\xi RS} := [W_{\kappa n}, W_{\kappa n - 1}, \ldots, W_{\kappa n - \xi RS}]$ from $\xi RS$ previous time steps within the current chunk; i.e., use the initial guess

$$J^{-1} := 0, \quad \hat{W}_0 := W_{\xi RS}, \quad \text{and} \quad V_0^\dagger := (V_{\xi RS}^T V_{\xi RS})^{-1} V_{\xi RS}^T .$$

If we reuse at most $\bar{K}$ columns, the total costs are $O(N \times \bar{K}) + O(N \times k \times M)$.

**RS-SVD.** Use a truncation of the singular value decomposition (SVD)

$$J^{-1} = \Psi \Sigma \Phi^T \quad \text{with} \quad \Sigma = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_N)$$

---

3The number of iterations for a particular time step is indicated by a subscript $k_n$. The subscript is dropped if the meaning of $k$ is obvious from the context.
as restart, where $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_N \geq 0$ are real singular values, and $\Psi \in \mathbb{R}^{N \times N}$ and $\Phi \in \mathbb{R}^{N \times N}$ are orthogonal matrices. We truncate this decomposition by cutting off all singular values below a given threshold; i.e., we restart with

$$J^{-1} = (\Psi \cdot j_{\min} \cdot j_{\max})^T,$$

where $\min = 1, \cdots, K =: \tilde{K}$.

$$=: \Sigma$$

The costs strongly depend on the efficient realization of the underlying SVD decomposition. Apart from this step, the total costs are $O(\tilde{K}^2) + O(N \times \tilde{K}) + O(N \times k \times M)$ if we truncate the SVD decomposition such that only $K$ values are left. For the efficient implementation of the SVD, we assume that we have a truncated SVD as in (18). At the end of the next chunk, our approximate inverse Jacobian reads

$$\Psi \Sigma \Phi^T + \sum_{m=1}^{M} \tilde{W}_k V_m^\dagger,$$

for which we have to compute an updated truncated SVD by computing updated SVD decompositions for each of the $M$ low-rank updates of the form

$$\Psi \Sigma \Phi^T + AB^T = \begin{bmatrix} \Psi & A \end{bmatrix} \begin{bmatrix} \Sigma & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} \Phi & B \end{bmatrix}^T,$$

with $A, B \in \mathbb{R}^{N \times k_m}$. We use the algorithm proposed in [5]; i.e., we compute the orthogonal components of $A$ and $B$. With the matrices $Q_A$ and $Q_B$ defining an orthonormal basis of the column space of $(I-\Psi \Psi^T)A$ and $(I-\Phi \Phi^T)B$, we define

$$R_A := Q_A^T(I-\Psi \Psi^T)A \quad \text{and} \quad R_B := Q_B^T(I-\Phi \Phi^T)B.$$  

With this, the update (20) can be transformed to

$$\Psi \Sigma \Phi^T + AB^T = \begin{bmatrix} \Psi \cdot Q_A \\ \Phi \cdot Q_B \end{bmatrix} S \begin{bmatrix} \Psi \cdot Q_B \end{bmatrix}^T,$$

with $S$ composed of

$$\begin{bmatrix} I & \tilde{\Psi}^T A \\ 0 & R_A \end{bmatrix} \begin{bmatrix} S & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} I & \tilde{\Phi}^T B \\ 0 & R_B \end{bmatrix}^T = \begin{bmatrix} \Sigma & 0 \\ 0 & R_A \end{bmatrix} + \begin{bmatrix} \tilde{\Psi}^T A \\ R_A \end{bmatrix} \begin{bmatrix} \tilde{\Phi}^T B \\ R_B \end{bmatrix}^T.$$  

Diagonalizing $S$ as $\Psi' S \Psi' = \Sigma'$ finally yields

$$\Psi \Sigma \Phi^T + AB^T = \begin{bmatrix} \Psi P \cdot \Psi' \end{bmatrix} \Sigma' \begin{bmatrix} \Psi \cdot Q \end{bmatrix} \Phi'^T.$$

After the update, the new SVD can be truncated again to keep a small but accurate representation. For the sake of clarity, the MVJ restart method RS-SVD as described in sections 3.2 and 3.3 is summarized in Algorithm 2.

**3.3. Efficient algorithmic realization and parallelization.** We briefly recapitulate the implementational aspects for an efficient algorithmic implementation of the basic building blocks for the two quasi-Newton methods from [8]. In particular, the focus lies on the algorithmic realization on distributed data. For an elaborate description, refer to [8]. Above all, we give details on the efficient realization of the newly introduced restart approach for the multivector method.
Algorithm 2 Pseudocode for the MVJ restart method RS-SVD as described in sections 3.2 and 3.3.

Data: chunk size $M$, threshold $\varepsilon$
\[
W, V^\dagger = \{ \}, \bar{V}, \bar{\Sigma}, \bar{\Psi} = 0, \eta = 0
\]
for $n = 0, 1, \ldots,$ do
\[
x^0 = \text{extr}(x_{n-1}^*, x_{n-2}^*, x_{n-3}^*)
\]
\[
\tilde{x}^0 = H(x^0) \quad \text{and} \quad r^0 = \tilde{x}^0 - x^0
\]
\[
x^1 = x^0 + \omega_0 \cdot r^0
\]
for $k = 1, 2, \ldots$ do
\[
\tilde{z}^k = H(x^k) \quad \text{and} \quad r^k = \tilde{z}^k - x^k
\]
if converged then
\[
1 \quad \text{break}
\]
end
\[
V_k = [\Delta x_i^k, \ldots, \Delta x_{k-1}^k] \quad \text{with} \quad \Delta x_i^k = R^i - r^k
\]
\[
W_k = [\Delta \tilde{x}_i^k, \ldots, \Delta \tilde{x}_{k-1}^k] \quad \text{with} \quad \Delta \tilde{x}_i^k = \tilde{x}^i - \tilde{x}^k
\]
\[
V_k^\dagger = (V_k^TV_k)^{-1}V_k^T \quad \text{via QR-dec. (24)}
\]
\[
W_k = W_k - \sum_{q=n-\eta}^n \tilde{W}_{k_q}(V_k^TV_k) \quad \text{via (26)}
\]
\[
\Delta x_{k+1} = - (\bar{\Psi}\Sigma) \cdot \bar{\Psi} - \sum_{q=n-\eta}^n \tilde{W}_{k_q}(V_k^TV_k^k)
\]
\[
x^k+1 = \tilde{z}^k + \Delta x^k+1
\]
end
store $\tilde{W} = [\tilde{W}_{kn}, \ldots, \tilde{W}_{k0}]$, $V^\dagger = [V_{kn}^\dagger, \ldots, V_{k0}^\dagger]$
\[
\eta = \eta + 1, t = t + \tau_n
\]
if $\eta > M$ then
\[
[\bar{\Psi}, \bar{\Sigma}, \bar{\Psi}] = \text{update-SVD}(\tilde{W}, V^\dagger, \varepsilon)
\]
clear $\tilde{W}, V^\dagger = []$
end end

Solving the least-squares system. The primary kernel of both quasi-Newton methods is the computation of the pseudoinverse $V_k^\dagger = (V_k^TV_k)^{-1}V_k^T$. Finding $V_k^\dagger y$ for a given vector $y$ is equivalent to solving the least-squares minimization
\[
(23) \quad z = \text{argmin}_{\tilde{z} \in \mathbb{R}^k} \|V_k \tilde{z} - y\|_2.
\]
This can be done very efficiently using a QR-decomposition of $V_k$ that exploits the fact that $V_k$ only grows by one column in each iteration [9].\footnote{Due to a possibly bad conditioning of the J-system, we have to solve the scaled system (2). This results in a modified least-squares problem, where the matrix $V_k$ as well as the right-hand-side vector $y$ need to be scaled accordingly, i.e., $z = \text{argmin}_{\tilde{z} \in \mathbb{R}^k} \|\Lambda_k V_k \tilde{z} - \Lambda_k y\|_2$. The weighting factors in $\Lambda_k$ change within each quasi-Newton iteration, which, in turn, requires a recomputation of the entire QR-factorization of the matrix $V_k$ per iteration; i.e., we compute $qr(\Lambda_k V_k) = \Lambda_k Q_k R_k$ and $
alpha = R_k^{-1}Q_k^\dagger \Lambda_k^{-1}(-\Lambda_k^{-1}k)$ for the LS method or $V_k^\dagger = (R_k^{-1}Q_k^\dagger \Lambda_k^{-1}k)\Lambda_k$ for the MVJ method in every iteration. After an initial phase, we keep the scaling factors constant, which allows us to exploit the updated QR-decomposition again.}

From the decomposition $V_k = QR$, we get $z$ from solving the quadratic system
\[
(24) \quad \hat{R} \tilde{z} = \hat{Q}^T y
\]
via backward substitution. $\hat{R} \in \mathbb{R}^{k \times k}$ denotes the first $k$ rows of $R$, and $\hat{Q}$ contains the first $k$ columns of $Q$ if $V_k \in \mathbb{R}^{N \times k}$. Although, for the MVJ method, we have
to compute the columns of the pseudoinverse $V_k^\dagger = (V_k^T V_k)^{-1} V_k^T$ via solving (24) for all unit vectors $e_i \in \mathbb{R}^n$, $i = 1,\ldots,N$, for the LS method it suffices to calculate the vector $z = \alpha = (V_k^T V_k)^{-1} V_k^T (\tilde{r}^k)$ using the right-hand side $y := \tilde{r}^k$. This updating scheme is particularly suited for a communication-avoiding and very efficient implementation on distributed data [21, 8]. For $p$ processors, $\hat{Q}$ is decomposed into row blocks corresponding to the interface data belonging to the respective surface partition; cf. Figure 1. This yields blocks of size $N/p \times k$ in the ideally balanced case. The matrix $\hat{R}$ is held on each processor as $k$ is assumably small. Thus, communication is only required for the dot-product and $l_2$-norm vector operations. The Givens rotations are applied fully locally on each processor. This yields a parallel computational complexity of

$$O(N/p \cdot k^2) + O(k^3) + O(k \log p)$$

to update an existing decomposition (adding or deleting a column from $V_k$). Solving the quadratic system (24) comprises computing $\hat{Q}^T y$ and the backward substitution step. For the former, each processor computes its additive contribution followed by an all-reduce step (runtime $O(N/p \cdot k) + O(k \log p)$), the latter is solved fully locally, leading to an accumulated parallel runtime complexity of

$$O(N/p \cdot k) + O(k \log p) + O(k^2)$$

for the LS method. The MVJ method requires the computation of $V_k^\dagger$, i.e., solving (24) for all unit vectors $e_i$, $i = 1,\ldots,N$. This accumulates to a parallel runtime of

$$O(N/p \cdot k^2)$$

as $\hat{Q}^T e_i$ comprises neither computational cost nor communication effort.

Computing the Newton update. For the LS method, $W_k \alpha$ can be computed locally on each processor without any communication in $O(N/p \cdot k)$ as $W_k$ is distributed in row blocks analogous to $\hat{Q}$. In the original MVJ method [8], the computation of the quasi-Newton update was rather involved, as it comprised three costly and complex matrix operations, namely, the computation of $\tilde{W}_k = W_k - J^{-1}_{\text{pre}} V_k$, the matrix product $\tilde{W}_k V_k^\dagger$, and the matrix-vector product $J^{-1} r^k$. This results in quadratic costs. Table 1 gives an overview on parallel runtimes for the mentioned matrix operations. As the novel restart approach directly addresses this issue, we do not go into detail for the realization of the original method. For more information, refer to [8].
\[ \tilde{W}_k = W_k - J_{\text{prev}}^{-1} V_k \]
\[
\begin{array}{c|c}
\text{Operation} & \text{Complexity} \\
\hline
\tilde{W}_k & O(N^2/p) + O(Nk \log p) \\
W_k V_k^\dagger & O(N^2/p) \\
J_{\text{prev}}^{-1} & O(N^2/p) + O(N \log p)
\end{array}
\]

As opposed to this, the quasi-Newton update for the new restart-based MVJ approach is calculated from a series of cheap matrix-vector operations,

\[ \Delta x^{k+1} = -\Psi \Sigma \Phi^T - \sum_{q=n-\eta}^n \tilde{W}_{k_q} \left( V_{k_q}^\dagger r^k \right), \]

where \( \eta \leq M \) is the number of computed time steps since the last restart. For the product \( \gamma = V_{k_q}^\dagger r^k \), each processor computes an additive contribution locally (cost \( O(N^2/p) \)), followed up by an all-reduce step (cost \( O(k \log p) \)). The result \( \gamma \in \mathbb{R}^{k_q} \) is stored on each processor and the multiplication \( \tilde{W}_{k_q} \gamma \) (cost \( O(N^2/p) \)) is fully local. The matrices \( \Psi \) and \( \Phi \) are stored in line blocks of size \( N/p \times K \). Thus, using arguments similar to those for the multiplication with \( V_{k_q}^\dagger \) and \( \tilde{W}_{k_q} \), and the costs for the multiplication of \( \Sigma \) with a vector of \( K^2 \), the overall parallel runtime for the Newton update thus accumulates to

\[ O(N/p \bar{K}) + O(K^2) + O(M N/p) + O(M k \log p). \]

**Efficient computation of \( \tilde{W} \).** Instead of recomputing \( \tilde{W}_k \) in every iteration, we update \( \tilde{W}_k := W_k - J_{\text{prev}}^{-1} V_k \) by adding the new column \( \Delta x_{k-1}^k - J_{\text{prev}}^{-1} \Delta r^k_{k-1} \), i.e.,

\[ (\tilde{W}_{k_n})_{i,0} = (W_{k_n})_{i,0} - \sum_{q=n-\eta}^n \tilde{W}_{k_q} \left( V_{k_q}^\dagger \cdot (V_{k_n})_{i,0} \right). \]

The new column is computed analogously to the second part of the computation of the Newton update and also yields identical parallel runtime complexity.

**Efficient computation of the SVD update.** Whenever a restart is triggered, the following steps need to be performed \( M \) times:

(a) Compute the orthogonal components of \( A \) and \( B \) with respect to \( \Psi \) and \( \Phi \), respectively (cf. (20)), and their QR-decomposition:

\[ \tilde{P} := (I - \Psi \Psi^T) A, \quad \text{qr}(\tilde{P}) = Q_A R_A, \]
\[ \tilde{Q} := (I - \Phi \Phi^T) B, \quad \text{qr}(\tilde{Q}) = Q_B R_B. \]

Calculating the orthogonal components of \( A \) with respect to \( \Psi \) is decomposed into computing the products \( \tilde{A} := \Psi^T A, \tilde{P} := \Psi \tilde{A}, \) and the actual QR-factorization \( \tilde{P} = Q_A R_A \). The matrices \( \Psi \) and \( \Phi \) are distributed analogously to \( A \) and \( B \) in row blocks. To compute \( \Psi^T A \), each processor locally computes its additive contribution, followed by an all-reduce operation (cf. Figure 2), which accumulates to a parallel complexity of \( O(\bar{K} N^2/p) + O(\bar{K} k \log p) \). The result is stored locally on each processor, as the rank \( \bar{K} \) of the SVD and the number of columns \( k \) is
assumingly small (cf. section 4). \( \hat{P} \) is computed fully local without any communication from \( \tilde{A} \) in time \( \mathcal{O}(N/p \tilde{K} k) \). Finally, the QR-factorization is computed as described earlier, yet there are no additional costs for the Givens rotations, since we always insert new columns at the end. Thus, the overall complexity for this step accumulates to a parallel runtime of

\[
\mathcal{O} \left( \tilde{K} N/p k \right) + \mathcal{O} \left( \tilde{K} k \log p \right).
\]

As the simulation proceeds, a lot of information is already covered by the truncated SVD representation of the Jacobian, and thus the number of new orthogonal modes (\( c_A \) and \( c_B \)) is expected to be very small. In particular, we have \( c_A, c_B \ll k \).

Due to the inexact arithmetic, \( \hat{P} \) is likely to have the same number of nonzero columns as \( \tilde{A} \). Columns \( p_i \in \text{img}(\Psi) \) fail to be orthogonalized within the QR-factorization and are deleted there. To this end, we delete the corresponding column from \( Q_A \) and the corresponding row from \( R_A \), as illustrated in Figure 3.

(b) Build the matrix \( S \in \mathbb{R}^{(\tilde{K}+c_A) \times (\tilde{K}+c_B)} \) according to (21). Having finished (a), the components \( A, B, R_A, \) and \( R_B \) are readily available on each processor and the matrix \( S \) (21) is built and stored locally in time \( \mathcal{O} \left( k(\tilde{K} + c)^2 \right) \) if \( c := \max(c_A, c_B) \) is the dimension of the column space of \( Q_A \) or \( Q_B \), respectively.

(c) Compute the SVD of \( S \): \( \text{svd}(S) = \Psi' \Phi' \). This step is purely sequential and takes \( \mathcal{O} \left( k(\tilde{K} + c)^3 \right) \) time.

(d) Rotate left and right eigenspaces:

\[
\left( \left[ \Psi \right] Q_A \right) \sum \left( \left[ \Phi \right] Q_B \right) \Phi^T.
\]

This step is embarrassingly parallel without the need of communication, since \( \Psi' \) and \( \Phi' \) are stored locally on each processor as can be seen from Figure 4. The parallel runtime complexity is \( \mathcal{O} \left( N/p(\tilde{K} + c)^2 \right) \).
(c) Truncate the SVD of $J^{-1}$: $\tilde{\Psi} \Sigma \tilde{\Phi}^T \leftarrow \tilde{\Psi} \tilde{\Sigma} \tilde{\Phi}^T$, i.e., find $q \in \{2, \ldots, \bar{K} + \min(c_A, c_B)\}$ such that $\sigma_q/\sigma_1 \leq \varepsilon < \sigma_{q-1}/\sigma_1$ and truncate $\tilde{\Psi} = [\tilde{\Psi}_1, \ldots, \tilde{\Psi}_q]$, $\tilde{\Phi} = [\tilde{\Phi}_1, \ldots, \tilde{\Phi}_q]$, and $\tilde{\Sigma} = \text{diag}(\sigma_1, \ldots, \sigma_q)$.

Summarizing, the parallel runtime complexity of updating the SVD of $J^{-1}$ by a rank-$k$ update $AB^T (= \tilde{W}_k V_k^\dagger)$ accumulates to

$$O \left( (\bar{K} + 2)N/p + k \right) + O \left( (\tilde{K} + c)^2 (N/p + k) \right) + O \left( (\tilde{K} + c)^3 \right) + O \left( (\tilde{K} + 1) k \log p \right)$$

$$\in O \left( \bar{K}^2 (N/p + k) \right) + O \left( \bar{K}^3 \right)$$

if $\bar{K} > k$,

which has to be performed $M$ times per restart. As restart is only performed at the end of a chunk of size $M$, the above costs can be seen as costs per time step and can certainly be upper bounded by costs of $O(\bar{K}^2 N/p) + O(\bar{K}^3/k)$ per iteration. Table 2 summarizes the parallel runtime complexities of the algorithmic building blocks for the LS and the MVJ restart methods for one quasi-Newton iteration.

4. Numerical results. In this section, we present numerical result in the context of fluid-structure interactions that aim to show various observations we made: Section 4.1 evaluates the efficiency of previous time step reuse in the LS($\xi$) versus the MVJ method. This evaluation shows both the robustness of the MVJ method and the sensitivity of the LS($\xi$) method on the correct choice of $\xi$. Both too small and too large values slow down the convergence. We do not further consider Broyden, as it has proven to be less efficient than MVJ in [20, 26]. In section 4.2, we show results that compare the different restart alternatives for the MVJ method. The restart is the central component that allows us to reduce the MVJ costs to linear complexity. We observe that the RS-SVD restart is robust and efficient in all examples. Section 4.3 addresses the main goal of the numerical results: We compare the restarted (linear complexity) MVJ with the filtered LS($\xi$) with very large $\xi$ and evaluate the suitability of these two “parameter-free” and linear complexity methods for our fluid-structure interaction scenarios. Section 4.4 finally shows scaling and runtime results for the efficient parallel implementation within the coupling library preCICE for both methods.

We use three fluid-structure scenarios in this section: The three-dimensional flexible tube and the fluid-structure benchmark (FSI3) from [28] and a simplified one-dimensional flexible tube scenario that allows for extensive testing of different coupling variants. In the following, we briefly sketch the respective scenario descriptions.

The two three-dimensional test cases are displayed in Figure 5.
Table 2
Overview of the parallel runtime complexities of the algorithmic building blocks of one iteration of the LS and the restart MVJ quasi-Newton approaches. For the LS method the number of columns $k$ within the least-squares system is typically much higher than for the MVJ method due to a high number $\xi$ of reused time steps. For most cases even $k > \bar{K}$ holds true. The effect for the runtime complexity can be seen from the scaling results in section 4.4.

<table>
<thead>
<tr>
<th>Block</th>
<th>LS</th>
<th>MVJ</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A.1) QR-decomposition update</td>
<td>$O(N/p k) + O(k^3) + O(k \log p)$</td>
<td></td>
</tr>
<tr>
<td>(A.2) QR-decomposition recomputation</td>
<td>$O(N/p k^2) + O(k^4) + O(k^2 \log p)$</td>
<td></td>
</tr>
<tr>
<td>(B.1) $\alpha = (V_k^T V_k)^{-1} V_k^T r_i^k$</td>
<td>$O(N/p) + O(k \log p) + O(k^2)$</td>
<td></td>
</tr>
<tr>
<td>(B.2) $Z_k = (V_k^T V_k)^{-1} V_k^T$</td>
<td></td>
<td>$O(N/p k^2)$</td>
</tr>
<tr>
<td>(C.1) $\Delta x^{k+1} = W_k \alpha$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(C.2) $\Delta x^{k+1} = J^{k-1} r^k - \sum_{q=n-\bar{K}}^n \tilde{W}_{kq} \left(V_k^T r^k\right)$</td>
<td></td>
<td>$O(N/p) + O(\bar{K}^2) + O(M N/p k) + O(M k \log p)$</td>
</tr>
<tr>
<td>(D) $(\tilde{W}<em>{k_n})</em>{i,0} = (W_{k_n})<em>{i,0} - \sum</em>{q=0}^M \tilde{W}<em>{kq} \left(V_k^T \cdot V</em>{k_n}\right)_{i,0}$</td>
<td></td>
<td>$O(M N/p k) + O(M k \log p)$</td>
</tr>
<tr>
<td>(E) $\Psi \sum \Psi^T + \sum_{m=1}^M \tilde{W}_{km} V_k^T$ (every $M$ time steps)</td>
<td></td>
<td>$O(M N/p + k) + O(M \bar{K}^3)$</td>
</tr>
</tbody>
</table>

Total/Iteration | $O(Nk^2/p) + O(k^3) + O(k^2 \log p)$ | $O(Nk^2/p) + O(k^4) + O(k^2 \log p)$ + $O(M \bar{K}^2) + O(M k \log p) + O(\bar{K}^3)$ |
in size, and the incompressible flow is driven by a parabolic velocity profile with mean inflow velocity $\bar{v} = 0.2 \text{ m s}^{-1}$ at the left boundary and free outflow at the right boundary. The flexible flap is modeled using a Saint-Venant–Kirchhoff material model. The physical parameters for both flow and solid material are given in Table 3. We use a rather coarse mesh with 23,924 cells for the flow solver and 328 cells for the solid part, resulting in 672 unknowns at the interface for the J-system. After an initialization phase of 10 s, we simulate a time frame of $0.318 \text{ s}$ with a time step size $dt = 1 \times 10^{-3} \text{ s}$, which corresponds to 10 oscillation periods of the attached cantilever. Both solvers use implicit Euler time steps. Quantitative simulation results, which are pretty much aligned with the reference values from [28], are presented in [7] for the given configuration.

The flexible tube example [2, 12, 15] simulates a wave propagating in a three-dimensional elastic tube induced by an initial pressure pulse of peak $1.333 \times 10^{3} \text{ Pa}$ with a duration of $1 \times 10^{-3} \text{ s}$. The tube is 0.05 m long, has a wall thickness of 0.001 m, and has an inner diameter of 0.01 m. The fluid mesh consists of 16,000 cells, and the solid mesh contains 6,400 cells, resulting in 9,600 unknowns at the interface for the J-system. The tube wall is considered an external elastic structure with neglected inertia. For physical parameters of flow and solid, see Table 3. A hundred time steps with $dt = 1 \times 10^{-4} \text{ s}$ are simulated.

### Table 3

<table>
<thead>
<tr>
<th>Fluid</th>
<th>Solid</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rho_f = 1 \times 10^{3} \text{ kg m}^{-3}$</td>
<td>$\rho_s = 1 \times 10^{3} \text{ kg m}^{-3}$</td>
</tr>
<tr>
<td>$\nu_f = 1.0 \times 10^{-3} \text{ Pa s}$</td>
<td>$E = 1.4 \times 10^{6} \text{ N m}^{-2}$</td>
</tr>
<tr>
<td>$Re = 200$</td>
<td>$\nu_s = 0.3$</td>
</tr>
</tbody>
</table>

To check for convergence of the implicit coupling scheme, we use a criterion based on the relative residuals for both pressure $x_f$ and displacements $x_d$, i.e.,

$$\frac{\|x_d^{k} - \tilde{x}_d^{k}\|_2}{\|x_d^{k}\|_2} < \varepsilon_{\text{conv}} \quad \text{and} \quad \frac{\|x_f^{k} - \tilde{x}_f^{k}\|_2}{\|x_f^{k}\|_2} < \varepsilon_{\text{conv}},$$

where $\tilde{x}_d^{k} = S_2(x_d^{k})$ and $\tilde{x}_f^{k} = S_1(x_d^{k})$ and $\varepsilon_{\text{conv}} \in \{10^{-04}, 10^{-05}\}$.

FSI simulations use two separate, possibly black-box solvers for the fluid domain and the structure domain, respectively. For our experiments, we use the incom-
pressible flow solver and the elastic structure solver from the open-source simulation toolbox OpenFOAM.\(^5\) The flow solver is based on a second order finite volume discretization of the incompressible Navier–Stokes equations and uses a coupled solution algorithm as described in [10], instead of the standard PISO (pressure implicit with splitting of operator) algorithm. Mesh movement is done via radial-basis function interpolation \(^6\). The solvers\(^6\) themselves were developed by Blom, van Zuijlen, and Bijl based on the foam-extend-3.1 project.\(^7\) For more information, see [3].

The coupling conditions are realized by a Dirichlet–Neumann approach: The fluid solver \(S_1\) takes displacements and velocities \((x_d)\) at the wet surface and computes forces \((x_f)\) exerted on the structure, whereas the structure solver \(S_2\) takes these forces and computes new displacements and velocities at the wet surface; cf. \((1)\).

All three-dimensional experiments were conducted using preCICE\(^8\) [16], a library for flexible numerical coupling of single-physics solvers for data transfer, data mapping, and the iterative solver.

For the one-dimensional flexible tube, we give a very brief scenario description, omitting formulas and details. Refer to the literature for more details [13, 16]. The fluid flow in the one-dimensional flexible tube is assumed to be incompressible and inviscid. The one-dimensional model is obtained by averaging over the tube in a radial direction, as shown in Figure 6. The conservation of mass and momentum simplifies to

\[
\partial_t (a u) + \partial_x (a u^2) + a \partial_x p = 0 \quad \text{and} \quad \partial_t + \partial_x (a u) = 0,
\]

with the inflow velocity \(u\), the kinematic pressure \(p\), and the cross-sectional area of the tube \(a\). We use a time varying, sine-shaped inlet velocity and nonreflecting outlet boundary conditions. The elastic wall is modeled by a Hookean constitutive law neglecting inertia. This enhances instabilities. The fluid exerts stresses on the structure only in a radial direction, leading to a purely radial motion of the tube wall. We simulate the scenario over one full period of the inlet velocity \([0; T]\) with 100 time steps. Following [13], we use the dimensionless structural stiffness \(\kappa\), the dimensionless time step size \(\tau\), and the spatial resolution \(N\) as parameters characterizing the difficulty: The coupling becomes more challenging for decreasing structural stiffness \(\kappa\), decreasing time step size \(\tau\), and increasing spatial resolution \(N\). We use a MATLAB implementation for the fluid and solid solver as well as for the coupling algorithms themselves to foster fast prototyping of different approaches.

### 4.1. Convergence properties of LS(\(\xi\)) versus MVJ.

To evaluate the efficiency of previous time step reuse, we apply the LS(\(\xi\)), as well as the implicit reuse in

\(^5\)http://www.openfoam.org/
\(^6\)https://github.com/davidsblom/FOAM-FSI
\(^7\)http://www.extend-project.de
\(^8\)http://www.precice.org
Table 4

Averaged number of coupling iterations for the LS and MVJ methods for the flexible tube and the FSI3 test case for the J-system and varying numbers of reused time steps $\xi$ for the LS approach. A relative convergence criterion of $\varepsilon_{\text{conv}} = 10^{-05}$ was used.

<table>
<thead>
<tr>
<th>Reuse $\xi$</th>
<th>Three-dimensional flexible tube</th>
<th>FSI3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 4 8 12 16</td>
<td>0 2 4 6 8 16 32</td>
</tr>
<tr>
<td>(J)-LS</td>
<td>28.9 14.6 13.4 13.2 13.3</td>
<td>20.0 8.0 6.2 5.6 5.3 6.2 11.9</td>
</tr>
<tr>
<td>(J)-MVJ</td>
<td>11.6</td>
<td>6.2</td>
</tr>
</tbody>
</table>

Fig. 7. Comparison of the mean number of coupling iterations of LS (with 8 reused time steps) and MVJ for the one-dimensional flexible tube scenario for various values of the structure stiffness $\kappa$ and the dimensionless time step size $\tau$.

the MVJ, for the FSI3 and the three-dimensional tube. Table 4 shows the resulting numbers of iterations for both test cases and both types of quasi-Newton fixed-point solvers (LS($\xi$) and MVJ). They show that both solvers yield comparable convergence properties, but the MVJ solver does not require the costly optimization of the highly scenario-dependent number of reused time steps $\xi$. For the flexible tube test case, the MVJ method is superior even if we compare it with the LS method with the optimal number of reused time steps.

4.2. Comparison of multivector restart variants. The proposed MVJ restart variants influence the accuracy of the inverse Jacobian approximation. We study the effects of this influence for the one-dimensional flexible tube scenario [13], which allows for a detailed analysis and fast prototyping using MATLAB due to its simplicity while still offering characteristic challenges in coupled FSI simulation. Figure 7 shows that MVJ without restart is competitive with LS with an optimal number of reused steps $\xi$ for this scenario. Recall that we already stated in section 2.2 that the performance of the LS method suffers from its strong dependency on the number of reused time steps $\xi$: For the one-dimensional flexible tube, we observe that we obtained unacceptably slow convergence for $\xi = 0$. For $\xi = 8$, the difficult cases (small $\tau$ and $\kappa$) show excellent results, but the method diverges for the easier cases.

Keeping these observations in mind, we restrict ourselves in the following to comparing the MVJ restart variants with each other and with the original MVJ. Figure 8 shows results for different chunk sizes $M$ of the MVJ estimation era and four parameter settings for ($\tau, \kappa$). The clear-all restart method RS-0 yields the worst performance, as expected, unless $M$ is large enough. As RS-0 yields a mixture of the LS($\xi = 0$) and MVJ methods, it inherits the bad LS($\xi = 0$) convergence. In contrast, explicitly reusing multisecant information at restart with the RS-LS method shows good results
Fig. 8. One-dimensional flexible tube scenario: averaged number of coupling iterations for the restart methods from section 3.2, for different sizes of the chunk size $M$ between MVJ restarts and various parameter combinations $(\tau, \kappa)$.

Fig. 9. One-dimensional flexible tube scenario: averaged number of coupling iteration for the RS-LS restart method for different numbers of reused time steps at restart $\xi_{RS}$, chunk sizes $M$, and parameter combinations $(\tau, \kappa)$. For small chunk sizes $M = 1, 2$ for all cases except the hardest setting ($\tau = 0.001$, $\kappa = 10$). Here, the amount of reused information across MVJ-chunk borders is not sufficient for small $M$. Nonetheless, the good results for RS-LS($\xi_{RS} = 2, M = 2$) raise hopes for a cheap but efficient fixed-point acceleration method for scenarios with moderate instabilities. However, the mixture with the LS($\xi$) method reintroduces the dependency on the parameter $\xi = \xi_{RS}$ (only in a less sensitive way), as shown in Figure 9, where we vary the number of reused time steps at restart ($\xi_{RS}$) for different chunk sizes. Once again, $M = 2$ shows superior performance for arbitrary numbers of reused time steps at restart (except for the last case), but the optimum is obtained for $\xi_{RS} = 2$ (cases 1 and 2) and $\xi_{RS} = 8$ (case 3), respectively, which is also the optimal parameter $\xi$ for the LS method. The optimum of RS-LS($M = 2, \xi_{RS}$) yields slightly faster convergence than the MVJ method, as it benefits from both reuse approaches.

With these preliminary observations, we turn to the more realistic three-dimensional flexible tube from section 2.2. For the RS-LS variant, we used $\xi_{RS} = M$ and collected the first five iterations from each of the last $\xi_{RS}$ time steps for explicit reuse at restart. The iteration numbers in Figure 10 show that both RS-0 and RS-LS achieve only slow convergence. While this was expected for RS-0, the results for RS-LS are, to some extent, contradicting the results for the one-dimensional tube scenario. The green dashed line in Figure 10 indicates the total number of degrees of freedom at the coupling surface, i.e., the maximum possible number of columns in the RS-LS method and the maximum possible rank of $J^{-1}$ and, therewith, also of the truncated SVD. We conclude that the RS-LS method may, in some cases, be a cheap and effective acceleration method, though its convergence properties are highly problem dependent such that it can be as slow as RS-0. The RS-SVD approach shows excellent results that achieve the original MVJ convergence quality with a fraction of the original costs due to the small rank of the truncated SVD, indicated by the red dashed lines. In the following, we therefore restrict ourselves to the RS-SVD method, since it shows...
the most robust and reliable results, even in scenarios tending to become severely unstable.

For the efficiency of the RS-SVD method, it is crucial that the actual rank $J^{-1}$ and, as a consequence, the rank of the truncated SVD be small compared to the number of interface elements $N$ and be independent from $N$. For the one-dimensional tube example, Figure 11 shows the rank of $J^{-1}$ as obtained for the original MVJ method (green markers), the size of the truncated SVD of the Jacobian (red lines), depending on the cut-off parameter $\epsilon$ for different numbers of interface unknowns $N$ (blue line on top). The associated iteration numbers are given as blue bars. Depending on the difficulty of the test case, the rank grows, but it keeps (nearly) constant for increasing $N$ after $N$ has become sufficiently large to capture all physically relevant interface modes. For the three-dimensional tube, a similar behavior is observed, as can be seen from Figure 12: Five different meshes with double the number of unknowns at the interface from one mesh to the next are considered. The average size of the SVD, depicted by a red dashed line, keeps constant as the dimensionality of the coupling problem increases. At the same time, the iteration numbers for the restart method increase moderately. The reason for this increase is probably to be
seen in the following trick we used to keep the larger problems computable for the OpenFOAM fluid solver: Due to the high dimensionality of the three-dimensional flow problem, a coarsening strategy for the mesh-movement algorithm needed to be applied, which selects a subset of the mesh points for the movement computation. This results in less accuracy and, hence, in worse iteration numbers for the FSI coupling. The effect increases for higher levels as we stick to a fixed number of selected points. However, the iteration numbers almost reach the iteration count from the original MVJ method for a suitable choice of the truncation threshold. As the MVJ method needs to store the $N \times N$ Jacobian matrix explicitly and also results in quadratic runtime complexity, higher levels are not feasible for the MVJ method.

Regarding the parameters for the $\text{RS-SVD}(M, \varepsilon)$ method, we see a slight dependency on the chunk size $M$ that vanishes for a proper choice of the truncation parameter $\varepsilon$. In theory, $M$ controls the emphasis on the restart technique compared to the MVJ properties for implicit reuse (cf. the RS-0 curves for different $M$). Small choices for $M$ stress the restart and require a sophisticated transmission of already captured information from the past across restart borders. To get a third FSI scenario to support this idea, we modified the three-dimensional flexible tube scenario from section 2.2 in such a way that instead of one pressure pulse, three pressure pulses propagate through the tube to foster displacements of the tube wall that are harder to capture with the quasi-Newton approaches (cf. Figure 13). We simulate a time frame of $3 \times 10^{-2}$ s with pressure pulses of peak 1,333.2 Pa at $t = 0$ s, $t = 6 \times 10^{-3}$ s, and $t = 1.0 \times 10^{-2}$ s with a duration of $3.0 \times 10^{-3}$ s each.

We observe that for the one-dimensional tube (Figure 8), the three-dimensional tube with multiple pressure pulses, and the FSI3 benchmark scenario (Figure 14), $M$ can actually be fixed to a small value, e.g., $M \in \{4, 8\}$. The truncation threshold for SVD controls the accuracy of the information transmission but also affects the runtime complexity of the method. For all the experiments (Figures 10, 11, 12, and 14), we see a clearly visible jump in the number of coupling iterations when going from $\varepsilon = 10^{-2}$ to $\varepsilon = 10^{-1}$ or from $\varepsilon = 10^{-1}$ to $\varepsilon = 0.5$, respectively. Hence, we assume that most of the information incorporated into the Jacobian lies within a radius of

![Fig. 12. Three-dimensional flexible tube: averaged size of the truncated SVD (red lines) of the RS-SVD method and mean number of coupling iterations for different values of the truncation parameter $\varepsilon$ and growing numbers of unknowns at the interface $N$. We used a QR2 filter with $\varepsilon_F = 10^{-02}$ and a relative convergence criterion with $\varepsilon_{\text{conv}} = 10^{-04}$. (Color available online.)](image-url)
one or two orders of magnitude around the largest singular value and can easily be determined.

From this, we conclude, that the RS-SVD method in fact has a complexity that is linear in the problem size $N$ with a small constant factor and is expected to be well suited and efficient for coupled FSI simulation without any tuning of parameters.

4.3. Comparison of filtered LS reuse and MVJ SVD restart. We have presented two different approaches—filtered LS with a basically unbounded number of reused time steps $\xi$ and MVJ with RS-SVD restart—in order to provide an efficient, fast, and robust quasi-Newton acceleration method with minimal requirements for the tuning of additional parameters. We refer to them as $\text{LS}(\infty)$-QR1 or $\text{LS}(\infty)$-QR2 (depending on the used filtering) and MVJ-RS-SVD. In the remainder of this section, we compare both approaches with respect to convergence properties and robustness (4.3) but also regarding efficiency and parallel scalability (4.4). For these comparisons, we restrict ourselves to the two variants of the three-dimensional flexible tube scenario and the FSI3 benchmark. For all experiments we use a relative convergence criterion with $\varepsilon_{\text{conv}} = 10^{-04}$.

Table 5 summarizes the results for the three-dimensional flexible tube for both methods. In particular, we study the effects of the two filtering variants presented in section 3.1. For all configurations, the QR1 filter shows poor results and fails to accelerate the convergence speed of the methods and to improve the condition of the least-squares problem. The QR2 filter, on the other hand, shows fairly good results and manages to stabilize the $\text{LS}(\xi)$ method for large choices of $\xi$. $\text{LS}(\xi)$ with $\xi = 256$ can be considered to be equivalent to $\text{LS}(\infty)$. Note that the QR2 filtering results in a higher runtime complexity than that of QR1 as it triggers a recomputation of the QR-
Three-dimensional flexible tube: averaged number of iterations for the filtered LS and the MVJ method with and without restart for the three-dimensional tube scenario. The two filtering variants from section 3.1 are investigated for different numbers of reused time steps $\xi$ and filtering thresholds $\varepsilon_F$. Italic numbers indicate that no columns have been deleted by the filtering. Slight differences in the iteration numbers for these cases between QR$^1$ and QR$^2$ are due to the different QR-decomposition algorithms used, inducing different rounding errors. The MVJ-RS-SVD uses the chunk size $M = 8$ and SVD truncation thresholds $10^{-2}$ (RS-SVD-2) and $10^{-1}$ (RS-SVD-1). Bold numbers indicate the best results for each method type.

<table>
<thead>
<tr>
<th>$\varepsilon_F$</th>
<th>QR1 filter</th>
<th></th>
<th></th>
<th></th>
<th>QR2 filter</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>$\xi = 8$</td>
<td>10.26</td>
<td>10.52</td>
<td>10.52</td>
<td>11.94</td>
<td>13.95</td>
<td>10.13</td>
<td>9.74</td>
<td>9.21</td>
</tr>
<tr>
<td>$\xi = 16$</td>
<td>9.53</td>
<td>9.82</td>
<td>9.82</td>
<td>11.65</td>
<td>15.00</td>
<td>9.25</td>
<td>8.27</td>
<td>7.59</td>
</tr>
<tr>
<td>$\xi = 256$</td>
<td>15.04</td>
<td>15.97</td>
<td>15.97</td>
<td>33.21</td>
<td>21.42</td>
<td>12.16</td>
<td>7.55</td>
<td>6.18</td>
</tr>
<tr>
<td>MVJ</td>
<td>6.00</td>
<td>6.00</td>
<td>6.00</td>
<td>6.32</td>
<td>6.76</td>
<td>5.99</td>
<td>5.99</td>
<td>5.99</td>
</tr>
<tr>
<td>RS-SVD-2</td>
<td>6.07</td>
<td>6.07</td>
<td>6.07</td>
<td>6.36</td>
<td>6.77</td>
<td>6.03</td>
<td>6.03</td>
<td>6.03</td>
</tr>
<tr>
<td>RS-SVD-1</td>
<td>6.28</td>
<td>6.28</td>
<td>6.28</td>
<td>6.42</td>
<td>7.05</td>
<td>6.20</td>
<td>6.20</td>
<td>6.20</td>
</tr>
</tbody>
</table>

factorization in every iteration. For the MVJ methods, the convergence properties are best if no filter is applied, as assumed due to the fact that the least-squares system is very small, as it is not enriched by previous time step information. Thus, we do not have to tune a filter for MVJ. The convergence speed for the LS(256)-QR2 method with $\varepsilon_F = 10^{-2}$ and the MVJ method is comparable. In particular, there is almost no loss in convergence speed when switching from the MVJ method to its efficient restart counterpart MVJ-RS-SVD.

For the three-dimensional flexible tube with multiple pressure pulses and the FS3 cylinder flap benchmark scenario, results achieved with the LS($\infty$)-QR1, the LS($\infty$)-QR2, the MVJ, and the the MVJ-RS-SVD are given in Table 6. The results confirm the observations made before: The QR2 filter is powerful enough to stabilize the LS($\infty$) method with, in this case, $\xi = 256$, i.e., practically unrestricted reuse of old time step information. The MVJ methods are slightly better than the LS($\infty$) methods regarding convergence speed. The restart of the MVJ method does not deteriorate the convergence.

4.4. Parallel scalability. In this section, we present a scalability study for LS($\infty$)-QR2 and MVJ-RS-SVD, based on the three-dimensional flexible tube scenario. Due to the high computational cost of the flow problem, the flexible tube scenario cannot be scaled arbitrarily high. Thus, we additionally consider an artificial test case, which allows us to evaluate the performance and scalability of the interface numerics and which is isolated from solver runs. Please note that for both test cases the number of processors only refers to the number of processors involved at the coupling interface. The number of processors involved for the complete multiphysics simulation is much higher.

We perform a weak-scaling test series for the LS(30) and MVJ-RS-SVD methods. Note that a reuse of $\xi = 30$ time steps is probably less than what would be required to render the tuning of parameters unnecessary. The MVJ-RS-SVD method restarts after every $M = 8$ time steps and uses a truncation threshold $\varepsilon = 10^{-2}$. Both methods are applied with a QR2 filtering with $\varepsilon_F = 10^{-2}$. For each runtime measurement, we performed a total of five measurements. Their standard deviation is indicated with error bars in the figures. All experiments were conducted on the Haswell nodes.
Three-dimensional flexible tube with multiple pressure pulses and FSI3: averaged number of iterations for the LS(∞) methods (with ξ = 256 mimicking ξ = ∞) compared to the MVJ methods. For the MVJ-RS-SVD variants, a chunk size of M = 8 and SVD truncation thresholds 10⁻² (RS-SVD-2) and 10⁻¹ (RS-SVD-1) were used. We study the effect of the filters from section 3.1. Italic numbers indicate that no columns are deleted by the filter. Bold numbers indicate the best results for each method type.

<table>
<thead>
<tr>
<th>Method</th>
<th>10⁻⁸</th>
<th>10⁻⁷</th>
<th>10⁻⁶</th>
<th>10⁻⁵</th>
<th>10⁻⁴</th>
<th>10⁻³</th>
<th>10⁻²</th>
<th>10⁻¹</th>
</tr>
</thead>
<tbody>
<tr>
<td>LS(256)</td>
<td>14.05</td>
<td>14.39</td>
<td>19.45</td>
<td>41.61</td>
<td>37.07</td>
<td>9.78</td>
<td>7.11</td>
<td>5.74</td>
</tr>
<tr>
<td>MVJ</td>
<td>4.89</td>
<td>4.89</td>
<td>4.89</td>
<td>5.03</td>
<td>5.23</td>
<td>4.91</td>
<td>4.91</td>
<td>5.48</td>
</tr>
<tr>
<td>RS-SVD-2</td>
<td>4.93</td>
<td>4.93</td>
<td>4.93</td>
<td>5.08</td>
<td>5.30</td>
<td>4.93</td>
<td>4.93</td>
<td>5.55</td>
</tr>
<tr>
<td>RS-SVD-1</td>
<td>5.11</td>
<td>5.11</td>
<td>5.11</td>
<td>5.29</td>
<td>5.57</td>
<td>5.17</td>
<td>5.17</td>
<td>5.69</td>
</tr>
</tbody>
</table>

FSI3 cylinder flap benchmark scenario

<table>
<thead>
<tr>
<th>Method</th>
<th>10⁻⁸</th>
<th>10⁻⁷</th>
<th>10⁻⁶</th>
<th>10⁻⁵</th>
<th>10⁻⁴</th>
<th>10⁻³</th>
<th>10⁻²</th>
<th>10⁻¹</th>
</tr>
</thead>
<tbody>
<tr>
<td>LS(256)</td>
<td>5.15</td>
<td>5.12</td>
<td>5.18</td>
<td>9.13</td>
<td>33.79</td>
<td>5.13</td>
<td>4.67</td>
<td>4.35</td>
</tr>
<tr>
<td>MVJ</td>
<td>3.27</td>
<td>3.27</td>
<td>3.27</td>
<td>3.41</td>
<td>3.33</td>
<td>3.28</td>
<td>3.28</td>
<td>3.43</td>
</tr>
<tr>
<td>RS-SVD-2</td>
<td>3.57</td>
<td>3.57</td>
<td>3.57</td>
<td>3.87</td>
<td>3.88</td>
<td>3.69</td>
<td>3.69</td>
<td>3.69</td>
</tr>
<tr>
<td>RS-SVD-1</td>
<td>4.64</td>
<td>4.64</td>
<td>4.64</td>
<td>4.79</td>
<td>4.81</td>
<td>4.49</td>
<td>4.49</td>
<td>4.49</td>
</tr>
</tbody>
</table>

Relation between algorithmic building blocks studied in our weak scalability experiments (Figures 15 and 17). The names are mapped to the corresponding formulas and parallel runtime complexities in Table 2.

<table>
<thead>
<tr>
<th>Per iteration</th>
<th>per time step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quasi-Newton iteration</td>
<td>Update matrices</td>
</tr>
<tr>
<td>(A.2)</td>
<td>(A.1), (D)</td>
</tr>
<tr>
<td>– apply filter</td>
<td>– compute update</td>
</tr>
<tr>
<td>(B)</td>
<td>(C)</td>
</tr>
<tr>
<td>– QR solve</td>
<td>– VB solve</td>
</tr>
<tr>
<td>(E)</td>
<td>(F)</td>
</tr>
</tbody>
</table>

partition of SuperMUC, Leibniz Supercomputing Center in Garching (Haswell architecture with Xeon E5-2697 v3 processors). Each node comprises 28 cores, and nodes are interconnected via an Infiniband FDR14. The main algorithmic building blocks that we examine in this study are summarized in Table 7. The table indicates the relation and hierarchy between the algorithmic blocks and links to the formulas and runtime complexities given in Table 2. Note that all steps summarized in the “time step converged” column are executed only once per time step, whereas all other steps have to be done in every iteration.

For the real three-dimensional flexible tube test case, we measure the parallel runtime over six levels, doubling the number of unknowns at the interface as well as the number of processors at the interface at each level. The initial workload on the lowest level, L₀, corresponds to 9,600 unknowns at the interface, distributed over four processors. From the results presented in Figure 15, we see that the most expensive part for the LS(30)-QR2 method is the apply filter block that triggers a recomputation of the QR-decomposition. This process dominates the cost per quasi-Newton iteration, but also shows a rather poor parallel scalability. This is due to the O(k²log p) factor in the recomputation of the QR-decomposition and the typically large value for k for large numbers of reused time steps ξ. For the MVJ-RS-SVD method, on the other hand, the apply filter block is very cheap, as we are given a small least-squares system (small k) such that filtering is hardly ever actually required.
However, we see the nonoptimal parallel scalability for all building blocks that include inserting columns in the QR-decomposition or recomputation of the latter. The most expensive part for MVJ-RS-SVD is the restart itself, which comprises several QR-decompositions and the sequential computation of a small SVD. However, the restart is only performed after each \( M \) time steps and is completely included in the time measurements for the timestep converged block, which in this case is even lower than for LS(30)-QR2. Furthermore, the update of the SVD shows quite good parallel scalability. Summarizing the measurements for this test case, MVJ-RS-SVD is slightly more efficient than the LS(30)-QR2 method and in particular shows better parallel scalability.

To get an impression on how the above runtime results compare to the plain vanilla MVJ method with the major matrix operations outlined in Table 1, we consider runtime measurements for the three-dimensional flexible tube scenario with \( N = 9,600 \) unknowns distributed over \( p = 4 \) processors; cf. Table 8. The work per iteration is reduced remarkably due to the improvements within the MVJ-RS-SVD method. Moreover, we could produce any factor between the runtimes for the plain vanilla MVJ and its restart counterpart if we increase the number of unknowns at the interface \( N \), as the former suffers from a quadratic complexity with respect to \( N \), while the restart counterpart shows linear complexity in \( N \). Compare also parallel runtime results for LS(0) (without filtering) and MVJ for \( N = 2^5, \ldots, 2^{14} \) unknowns in [8, Fig. 16] and a strong scaling for \( N = 2^{14}, p = 2^4, \ldots, 2^7 \) [8, Fig. 14] for an artificial test case. Note that the per-time-step cost for MVJ-RS-SVD does not grow remarkably, as the rank of the SVD is small and independent from \( N \).

When scaling beyond \( N = 153,600 \) unknowns at the interface on \( p = 128 \) cores, the flow solver of openFOAM (running on \( p = 1024 \) cores) becomes a major bottleneck, and the internal memory consumption exceeds the resources. Thus, we consider a dummy solver that generates artificial (random) values at the coupling interface at...
minimal computational load, to isolate the coupling functionality from solver effects. Figure 16 shows a schematic drawing of both dummy solvers that are coupled via preCICE. The grid vertices are numbered in rowwise order and partitioned among a given number of processors by splitting the linear representation of vertices induced by their numbering in partitions of similar size. We solely consider the coupling surface itself and thus only generate nodes in a two-dimensional plane. The meshes for both dummy solvers are identical.

In order to model a realistic artificial solver environment, we adopt the parameter settings and characteristics from the three-dimensional tube scenario. In other words, the artificial solvers help to emulate the behavior of the quasi-Newton methods for the three-dimensional tube scenario. For the three-dimensional flexible tube experiments we got an average number of \( k = 230 \) used columns in the least-squares system for the LS(30)-QR2 method and an average rank of \( \bar{K} = 210 \) of the truncated SVD for MVJ-RS-SVD. Further, LS(30)-QR2 converged after an average number of eight coupling iterations per time step, while the MVJ-RS-SVD method converged within seven iterations on average. We fixed the maximum size of the least-squares system and the rank and the number of performed iterations per time step to these values. Apart from this, the configuration remains unchanged compared to the flexible tube scenario. This allows for a weak-scaling series over nine levels, ranging from \( N = 2^{13} \) unknowns distributed over \( p = 4 \) processors to \( N = 2^{22} \) unknowns at the interface on \( p = 2^{11} \) processors. The results are given in Figure 17. The measurements for the first six levels are in perfect agreement with the results from the real scenario, which justifies the choice of fixed parameters. Due to the fact that there is no fluctuation for the rank of the truncated SVD, we get a perfect parallel scalability for the SVD update and restart functionality of MVJ-RS-SVD. The LS(30)-QR2 method, on the other hand, exhibits a bad scalability for all operations that involve \( O(\log p) \) complexity terms multiplied by a polynomial in the number of used columns \( k \) in the least-squares system due to the extensive reuse. In particular, filtering, i.e., the recomputation of the QR-decomposition, dominates the runtime. This leads to a per-iteration execution time of \( \approx 2.6 \) s for LS(30)-QR2, compared with a per-iteration runtime of \( \approx 0.018 \) s and a runtime of \( \approx 0.56 \) s per converged time step for the MVJ-RS-SVD method on the highest level. Summarizing, the performance and parallel scalability study shows that the newly introduced MVJ-RS-SVD method clearly outperforms the LS method if the reuse parameter \( \xi \) is fixed to a large value and good conditioning and stability is ensured via a special filtering technique. Parallel scalability as well as total execution time for the MVJ-RS-SVD method are considerably better than for the LS(\( \infty \))-QR2 method. Clearly, the latter suffers from the recomputation of the QR-decomposition in every iteration, which appears to be extremely costly due to the high number of columns in the least-squares system.
Fig. 17. Artificial text case: weak-scaling series for the LS(30)-QR2 and the MVJ-RS-SVD methods. To isolate coupling functionality from solver effects, artificial solvers providing random data at the interface are applied. Parameters and properties of the methods are fixed to values that represent the properties of the flexible tube scenario to ensure a realistic workload. Nine levels are considered from $N = 2^{13}$ unknowns on $p = 4$ processors up to $N = 2^{22}$ unknowns distributed over $p = 2^{11}$ processors. (Color available online.)

5. Conclusion. We have analyzed two different quasi-Newton methods suited for partitioned multiphysics simulations using black-box solvers. The numerical results and the complexity analysis of the underlying algorithms showed that the novel combination of the MVJ method—which can be interpreted as a generalized Broyden method—yields a very powerful and robust iterative solver for nonlinear interface equations. This required a careful algorithm design based on efficient decompositions of the inverse Jacobian approximations and their truncation. In particular, it comes without a strong sensitivity on solver parameters and has an optimal linear complexity. We compared this method to an alternative and algorithmically simpler approach that combines standard least-squares quasi-Newton with reuse of information from a large number of old time steps and a filtering method automatically extracting the useful information from the large data set collected over all of these time steps. This approach showed similar convergence properties as the MVJ variant in terms of required coupling iterations, but longer runtime and worse parallel scalability. Although the methods were tested for fluid-structure interaction scenarios only, we expect them to be powerful tools also in many other coupled applications having to deal with black-box software components. One might argue that the costs for coupling numerics on a lower-dimensional surface are always negligible and not worth the effort we have invested in our work. However, two aspects render this statement wrong: (1) The number of iterations is actually very crucial, and also trial-and-error for parameter optimization has to be avoided as each extra iteration corresponds to solving a full implicit step in the involved single-physics domains. (2) Uekermann [29] showed that for massively parallel simulations, i.e., more than a few thousand cores, the computational time spent in the coupling component is no longer negligible relative to the solver runtime. Thus, the linear complexity in our approach is mandatory to allow for really large-scale simulations.

Acknowledgments. The methods presented in this paper have been implemented in the coupling library preCICE (www.precice.org). We acknowledge the work of all preCICE developers. Being able to use preCICE enabled the easy and fast setup of all fluid-structure scenarios shown in the numerical results on high-performance computers.
REFERENCES


