Archives of Computational Methods in Engineering
https://doi.org/10.1007/s11831-023-09907-y

REVIEW ARTICLE

Quasi-Newton Methods for Partitioned Simulation of Fluid–Structure Interaction Reviewed in the Generalized Broyden Framework

Nicolas Delaissé[1] · Toon Demeester[1] · Rob Haelterman[3] · Joris Degroote[1,2]

Received: 17 November 2022 / Accepted: 26 February 2023

Abstract
Fluid–structure interaction simulations can be performed in a partitioned way, by coupling a flow solver with a structural solver. However, Gauss–Seidel iterations between these solvers without additional stabilization efforts will converge slowly or not at all under common conditions such as an incompressible fluid and a high added mass. Quasi-Newton methods can then stabilize and accelerate the coupling iterations, while still using the solvers as black boxes and only accessing data at the fluid–structure interface. In this review, the IQN-ILS, IQN-MVJ, IBQN-LS, MVQN, IQN-IMVLS and IQN-ILSM methods are reformulated in the generalized Broyden framework to illustrate their similarities and differences. Also related coupling techniques are reviewed and a performance comparison is provided where available.

1 Introduction

Fluid–Structure Interaction (FSI) is the interaction of a fluid with a moving or deforming structure and occurs in many different branches of engineering. In mechanical engineering, the blades of a wind turbine deform due to their interaction with the wind [1, 2]. Also Flow-Induced Vibration (FIV) can occur, for example in tube bundles with external flow, leading to leakage or even rupture of the tubes [3]. In civil engineering, there are interactions between wind flow and bridges [4], silos [5], tents [6] and many other structures. In the biomedical field, heart valves and arteries are flexible structures that interact with the blood flow [7, 8].

As fluid–structure interaction is a multi-physics problem, complex phenomena can occur and numerical simulations are frequently used for the analysis. These numerical simulations of FSI can be performed in a monolithic or partitioned way. Monolithic codes solve the equations for the fluid and for the structure simultaneously, for example with a Newton–Raphson procedure [9] or multigrid method [10]. In this review, the focus is on the partitioned approach, as it allows to reuse mature and optimized codes to solve the subproblems.

Among the partitioned approaches, one can distinguish between the weakly and strongly coupled techniques. Weakly coupled techniques (also called explicit or loose coupling) solve the flow equations and the structure equations only once per time step [11, 12]. Consequently, the equilibrium between the fluid and the structure is not satisfied exactly. In this context, the term equilibrium refers to the equality of velocities and forces on the fluid–structure interface. These weakly coupled techniques are typically suitable for aeroelastic simulations with light and compressible fluids [13], but specific schemes can also be applied with dense and incompressible fluids [14].

By contrast, strongly coupled techniques (or implicit coupling) use coupling iterations between the flow solver and structure solver to enforce the equilibrium at the fluid–structure interface up to a convergence tolerance in a steady simulation or in each time step of an unsteady one [15–17]. As a result, the flow problem and structure problem are solved multiple times per (time) step. Obviously, these coupling iterations increase the computational cost, but the cost per coupling iteration normally decreases during the iterations within a (time) step as the change per iteration decreases. In the remainder of the paper an unsteady simulation will be

✉ Joris Degroote
Joris.Degroote@UGent.be

Nicolas Delaissé
Nicolas.Delaisse@UGent.be

[1] Department of Electromechanical, Systems and Metal Engineering, Ghent University, Sint-Pietersnieuwstraat 41, 9000 Ghent, Belgium

[2] Flanders Make @ UGent – Core Lab MIRO, Ghent, Belgium

[3] Department of Mathematics, Royal Military Academy, Renaissancelaan 30, 1000 Brussels, Belgium

Published online: 10 April 2023

assumed; a steady simulation is then a special case with one large time step.

An important parameter for the choice between weakly and strongly coupled techniques, but also for the stability of the coupling iterations in several strongly coupled techniques, is the ratio of the added mass to the mass of the structure [18]. The added mass is the apparent increase in mass of the structure due to the fluid that is displaced by the motion of this structure. Physically, it is influenced by the shape of the fluid domain and the density of the fluid. Numerically, also the time step size determines its effect, but this effect depends on whether the fluid is compressible or not. FSI problems with a compressible fluid can always be stabilized as long as the time step is sufficiently small, regardless of the ratio of the apparent added mass to the structural mass. However, for an incompressible fluid, stability cannot be obtained by decreasing the time step size [13, 15]. On the contrary, for incompressible flows in combination with flexible structures, decreasing the time step size may even increase the instability [15, 19–21]. For example, for the simulation of an elastic panel clamped at both ends and adjacent to a semi-infinite fluid domain, the added mass effect of a compressible fluid is proportional to the time step size, while for incompressible fluids it approaches a constant as the time step size decreases [13].

Especially for an incompressible fluid, many cases have a high added mass, e.g., blood flow in a vascular system [22], vibrations of tube bundles in lead-bismuth eutectic [23] or flutter of a slender cylinder in axial flow [24]. For these cases, the straightforward iteration between flow and structure solver within a time step will typically converge very slowly, or not at all, if no additional stabilization efforts are implemented. In this work, the focus is on techniques which consider the solvers as black boxes, as this is typically the case in a partitioned approach. Then, stabilization methods that alter one of the solvers, e.g., including an approximate added mass operator in the structural solver as in [25], are not possible. To stabilize and accelerate the convergence of coupling iterations with black box solvers, quasi-Newton methods have been developed in the fluid–structure interaction community. These methods will be reviewed in this work using the generalized Broyden framework.

The remainder of this review paper is structured as follows. First the FSI problem is posed and the necessary notation is introduced in Sect. 2. Then, the most basic solution approach is discussed in Sect. 3, with focus on its shortcomings in terms of stability and convergence speed, and how they can be overcome by introducing Jacobian information. In Sect. 4, a general method to obtain these Jacobians is discussed, called generalized Broyden. Thereafter, in Sect. 5, different quasi-Newton techniques are discussed in detail, including the Interface Quasi-Newton technique with an approximation for the Inverse of the Jacobian from a Least-Squares model (IQN-ILS), the Interface Quasi-Newton technique with Multi-Vector Jacobian (IQN-MVJ), the Interface Block Quasi-Newton technique with approximations from Least-Squares models (IBQN-LS), the Multi-Vector update Quasi-Newton technique (MVQN), the Interface Quasi-Newton Implicit Multi-Vector Least-Squares (IQN-IMVLS) and the Interface Quasi-Newton algorithm with an approximation for the Inverse of the Jacobian from a Least-Squares model and additional Surrogate Model (IQN-ILSM). This section ends with further notes and extensions on these methods. Finally, some numerical results to compare the different techniques are provided in Sect. 6, followed by the conclusions in Sect. 7.

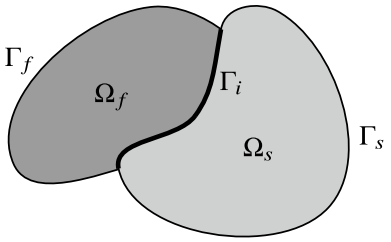


Fig. 1 The fluid subdomain Ω_f, the structure subdomain Ω_s, their boundaries Γ_f and Γ_s and the fluid–structure interface Γ_i

2 Formulation of the FSI Problem

An abstract fluid–structure interaction problem, as shown in Fig. 1, consists of the subdomains Ω_f and Ω_s, with the subscripts f and s denoting fluid and structure, respectively. The boundaries of the subdomains are denoted as $\Gamma_f = \partial\Omega_f$ and $\Gamma_s = \partial\Omega_s$ and the fluid–structure interface $\Gamma_i = \Gamma_f \cap \Gamma_s$ is their common boundary.

Besides having to satisfy the flow and structure equations in the respective subdomains while taking into account the appropriate boundary conditions on $\Gamma_f \setminus \Gamma_i$ and on $\Gamma_s \setminus \Gamma_i$, the solution of the FSI problem is also required to fulfill the equilibrium conditions on the fluid–structure interface Γ_i. The equilibrium conditions on a no-slip fluid–structure interface are twofold. First, the equality of fluid and solid velocity on Γ_i is needed (kinematic condition)

$$\vec{v} = \frac{\mathrm{D}\vec{u}}{\mathrm{D}t}, \tag{1}$$

where $\vec{v}$ is the velocity vector in the fluid domain and $\vec{u}$ the displacement vector in the structure domain. Remark that this equality also implies equal accelerations on the interface. Second, equal magnitude but opposite sense of traction on Γ_i is required (dynamic condition)

$$\bar{\sigma}_f \cdot \vec{n}_f = -\bar{\sigma}_s \cdot \vec{n}_s, \tag{2}$$

where $\bar{\sigma}_{f,s}$ is the stress tensor in $\Omega_{f,s}$ and $\vec{n}_{f,s}$ the unit normal vector that points outwards from the corresponding subdomain.

As this work discusses coupling techniques that consider the solvers as black boxes, only the variables on the fluid–structure interface Γ_i are of interest. However, the discretization of this interface is often different in the flow and structure subdomains. Given the focus of this review on coupling techniques, it is assumed that an interpolation layer is wrapped around or included in one (or both) of the solvers, invisible to the implementation of the coupling technique. As a consequence, the discretized displacement on either side of the fluid–structure interface can be represented as a column array $\boldsymbol{x} \in \mathbb{R}^{n_x \times 1}$ containing all components of the displacement vector $\vec{u}$ in each of the n_p grid points on the interface.

$$\boldsymbol{x} = \left[u_{1,1} \ \cdots \ u_{1,d} \ u_{2,1} \ \cdots \ u_{2,d} \ \cdots \ u_{n_p,1} \ \cdots \ u_{n_p,d}\right]^{\mathrm{T}}, \tag{3}$$

with the first subscript referring to the grid point (1 to n_p) and the second one to the component (1 to d, with d the dimension).

Similarly, the pressure p and all components of the viscous traction vector $\vec{t}$ in each load point (1 to n_l) on either side of the fluid–structure interface are grouped in a column array $\boldsymbol{y} \in \mathbb{R}^{n_y \times 1}$

$$\boldsymbol{y} = \left[p_1 \ t_{1,1} \ \cdots \ t_{1,d} \ \cdots \ p_{n_l,1} \ t_{n_l,1} \cdots \ t_{n_l,d}\right]^{\mathrm{T}}, \tag{4}$$

also called load vector, with the same meaning of the subscripts as above. Note that the n_l load points do not need to coincide with the discretization of the displacement. It is important that the pressure load $p \cdot \vec{n}$ and viscous traction $\vec{t}$ are not added, but included individually into $\boldsymbol{y}$, because the pressure is typically dominant and has to stay perpendicular to the surface, also when interpolation is performed. If pressure and viscous traction were added, the resulting interpolated vector would have a pressure contribution that is not necessarily perpendicular to the surface after interpolation, resulting in an artificial shear component that can be much larger than the physical shear component.

With the typical Dirichlet–Neumann decomposition of the FSI problem, the displacement (linked to the velocity through the time discretization in time-dependent problems) is imposed at the interface in the flow solver and a pressure and viscous traction distribution is applied on the interface in the structure solver. A flow solver with a deforming grid using the Arbitrary Lagrangian–Eulerian (ALE) frame of reference will be assumed for the explanation, but this can be replaced by other techniques, for example the combination of the ALE approach and the Chimera technique [26] to handle large body motions or non-conforming alternatives, such as Immersed Boundary Methods (IBM) [27] and Embedded Boundary Methods (EBM) [28], which can handle large deformations and even topology changes. The flow calculation in a coupling iteration within a time step can be written as

$$\boldsymbol{y} = \mathcal{F}(\boldsymbol{x}). \tag{5}$$

This notation concisely represents several operations and hides the dependence on previous time steps and the variables in the fluid domain next to the interface, while emphasizing the dependence on the discretized displacement $\boldsymbol{x}$ of the fluid–structure interface. It represents the following actions. First, the discretized displacement is given to the flow solver and the fluid domain adjacent to the interface is adapted accordingly. Then, the flow equations are solved in the entire fluid domain, resulting in a new load distribution $\boldsymbol{y}$ on the interface.

Similarly, the calculation of the structure is represented by the function

$$\boldsymbol{x} = \mathcal{S}(\boldsymbol{y}). \tag{6}$$

As before, this expression hides the dependence on both the previous time steps and the variables in the structure domain next to the interface. It indicates that the fluid pressure and viscous traction distribution on the interface $\boldsymbol{y}$ is given to the structure code. Subsequently, that code calculates the displacement of the entire structure and thus also the new displacement $\boldsymbol{x}$ of the fluid–structure interface.

With these notations, the FSI problem is formulated as the system

$$\begin{cases} \mathcal{F}(\boldsymbol{x}) = \boldsymbol{y} \\ \mathcal{S}(\boldsymbol{y}) = \boldsymbol{x} \end{cases} \tag{7}$$

that has to be solved for $\boldsymbol{x}$ and $\boldsymbol{y}$. This problem can be rewritten as the root-finding problem

$$\begin{cases} \mathcal{F}(\boldsymbol{x}) - \boldsymbol{y} = \boldsymbol{0} \\ \mathcal{S}(\boldsymbol{y}) - \boldsymbol{x} = \boldsymbol{0} \end{cases} \tag{8}$$

with unknowns $\boldsymbol{x}$ and $\boldsymbol{y}$.

Moreover, the system in Eq. (7) can be reduced by substituting one equation in the other. Commonly, the first line is substituted in the second, but the other way around is equally possible. In this way, the FSI problem is simplified to a smaller system of equations

$$\mathcal{S} \circ \mathcal{F}(\boldsymbol{x}) = \boldsymbol{x}, \tag{9}$$

which has to be solved for $\boldsymbol{x}$. The notation $\circ$ refers to function composition, so $\mathcal{S} \circ \mathcal{F}(\boldsymbol{x})$ is equivalent with $\mathcal{S}(\mathcal{F}(\boldsymbol{x}))$. This looks like a fixed-point equation for $\boldsymbol{x}$, but can also be written as a root-finding problem with unknown $\boldsymbol{x}$

$$\mathcal{S} \circ \mathcal{F}(\boldsymbol{x}) - \boldsymbol{x} = \boldsymbol{0}. \tag{10}$$

To write this more compactly, the residual operator $\mathcal{R}(\cdot)$ is defined as

$$\mathcal{R}(\boldsymbol{x}) = \mathcal{S} \circ \mathcal{F}(\boldsymbol{x}) - \boldsymbol{x}, \tag{11}$$

with output $\boldsymbol{r} = \mathcal{R}(\boldsymbol{x})$. The FSI problem thus reduces to finding the $\boldsymbol{x}$ that fulfills

$$\mathcal{R}(\boldsymbol{x}) = \boldsymbol{0}. \tag{12}$$

In this section, we have presented two formulations of the FSI problem. The first is the complete system Eq. (7) with $n_x + n_y$ unknowns. The second is the reduced system Eq. (9), which has the benefit of having only n_x unknowns. This system has been written more compactly using the residual operator $\mathcal{R}$ resulting in Eq. (12). In the next sections, several methods are discussed to solve the FSI problem presented here, in one of both formulations.

Both the solver operators as well as the residual operator are typically nonlinear. Therefore, the FSI problem exhibits similarities with nonlinear root-finding problems. The main difference is that an FSI problem usually involves time stepping (except for steady cases), which means that a nonlinear system has to be solved in each time step. Therefore, within each time step, coupling iterations are performed until the solution is reached. The nonlinear systems in subsequent time steps are somehow related to each other, because the solver operators change only gradually in time. As the solution is typically continuous, the initial guess for $\boldsymbol{x}$ at the start of each time step can be obtained by extrapolating the solution from previous time steps [29].

3 Solving the FSI Problem

3.1 Gauss–Seidel Scheme

In order to solve the FSI problem, Eq. (8) has to be solved in each time step. One of the basic methods to solve such a system of nonlinear equations is the block Gauss–Seidel scheme. In this block-iterate scheme, each of the nonlinear equations is solved for one of the unknowns consecutively, and each unknown is updated to its new value as soon as it becomes available.

Because, further on, it will become necessary to make a distinction between the output of one solver and the input of the next, a tilde symbol is introduced to indicate the output of a solver:

$$\tilde{\boldsymbol{y}} = \mathcal{F}(\boldsymbol{x}) \quad \text{and} \quad \tilde{\boldsymbol{x}} = \mathcal{S}(\boldsymbol{y}). \tag{13}$$

Using the superscript $k+1$ to indicate the current iteration, the block Gauss–Seidel scheme takes the following form

$$\boldsymbol{x}^{k+1} = \tilde{\boldsymbol{x}}^{k} \tag{14a}$$

$$\tilde{\boldsymbol{y}}^{k+1} = \mathcal{F}(\boldsymbol{x}^{k+1}) \tag{14b}$$

$$\boldsymbol{y}^{k+1} = \tilde{\boldsymbol{y}}^{k+1} \tag{14c}$$

$$\tilde{\boldsymbol{x}}^{k+1} = \mathcal{S}(\boldsymbol{y}^{k+1}) \tag{14d}$$

The lastly calculated displacement vector $\tilde{\boldsymbol{x}}^k$ is used as $\boldsymbol{x}^{k+1}$, the input of the flow solver in the following iteration. Subsequently, this vector is used to calculate a new load vector $\tilde{\boldsymbol{y}}^{k+1}$ which is thereafter used as input of the structure solver $\boldsymbol{y}^{k+1}$, to calculate a new displacement vector $\tilde{\boldsymbol{x}}^{k+1}$. This iteration scheme, in which the output of the flow and structure solver is passed unchanged to the structure and flow solver, respectively, is the most basic way to find an equilibrium and is also called Gauss–Seidel scheme or fixed point iteration scheme.

The final solution of the FSI problem Eq. (7) has to fulfill the kinematic Eq. (1) and dynamic equilibrium condition Eq. (2) up to a certain tolerance. This means that $\tilde{\boldsymbol{x}}$ and $\tilde{\boldsymbol{y}}$ have to approach $\boldsymbol{x}$ and $\boldsymbol{y}$, respectively, which is expressed by the convergence conditions

$$\left\| \tilde{\boldsymbol{x}}^{k+1} - \boldsymbol{x}^{k+1} \right\|_2 \leq \epsilon_x \tag{15a}$$

$$\left\| \tilde{\boldsymbol{y}}^{k+1} - \boldsymbol{y}^{k} \right\|_2 \leq \epsilon_y. \tag{15b}$$

Because the output of each of the solvers is passed unchanged to the other, this can also be written as

$$\left\| \tilde{\boldsymbol{x}}^{k+1} - \tilde{\boldsymbol{x}}^{k} \right\|_2 \leq \epsilon_x \tag{16a}$$

$$\left\| \tilde{\boldsymbol{y}}^{k+1} - \tilde{\boldsymbol{y}}^{k} \right\|_2 \leq \epsilon_y, \tag{16b}$$

which relates to the fixed point formulation in Eq. (9).

By eliminating the occurrence of the load vector $\boldsymbol{y}$, the procedure can be simplified to

$$\boldsymbol{x}^{k+1} = \tilde{\boldsymbol{x}}^{k} \tag{17a}$$

$$\tilde{\boldsymbol{x}}^{k+1} = \mathcal{S} \circ \mathcal{F}(\boldsymbol{x}^{k+1}) \tag{17b}$$

Furthermore, with the use of the residual operator introduced in Eq. (11), the iteration scheme becomes

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^{k} + \boldsymbol{r}^{k} \tag{18a}$$

$$\boldsymbol{r}^{k+1} = \mathcal{R}(\boldsymbol{x}^{k+1}), \tag{18b}$$

which is considered converged once

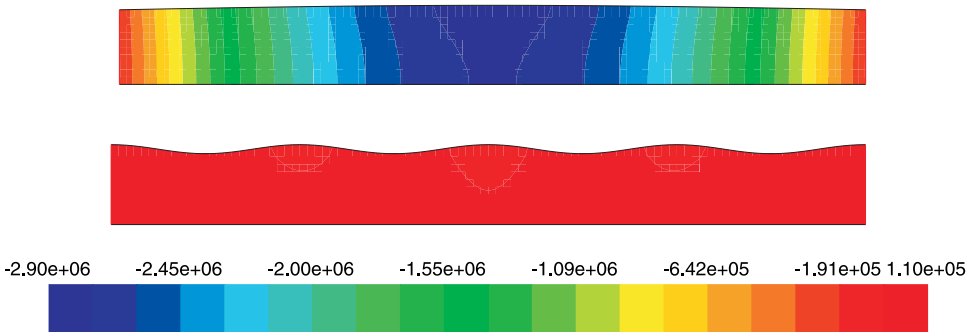


Fig. 2 The pressure contours (in Pa) in an axisymmetric tube due to two displacements of the tube's wall with the same amplitude but a different wave number. Initially, the fluid is at rest and the tube has a constant cross-section and zero pressure at both ends. A displacement of the tube's wall with a low wave number (top) creates much larger pressure variations than a displacement with a high wave number (bottom). Only the difference between the two calculations and not the values as such are important [36]

$$\left\| \boldsymbol{r}^{k+1} \right\|_2 \leq \epsilon_x. \tag{19}$$

3.2 Motivation for Using Quasi-Newton Methods

Unfortunately, the Gauss–Seidel scheme explained above is not unconditionally stable due to among others the added mass effect.

Many researchers have investigated the stability of Gauss–Seidel iterations. For example, its convergence behaviour has been studied based on a simple model problem with a single degree of freedom on the interface [30]. Some investigated the added-mass effect [31]. Many others have explored the case of blood flow through a simplified artery [15, 32, 33]. They observed that, besides an apparent fluid mass of the similar order of magnitude as the actual structural mass, also a decrease in the stiffness of the structure or increase in domain length has a destabilizing effect. A first attempt to mathematically analyze the stability was done through the determination of the maximum relaxation factor to obtain convergence [15, 25].

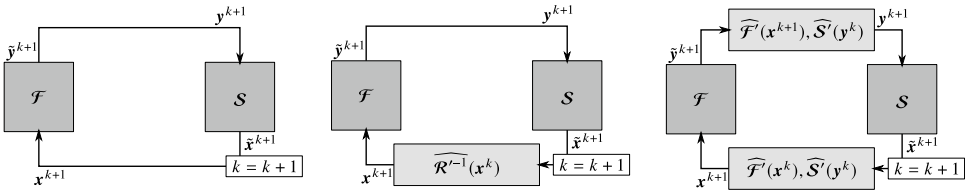
Instead of looking at a single number, the stability of a Gauss–Seidel scheme for a simplified flexible tube model with Dirichlet–Neumann decomposition can be examined by splitting the error on the interface into Fourier modes [34]. The mentioned error is the difference between the correct interface displacement and the one in a Gauss–Seidel iteration, based on linearized equations and without taking the boundary conditions into account. In this way, the authors were able to identify which frequency components become unstable. The analysis was first performed for a tube wall without inertia [34] and thereafter repeated including inertia [35], which proved to stabilize the convergence behaviour.

From this analysis it can be deduced that only a limited number of modes of the interface displacement are unstable and that the lowest wave numbers have the highest amplification factor and are hence the most unstable ones. This observation is true for different combinations of parameter values. In other words, the divergence or slow convergence of Gauss–Seidel iterations are caused by a limited number of unstable and slowly converging modes corresponding to the lowest wave numbers.

The physical explanation for this observation is shown in Fig. 2. The figure shows an axisymmetric tube, the wall of which is perturbed with two different wave numbers, while on the in- and outlet a zero pressure boundary condition is imposed. Initially, its cross section is constant and the incompressible fluid is at rest. In the upper part of Fig. 2, a low wave number perturbation is applied and, because the fluid is incompressible, it is accelerated globally resulting in large pressure variations. In the lower part, a higher wave number perturbation is applied and the fluid acceleration is confined to more local regions. As a consequence, the pressure variations are much smaller for higher wave numbers. The pressure variations in the lower part of Fig. 2 are even barely visible, because the same scale is used for both cases.

Although the above analysis was performed on a flexible tube, the results are more widely applicable to incompressible fluids with a fluid–structure density ratio around one. For example, [37] arrived at the same conclusions by examining the stability of Gauss–Seidel iterations for a semi-infinite open fluid domain bounded by a string or a beam.

In summary, Gauss–Seidel iterations are not suitable for incompressible fluid cases with high added mass, because there is a limited number of error modes that are unstable. In order to obtain a solution for these cases, the unstable modes have to be removed by another technique to (efficiently) achieve convergence. Based on the results from the Fourier decomposition, it follows that only the low



(a) Gauss-Seidel scheme.

(b) Residual formulation quasi-Newton scheme.

(c) Block iteration quasi-Newton scheme.

Fig. 3 Schematic representation of different iteration schemes

wave number modes have to be stabilized, while the others can still be treated using Gauss–Seidel iteration. The next section explains that the stabilization of these modes is achieved by including derivative information, which is the basic principle behind quasi-Newton techniques.

3.3 Quasi-Newton Schemes

In order to overcome this limitation of the Gauss–Seidel iterations for problems with high added mass and an incompressible fluid, the quasi-Newton iteration scheme is adopted. To improve stability, one or both of the vectors $\tilde{\boldsymbol{x}}$ and $\tilde{\boldsymbol{y}}$ are modified before passing them to the other solver. If only one solver output is adapted, it is usually the output of the structural solver $\boldsymbol{\mathcal{S}}$ as in Fig. 3b, but the opposite is equally possible. Figure 3c shows a schematic representation of adapting both solver outputs.

In the remainder of this section we will first introduce the adaptation of one output where we will use the modification of the structural output as example. This scheme will be referred to as the residual formulation scheme. Thereafter, the adaptation of both solver outputs will be introduced. The corresponding scheme is called the block iteration scheme.

3.3.1 Residual Formulation Quasi-Newton Scheme

In this scheme, the output $\tilde{\boldsymbol{x}}^k$ of the structural solver is modified to $\boldsymbol{x}^{k+1}$ which is subsequently used as input for the flow solver. The output $\tilde{\boldsymbol{y}}^k$ of the flow solver is passed unchanged to the structural solver. Therefore, the load vector $\boldsymbol{y}$ can be left out altogether, as was the case for Gauss–Seidel iterations. With the use of the residual operator, defined in Eq. (11), the residual $\boldsymbol{r}^{k+1}$ in iteration $k+1$ is written as

$$\boldsymbol{r}^k = \tilde{\boldsymbol{x}}^k - \boldsymbol{x}^k = \boldsymbol{\mathcal{S}} \circ \boldsymbol{\mathcal{F}}(\boldsymbol{x}^k) - \boldsymbol{x}^k = \boldsymbol{\mathcal{R}}(\boldsymbol{x}^k) \tag{20}$$

and as before convergence is reached when Eq. (19) is satisfied. The difference with the Gauss–Seidel scheme is that $\boldsymbol{x}^{k+1}$ is no longer equal to $\tilde{\boldsymbol{x}}^k$. The adaption of the displacement vector follows from the use of a Newton–Raphson approach to solve the root-finding problem Eq. (12). This method uses the Jacobian of the nonlinear equation, which is denoted here by $\boldsymbol{\mathcal{R}}'$, to estimate the input $\boldsymbol{x}^{k+1}$ that will direct the residual to $\boldsymbol{0}$ by solving

$$\boldsymbol{\mathcal{R}}'(\boldsymbol{x}^k)(\boldsymbol{x}^{k+1} - \boldsymbol{x}^k) = \boldsymbol{0} - \boldsymbol{r}^k \tag{21}$$

for $\boldsymbol{x}^{k+1}$. Note that Gauss–Seidel iteration Eq. (18) is retrieved if $\boldsymbol{\mathcal{R}}'(\boldsymbol{x}^k) = -\boldsymbol{I}$. Likewise, relaxed Gauss–Seidel iteration is obtained if the Jacobian is $-\omega\boldsymbol{I}$.

Because both the flow and structure solvers are considered black box solvers, the Jacobians of $\boldsymbol{\mathcal{F}}$ and $\boldsymbol{\mathcal{S}}$ are not accessible and hence, neither is $\boldsymbol{\mathcal{R}}'$. Therefore, the Jacobian of the residual operator is approximated, resulting in a quasi-Newton method

$$\widehat{\boldsymbol{\mathcal{R}}'}(\boldsymbol{x}^k)(\boldsymbol{x}^{k+1} - \boldsymbol{x}^k) = \boldsymbol{0} - \boldsymbol{r}^k, \tag{22}$$

where $\widehat{\boldsymbol{\mathcal{R}}'}(\boldsymbol{x}^k)$ is the approximated Jacobian.

As explained in the previous section, the instability of Gauss–Seidel iterations is caused by a limited set of modes, i.e., for the vectors $\boldsymbol{x}$ in a small subspace of $\mathbb{R}^{n_x \times 1}$. Consequently, an approximation of the complete Jacobian of the residual operator $\boldsymbol{\mathcal{R}}$ is not required. An approximated Jacobian which takes care of these unstable modes and leaves the other modes unchanged is sufficient. Leaving some modes unchanged means that the quasi-Newton method will actually perform Gauss–Seidel iterations for those modes.

Solving the linear system in the equation above can be avoided by approximating the inverse of the Jacobian directly and calculating the update of the displacement vector as

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \widehat{\boldsymbol{\mathcal{R}}'^{-1}}(\boldsymbol{x}^k)\, \boldsymbol{r}^k. \tag{23}$$

To conclude this section, a new short-hand notation is introduced for the approximate (inverse) Jacobians. For a nonlinear function $\boldsymbol{r} = \boldsymbol{\mathcal{R}}(\boldsymbol{x})$, the approximate Jacobian and inverse Jacobian are written as

$$\partial_{\boldsymbol{x}}^k \boldsymbol{r} \equiv \widehat{\boldsymbol{\mathcal{R}}'}(\boldsymbol{x}^k) \tag{24a}$$

$$\partial_{\boldsymbol{r}}^k \boldsymbol{x} \equiv \widehat{\boldsymbol{\mathcal{R}}'^{-1}}(\boldsymbol{x}^k). \tag{24b}$$

3.3.2 Block Iteration Quasi-Newton Scheme

Instead of only adapting the output of one solver, it is also possible to adapt the output of both the flow and structure solver. Now both $\boldsymbol{x}^{k+1}$ and $\boldsymbol{y}^{k+1}$ are different from $\tilde{\boldsymbol{x}}^k$ and $\tilde{\boldsymbol{y}}^{k+1}$, and, because the load vector is no longer passed unchanged, it is not possible to use the residual operator. The convergence conditions are again given by Eq. (15).

The modification of the output of the solvers is determined by applying block Newton–Raphson iterations to the root-finding problem Eq. (8) with unknowns $\boldsymbol{x}$ and $\boldsymbol{y}$

$$\begin{bmatrix} \mathcal{F}'(\boldsymbol{x}) & -\boldsymbol{I} \\ -\boldsymbol{I} & \mathcal{S}'(\boldsymbol{y}) \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{x} \\ \Delta\boldsymbol{y} \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{0} \end{bmatrix} - \begin{bmatrix} \mathcal{F}(\boldsymbol{x}) - \boldsymbol{y} \\ \mathcal{S}(\boldsymbol{y}) - \boldsymbol{x} \end{bmatrix}, \tag{25}$$

where $\Delta\boldsymbol{x}$ and $\Delta\boldsymbol{y}$ are the updates for the input $\boldsymbol{x}$ and $\boldsymbol{y}$ of the flow and structure solvers, respectively. Further, $\boldsymbol{I}$ is the identity matrix and $\mathcal{F}'$ and $\mathcal{S}'$ are the Jacobians of the flow and structure equations. Note that the two identity matrices will have different dimensions if the size of $\boldsymbol{x}$ and $\boldsymbol{y}$ differ.

Starting from the displacement $\boldsymbol{x}^k$ that was given as input to the flow solver in the previous coupling iteration, the displacement $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \Delta\boldsymbol{x}^k$ is calculated by solving the system

$$\left(\boldsymbol{I} - \mathcal{S}'(\boldsymbol{y}^k)\mathcal{F}'(\boldsymbol{x}^k)\right)\Delta\boldsymbol{x}^k = \tilde{\boldsymbol{x}}^k - \boldsymbol{x}^k + \mathcal{S}'(\boldsymbol{y}^k)(\tilde{\boldsymbol{y}}^k - \boldsymbol{y}^k) \tag{26}$$

for $\Delta\boldsymbol{x}^k$.

Using the updated value $\boldsymbol{x}^{k+1}$ and after calling the flow solver to determine $\tilde{\boldsymbol{y}}^{k+1} = \mathcal{F}(\boldsymbol{x}^{k+1})$, the pressure and viscous traction distribution $\boldsymbol{y}^{k+1} = \boldsymbol{y}^k + \Delta\boldsymbol{y}^k$ is calculated by solving the analogous system

$$\begin{aligned} \left(\boldsymbol{I} - \mathcal{F}'(\boldsymbol{x}^{k+1})\mathcal{S}'(\boldsymbol{y}^k)\right)\Delta\boldsymbol{y}^k =& \tilde{\boldsymbol{y}}^{k+1} - \boldsymbol{y}^k \\ &+ \mathcal{F}'(\boldsymbol{x}^{k+1})(\tilde{\boldsymbol{x}}^k - \boldsymbol{x}^{k+1}) \end{aligned} \tag{27}$$

for $\Delta\boldsymbol{y}^k$. Subsequently, the structure solver is called to determine $\tilde{\boldsymbol{x}}^{k+1} = \mathcal{S}(\boldsymbol{y}^{k+1})$.

Similar to the previous section, the Jacobians are not accessible, because the solvers are considered black boxes. Therefore, approximations denoted by $\widehat{\mathcal{F}'}$ and $\widehat{\mathcal{S}'}$ are used instead. Note that here two normal Jacobians are required, one for each solver, whereas in the previous section only one inverse Jacobian was required, namely the inverse Jacobian of the residual operator.

Adopting the same short-hand for the approximated Jacobians as in the previous section results in the following notations

$$\partial_{\boldsymbol{x}}^k\tilde{\boldsymbol{y}} \equiv \widehat{\mathcal{F}'}(\boldsymbol{x}^k) \tag{28a}$$

$$\partial_{\boldsymbol{y}}^k\tilde{\boldsymbol{x}} \equiv \widehat{\mathcal{S}'}(\boldsymbol{y}^k). \tag{28b}$$

4 Approximating Jacobians

The previous section introduced quasi-Newton approaches to stabilize and at the same time accelerate the convergence of coupling iterations. These schemes adapt either one or both of the solver outputs before passing them on, resulting in respectively, a quasi-Newton system for the residual formulation of the FSI problem, or a block iteration quasi-Newton system. These systems each contain one or more approximate Jacobians. The residual formulation scheme requires an approximation for the inverse of the Jacobian $\mathcal{R}'(\boldsymbol{x}^k)$, which is denoted as $\partial_{\boldsymbol{r}}^k\boldsymbol{x}$. The block iteration quasi-Newton scheme requires approximations for the Jacobians of the flow solver and structure solver, so $\partial_{\boldsymbol{x}}^k\tilde{\boldsymbol{y}}$ approximates $\mathcal{F}'(\boldsymbol{x}^k)$ and $\partial_{\boldsymbol{y}}^k\tilde{\boldsymbol{x}}$ approximates $\mathcal{S}'(\boldsymbol{y}^k)$. In these notations the superscript k refers to the iteration in which the Jacobian has been approximated.

All of these approximate Jacobians can be created using the *generalized Broyden* method. In this section, we will explain this method for the construction of an approximate Jacobian of an arbitrary nonlinear function $\boldsymbol{b} = \mathcal{B}(\boldsymbol{a})$. For now, we leave out the added complexity of FSI problems, for which an approximate Jacobian has to be constructed in each time step. This will be explained in the next section. Here, we just have an iterative method, where in each iteration k the Jacobian $\mathcal{B}'(\boldsymbol{a}^k)$ is approximated by $\partial_{\boldsymbol{a}}^k\boldsymbol{b}$. The same technique can also be used to approximate the inverse Jacobian $\mathcal{B}'(\boldsymbol{a}^k)^{-1}$ by $\partial_{\boldsymbol{b}}^k\boldsymbol{a}$.

Instead of immediately presenting the rather complex generalized Broyden equation, it is introduced step by step, in a way that better fits the quasi-Newton FSI explanations found in literature.

4.1 Satisfying the Secant Conditions

The core idea of any quasi-Newton Jacobian approximation is to use the nonlinear function input–output information from previous iterations. Indeed, an input $\boldsymbol{a}^i$ resulting in a certain output $\boldsymbol{b}^i$ is a piece of valuable information about the behavior of the black box function $\mathcal{B}$, which can be used to approximate the Jacobian $\mathcal{B}'(\boldsymbol{a}^k)$ by $\partial_{\boldsymbol{a}}^k\boldsymbol{b}$. In the current iteration $k+1$, the inputs of all previous iterations

$$\boldsymbol{a}^0, \boldsymbol{a}^1, \ldots, \boldsymbol{a}^{k-1}, \boldsymbol{a}^k \tag{29}$$

are available, as well as the corresponding outputs

$$\boldsymbol{b}^0, \boldsymbol{b}^1, \ldots, \boldsymbol{b}^{k-1}, \boldsymbol{b}^k. \tag{30}$$

The input–output info is stored and used in the form of differences between consecutive iterations, defined as

$$\delta\boldsymbol{a}^i \equiv \boldsymbol{a}^{i+1} - \boldsymbol{a}^i \tag{31a}$$

$$\delta\boldsymbol{b}^i \equiv \boldsymbol{b}^{i+1} - \boldsymbol{b}^i \tag{31b}$$

for $0 \leq i \leq k-1$. The δ notation refers to the difference between previous iterations, in contrast to the Δ notation, which refers to the desired change or update that needs to be performed.

Each pair $(\delta \boldsymbol{a}^i, \delta \boldsymbol{b}^i)$ is called the *secant information* at iterations i and is related to a secant line to the nonlinear function $\boldsymbol{\mathcal{B}}$. Therefore, it can be interpreted as a finite difference approximation for the Jacobian in the direction $\delta \boldsymbol{a}^i$.

Furthermore, each secant information pair has a corresponding secant equation:

$$\delta \boldsymbol{b}^i = \partial_{\boldsymbol{a}}^k \boldsymbol{b} \; \delta \boldsymbol{a}^i. \tag{32}$$

If the approximated Jacobian $\partial_{\boldsymbol{a}}^k \boldsymbol{b}$ meets this *secant condition*, it uses a finite difference approximation for the actual Jacobian in the direction of $\delta \boldsymbol{a}^i$, with the input–output information of iterations i and $i+1$. This secant information is relevant only if the Jacobian stays more or less the same during the k iterations, which means that $\boldsymbol{\mathcal{B}}(\boldsymbol{a})$ has to behave close to linearly in the neighbourhood of $\boldsymbol{a}^k$.

The idea is to construct $\partial_{\boldsymbol{a}}^k \boldsymbol{b}$, so that it fulfills all the k secant equations. To write this compactly, the differences defined previously in Eq. (31) are stored in the matrices $\boldsymbol{A}^k$ and $\boldsymbol{B}^k$ as follows

$$\boldsymbol{A}^k = \begin{bmatrix} \delta \boldsymbol{a}^{k-1} & \delta \boldsymbol{a}^{k-2} & \cdots & \delta \boldsymbol{a}^1 & \delta \boldsymbol{a}^0 \end{bmatrix} \tag{33a}$$

$$\boldsymbol{B}^k = \begin{bmatrix} \delta \boldsymbol{b}^{k-1} & \delta \boldsymbol{b}^{k-2} & \cdots & \delta \boldsymbol{b}^1 & \delta \boldsymbol{b}^0 \end{bmatrix}. \tag{33b}$$

Now, the k secant conditions can be collected in the matrix equation

$$\boldsymbol{B}^k = \partial_{\boldsymbol{a}}^k \boldsymbol{b} \, \boldsymbol{A}^k. \tag{34}$$

With n_a and n_b being the length of the input and output vectors, this is a system of $n_b k$ scalar equations for $n_b n_a$ unknowns (the elements of the matrix $\partial_{\boldsymbol{b}}^k \boldsymbol{a}$). The system is thus typically underdetermined ($k < n_a$). In order to find a unique solution, the least-norm solution is sought, which is in this case defined as the smallest matrix in the Frobenius norm that satisfies all secant conditions. The solution is given as

$$\partial_{\boldsymbol{a}}^k \boldsymbol{b} = \boldsymbol{B}^k \boldsymbol{A}^{k+}, \tag{35}$$

where $\boldsymbol{A}^{k+}$ is the pseudo-inverse[1] (or Moore–Penrose inverse) of the rectangular matrix $\boldsymbol{A}^k$, defined as

$$\boldsymbol{A}^{k+} = \left(\boldsymbol{A}^{k\mathrm{T}} \boldsymbol{A}^k \right)^{-1} \boldsymbol{A}^{k\mathrm{T}}. \tag{36}$$

To calculate the pseudo-inverse, it is necessary that the columns of $\boldsymbol{A}^k$ are linearly independent. For now, we will assume this is always the case and the issue of linear dependence of the secant information is addressed in detail in the discussion on filtering in Sect. 5.1.

The expression for the approximate Jacobian presented above is elegant and short, but not very intuitive. Therefore, a different approach to obtain the same expression is given below.

The purpose of the approximate Jacobian is to determine an estimated change in output $\Delta \boldsymbol{b}$ that corresponds to an arbitrary change in input $\Delta \boldsymbol{a}$, by evaluating

$$\Delta \boldsymbol{b} = \partial_{\boldsymbol{a}}^k \boldsymbol{b} \; \Delta \boldsymbol{a}. \tag{37}$$

For this purpose, the secant information from the previous iterations is utilised in the following approach.

First, the arbitrary vector $\Delta \boldsymbol{a}$ is approximated as a linear combination of vectors $\delta \boldsymbol{a}^i$, i.e.

$$\Delta \boldsymbol{a} \approx \boldsymbol{A}^k \boldsymbol{c} \tag{38}$$

with $\boldsymbol{c} \in \mathbb{R}^{k \times 1}$ a coefficient vector.

It follows from the secant information that an input difference $\delta \boldsymbol{a}^i$ corresponds to an output difference $\delta \boldsymbol{b}^i$, for $0 \leq i \leq k-1$. Therefore, and under the assumption that the linear behavior of $\boldsymbol{\mathcal{B}}$ is locally dominant, it can be stated that a linear combination of vectors $\delta \boldsymbol{a}^i$ will correspond to the same linear combination of vectors $\delta \boldsymbol{b}^i$. This principle allows to determine $\Delta \boldsymbol{b}$ as

$$\Delta \boldsymbol{b} = \boldsymbol{B}^k \boldsymbol{c}. \tag{39}$$

Finally, it remains to determine the coefficients $\boldsymbol{c}$. The system in Eq. (38) is typically overdetermined. Hence, the least-squares solution for $\boldsymbol{c}$ will be used, which can be obtained by solving the square system of *normal equations*

$$\boldsymbol{A}^{k\mathrm{T}} \Delta \boldsymbol{a} = \boldsymbol{A}^{k\mathrm{T}} \boldsymbol{A}^k \boldsymbol{c}. \tag{40}$$

Therefore, the coefficient vector is given as

$$\boldsymbol{c} = \boldsymbol{A}^{k+} \Delta \boldsymbol{a}. \tag{41}$$

Using this to calculate $\Delta \boldsymbol{b}$ results in

$$\Delta \boldsymbol{b} = \boldsymbol{B}^k \boldsymbol{c} = \boldsymbol{B}^k \boldsymbol{A}^{k+} \Delta \boldsymbol{a}. \tag{42}$$

Comparison with Eq. (37) reveals the same Jacobian as determined before in Eq. (35).

Matrix-free implementation Some of the algorithms explained in Sect. 5 require the explicit construction of the Jacobian matrix, while for others only its product with a vector, e.g., $\partial_{\boldsymbol{a}}^k \boldsymbol{b} \; \Delta \boldsymbol{a}$, is required. This last set of algorithms

[1] Note that for $\boldsymbol{A}^k \in \mathbb{R}^{n_a \times k}$ with $n_a > k$, $\boldsymbol{A}^{k+} \boldsymbol{A}^k$ is the identity matrix of size $k \times k$, while $\boldsymbol{A}^k \boldsymbol{A}^{k+}$ is a $n_a \times n_a$ matrix of rank k if the columns of $\boldsymbol{A}^k$ are linearly independent.

allows matrix-free implementation, for which the Jacobian matrix Eq. (35) never has to be calculated explicitly in practice, nor is the explicit calculation of the pseudo-inverse defined in Eq. (36) needed. How this is achieved is explained here.

Equations (40) and (41) show that the product of the pseudo-inverse with a vector is in fact the solution of the normal equations, but solving the normal equations Eq. (40) becomes unstable if the number of columns in the matrix $\boldsymbol{A}^k$ is rather high. A more robust method to calculate the pseudo-inverse uses the reduced or economy-size QR decomposition [38] of $\boldsymbol{A}^k$

$$\boldsymbol{A}^k = \boldsymbol{Q}_A^k \boldsymbol{R}_A^k, \tag{43}$$

where $\boldsymbol{Q}_A^k \in \mathbb{R}^{n_a \times k}$ is a matrix with orthonormal columns and $\boldsymbol{R}_A^k \in \mathbb{R}^{k \times k}$ is an upper triangular matrix.[2] Applying this to the normal equations Eq. (40) and using the fact that the inverse of $\boldsymbol{R}_A^k$ exists because the columns of $\boldsymbol{A}^k$ are linearly independent, results in

$$\boldsymbol{R}_A^k \, \boldsymbol{c} = \boldsymbol{Q}_A^{k\,\mathrm{T}} \Delta \boldsymbol{a}. \tag{44}$$

Symbolically, this means that the pseudo-inverse can be written as $\boldsymbol{R}_A^{k\,-1} \boldsymbol{Q}_A^{k\,\mathrm{T}}$, but it should never be constructed or stored. Instead, the product of the pseudo-inverse with a vector can be calculated by first evaluating the right hand side of Eq. (44) and subsequently solving the system using back-substitution, as $\boldsymbol{R}_A^k$ is an upper triangular matrix. The complete procedure to efficiently determine $\Delta \boldsymbol{b}$ given $\Delta \boldsymbol{a}$ is summarized in Algorithm 1.

In the following, the notation with the pseudo-inverse will still be used. Nonetheless, it should be kept in mind that the actual calculation has to be done using QR decomposition and back-substitution, avoiding the calculation of the inverse of matrices as well as the construction of large dense square matrices.

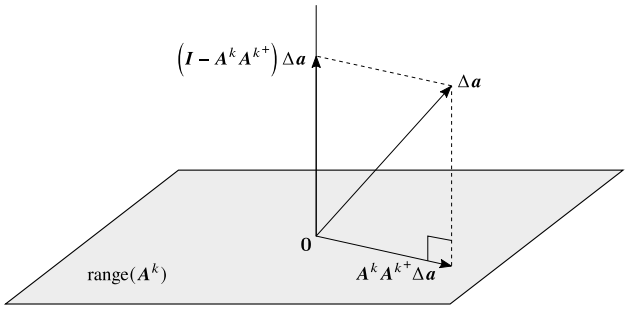


Fig. 4 The vector $\Delta \boldsymbol{a}$ is split into a part inside the range of $\boldsymbol{A}^k$ and another part perpendicular to that range

4.2 Adding an Initial Estimate for the Jacobian

Assuming the columns $\boldsymbol{A}^k$ are linearly independent, the above obtained approximated Jacobian $\boldsymbol{B}^k \boldsymbol{A}^{k+}$ is of rank k. For the current discussion, it is assumed that $k \ll n_a$. Therefore, the matrix is a low-rank Jacobian approximation, and has an *image* or *range* of dimension k and a *nullspace* of dimension $n_a - k$. As a result, with regard to its product with an arbitrary $\Delta \boldsymbol{a}$, only the part of $\Delta \boldsymbol{a} \in \text{range}(\boldsymbol{A}^k)$ will have a non-zero result. This becomes clear from the definition of the pseudo inverse in Eq. (36). The part of $\Delta \boldsymbol{a} \perp \text{range}(\boldsymbol{A}^k)$ falls in the nullspace of the approximated Jacobian and the product of this part with the Jacobian is therefore zero. In other words, the approximated Jacobian is zero in every direction that is not a linear combination of the directions $\delta \boldsymbol{a}^i$, for $0 \leq i \leq k-1$, encountered in the previous iterations.

Nonetheless, a full rank approximation of the Jacobian may be required, e.g., when it is used in a quasi-Newton method according to the residual formulation scheme. If this is the case, the current Jacobian approximation, using the approximation based on secant conditions for the part of $\Delta \boldsymbol{a} \in \text{range}(\boldsymbol{A}^k)$, can be expanded with an initial estimate of the approximate Jacobian $\partial_{\boldsymbol{a}}^0 \boldsymbol{b}$ for the remaining part $\Delta \boldsymbol{a} \perp \text{range}(\boldsymbol{A}^k)$.

Algorithm 1 Determination of $\Delta \boldsymbol{b}$, given $\Delta \boldsymbol{a}$, with QR decomposition and back-substitution. It is assumed that the columns of $\boldsymbol{A}^k$ are linearly independent.

1: construct $\boldsymbol{A}^k$ and $\boldsymbol{B}^k$ using Eqs. (33)
2: calculate QR decomposition $\boldsymbol{A}^k = \boldsymbol{Q}_A^k \boldsymbol{R}_A^k$
3: solve $\boldsymbol{R}_A^k \boldsymbol{c} = \boldsymbol{Q}_A^{k\,\mathrm{T}} \Delta \boldsymbol{a}$ with back-substitution
4: $\Delta \boldsymbol{b} = \boldsymbol{B}^k \boldsymbol{c}$

The splitting of $\Delta \boldsymbol{a}$ in these two parts is based on orthogonal projection and visualized in Fig. 4. The orthogonal projection of a vector $\Delta \boldsymbol{a}$ onto the range of range$(\boldsymbol{A}^k)$ is given by

[2] Note that $\boldsymbol{Q}_A^k \boldsymbol{Q}_A^{k\,\mathrm{T}} = \boldsymbol{A}^k \boldsymbol{A}^{k+} \neq \boldsymbol{I}$ and that $\boldsymbol{Q}_A^{k\,\mathrm{T}} \boldsymbol{Q}_A^k = \boldsymbol{A}^{k+} \boldsymbol{A}^k = \boldsymbol{I}$ equals the identity matrix of size $k \times k$.

$$\boldsymbol{A}^k\left(\boldsymbol{A}^{k\mathrm{T}}\boldsymbol{A}^k\right)^{-1}\boldsymbol{A}^{k\mathrm{T}}\Delta\boldsymbol{a} = \boldsymbol{A}^k\boldsymbol{A}^{k+}\Delta\boldsymbol{a}. \tag{45}$$

This is the part of $\Delta\boldsymbol{a} \in \text{range}(\boldsymbol{A}^k)$. Using the complementary projector or just calculating the difference of $\Delta\boldsymbol{a}$ and its orthogonal projection

$$\Delta\boldsymbol{a} - \boldsymbol{A}^k\boldsymbol{A}^{k+}\Delta\boldsymbol{a} = \left(\boldsymbol{I} - \boldsymbol{A}^k\boldsymbol{A}^{k+}\right)\Delta\boldsymbol{a}, \tag{46}$$

gives the part of $\Delta\boldsymbol{a} \perp \text{range}(\boldsymbol{A}^k)$. Refer to [39] for a more complete discussion of projectors. Moreover, note that, using the QR decomposition these two parts are given by $\boldsymbol{Q}_A^k\boldsymbol{Q}_A^{k\mathrm{T}}\Delta\boldsymbol{a}$ and $(\boldsymbol{I} - \boldsymbol{Q}_A^k\boldsymbol{Q}_A^{k\mathrm{T}})\Delta\boldsymbol{a}$.

Now, the Jacobian approximation based on secant information can be extended with an initial Jacobian $\partial_{\boldsymbol{a}}^0\boldsymbol{b}$:

$$\partial_{\boldsymbol{a}}^k\boldsymbol{b} = \boldsymbol{B}^k\boldsymbol{A}^{k+} + \partial_{\boldsymbol{a}}^0\boldsymbol{b}\left(\boldsymbol{I} - \boldsymbol{A}^k\boldsymbol{A}^{k+}\right) \tag{47}$$

The questions why and how an initial Jacobian can be added have been answered. What remains is the choice of its value. Often, the identity matrix is used, scaled with a factor, typically -1 or $-\omega$, which corresponds to (relaxed) Gauss–Seidel iteration, as explained below Eq. (21). This is the simplest approach to obtaining a full rank Jacobian approximation and will also be used in Sect. 5. In the case a low rank approximation suffices, e.g., the Jacobians for block iteration quasi-Newton, $\partial_{\boldsymbol{a}}^0\boldsymbol{b} = \boldsymbol{0}$ can be used, which means the second term disappears completely. In still other situations, a physics-based surrogate may be available to use as initial Jacobian. This approach may accelerate convergence, but is application-specific and will be discussed further in Sect. 5.6.

4.3 Generalized Broyden Method

Up to now, the approximation of the Jacobian $\partial_{\boldsymbol{a}}^k\boldsymbol{b}$ was determined such that it met all secant conditions. However, this is not the only way to use the secant information. Another often used method (although not in FSI) is to only require the approximated Jacobian to fulfill the latest secant equation. Therefore, the matrices $\boldsymbol{A}^k$ and $\boldsymbol{B}^k$ only contain the latest piece of secant information:

$$\boldsymbol{A}^k = \left[\delta\boldsymbol{a}^{k-1}\right] \tag{48}$$

$$\boldsymbol{B}^k = \left[\delta\boldsymbol{b}^{k-1}\right]. \tag{49}$$

For all vectors $\Delta\boldsymbol{a} \perp \text{range}(\boldsymbol{A}^k)$ (i.e., $\Delta\boldsymbol{a} \perp \delta\boldsymbol{a}^{k-1}$), we want to use the previous Jacobian $\partial_{\boldsymbol{a}}^{k-1}\boldsymbol{b}$. In other words, the effect of the approximated Jacobian remains unchanged in all directions orthogonal to $\delta\boldsymbol{a}^{k-1}$. This is called the *no-change condition*, which can be written formally as

$$\partial_{\boldsymbol{a}}^k\boldsymbol{b}\ \Delta\boldsymbol{a} = \partial_{\boldsymbol{a}}^{k-1}\boldsymbol{b}\ \Delta\boldsymbol{a} \qquad \forall\Delta\boldsymbol{a} \perp \text{range}(\boldsymbol{A}^k). \tag{50}$$

To obtain a Jacobian approximation with these specifications, $\partial_{\boldsymbol{a}}^0\boldsymbol{b}$ is replaced by $\partial_{\boldsymbol{a}}^{k-1}\boldsymbol{b}$:

$$\partial_{\boldsymbol{a}}^k\boldsymbol{b} = \boldsymbol{B}^k\boldsymbol{A}^{k+} + \partial_{\boldsymbol{a}}^{k-1}\boldsymbol{b}\left(\boldsymbol{I} - \boldsymbol{A}^k\boldsymbol{A}^{k+}\right) \tag{51}$$

where $\boldsymbol{A}^k$ and $\boldsymbol{B}^k$ now contain only one column. This is a recursive expression for the approximated Jacobian. In fact, this is Broyden's original method,[3] to construct the approximate Jacobian [40]. It was developed in the sixties, to solve systems of nonlinear equations.

Furthermore, Broyden's method can be generalized. Instead of using only one secant condition and the approximate Jacobian from the previous iteration, m secant conditions can be used in combination with the approximate Jacobian from m iterations ago. This gives rise to the *generalized Broyden* method

$$\partial_{\boldsymbol{a}}^k\boldsymbol{b} = \boldsymbol{B}^k\boldsymbol{A}^{k+} + \partial_{\boldsymbol{a}}^{k-m}\boldsymbol{b}\left(\boldsymbol{I} - \boldsymbol{A}^k\boldsymbol{A}^{k+}\right) \tag{52}$$

with

$$\boldsymbol{A}^k = \left[\delta\boldsymbol{a}^{k-1}\ \delta\boldsymbol{a}^{k-2}\ \cdots\ \delta\boldsymbol{a}^{k-m}\right] \tag{53a}$$

$$\boldsymbol{B}^k = \left[\delta\boldsymbol{b}^{k-1}\ \delta\boldsymbol{b}^{k-2}\ \cdots\ \delta\boldsymbol{b}^{k-m}\right]. \tag{53b}$$

This equation for the approximate Jacobian in generalized Broyden, however, can also be obtained in a more formal way, namely as the unique matrix that satisfies a number of conditions. Two equivalent ways are described in [41].

First, the approximated Jacobian can be obtained as the only matrix that simultaneously satisfies the m secant conditions in Eq. (34) and the $n_a - m$ no-change conditions

$$\partial_{\boldsymbol{a}}^k\boldsymbol{b}\ \Delta\boldsymbol{a} = \partial_{\boldsymbol{a}}^{k-m}\boldsymbol{b}\ \Delta\boldsymbol{a} \qquad \forall\Delta\boldsymbol{a} \perp \text{range}(\boldsymbol{A}^k). \tag{54}$$

Secondly, it can be obtained as the unique matrix that satisfies the m secant conditions Eq. (34) and minimizes the difference with the approximate Jacobian from m iterations ago, i.e.

$$\left\|\partial_{\boldsymbol{a}}^k\boldsymbol{b} - \partial_{\boldsymbol{a}}^{k-m}\boldsymbol{b}\right\|_F, \tag{55}$$

where the subscript F denotes the Frobenius norm.

[3] Approximating the Jacobian, respectively the inverse Jacobian, results in the *good Broyden's method* respectively the *bad Broyden's method*.

Furthermore, previously discussed methods are retrieved by choosing certain values for the parameter m in the generalized Broyden method. For $m = 1$, Broyden's original method is recovered, while for $m = k$ the pure secant method from Sects. 4.1 and 4.2 is obtained.

The generalized Broyden method was established much later than Broyden's original method. The first extension to the original one in the eighties led to a rather complex modified Broyden method [42, 43]. In the nineties, Eyert [44] simplified this method by removing some nonessential parameters, resulting in the generalized Broyden method presented here.

Around that same time, the connection between the generalized Broyden method and Anderson acceleration (or Anderson mixing) was discovered. Anderson acceleration [45] was introduced in the sixties to accelerate fixed-point iterations. Based on the work by Van Leuken [46], Eyert showed that Anderson acceleration is mathematically equivalent to generalized Broyden with $m = k$, i.e., the pure secant method introduced in Sects. 4.1 and 4.2. This is not immediately apparent due to the very different ideas on which Anderson and Broyden originally based their methods.

In partitioned FSI simulations, several variants of the generalized Broyden method are used to approximate Jacobians in quasi-Newton iterations. These techniques were developed independently from the older methods (Anderson, Broyden and generalized Broyden) and the correspondence to those methods was only discovered recently [47, 48].

Because a nonlinear system of equations has to be solved in every time step of an FSI simulation, there are some particularities with respect to Jacobian approximation, such as the reuse of secant information from previous time steps, as well as the removal of old and irrelevant secant information. These topics are discussed in the next section.

Computational complexity and storage This section ends with a first look into the computational complexity to obtain and use these approximate Jacobians. For simplicity, it is assumed that $\boldsymbol{a}$ and $\boldsymbol{b}$ are both vectors of length n_a. This is usually not true for the block methods, but n_b is typically proportional to n_a. Further, it is assumed that $n_a \gg k$, i.e., the length of the vectors are much larger than the number of secant pairs available. More details will be provided later on for the different FSI methods. No details about the number of operations will be given, only the complexity of the leading-order term will be discussed.

At the basis of the generalized Broyden method is the economy-size QR decomposition of $\boldsymbol{A}^k$, which is used for determination of the pseudo-inverse of $\boldsymbol{A}^k$. This QR decomposition is typically done with Householder transformations, resulting in a complexity of $\mathcal{O}(n_a m^2)$, which is also the total complexity of the evaluation of the product of this pseudo-inverse with a vector. Already, it can be noted that, in the case that $m = k$, the computational cost quickly rises relative to a low fixed value for m.

If the approximate Jacobian is only needed to calculate its product with a vector, its explicit construction can be avoided. In some algorithms of the next section, however, the approximate Jacobian is used explicitly. Then, the construction of this $n_a \times n_a$ matrix has a complexity of $\mathcal{O}(n_a^2 m)$. In addition, the $n_a \times n_a$ matrix requires a storage capacity $\mathcal{O}(n_a^2)$, which is a strong disadvantage of these select algorithms.

In other algorithms, it is possible to avoid this expensive construction and use a matrix-free method to multiply the approximate Jacobian with a vector, i.e., without large dense square matrices. In practice, this is done by evaluating the product using Eq. (47) and multiplying the factors within each term from right to left. Then the complexity of this evaluation is only $\mathcal{O}(n_a m^2)$, which is the complexity of performing the QR decomposition needed to evaluate the product of the pseudo-inverse of $\boldsymbol{A}^k$ with a vector. Because only the secant information has to be stored, the storage requirements $\mathcal{O}(n_a m)$ are lower as well.

4.4 Difference Between the Anderson and Broyden Approach

The previous part formulated the generalized Broyden method, in which the parameter m determines how the secant information from previous iterations is included. Setting $m = k$ corresponds to the Anderson method, in which the approximated Jacobian is determined by imposing all secant equations directly. For $m = 1$, Broyden's original method is retrieved, here simply referred to as the Broyden method, in which the approximated Jacobian only fulfills the latest secant equation and the secant information from the previous iterations is included indirectly by imposing no-change conditions. In this section, the difference in behaviour between these two extreme versions of the generalized Broyden method will be clarified.

Consider for example the approximation of the Jacobian $\mathcal{B}'(\boldsymbol{a}^k)$ by $\partial_{\boldsymbol{a}}^k \boldsymbol{b}$, when previously three iterations have been performed ($k = 2$). The matrices containing the differences between consecutive iterations are

$$\boldsymbol{A}^2 = \left[\delta\boldsymbol{a}^1 \; \delta\boldsymbol{a}^0\right] \tag{56a}$$

$$\boldsymbol{B}^2 = \left[\delta\boldsymbol{b}^1 \; \delta\boldsymbol{b}^0\right] \tag{56b}$$

Without loss of generality, it is stated that

$$\delta\boldsymbol{a}^1 = \boldsymbol{p} \tag{57a}$$

$$\delta\boldsymbol{a}^0 = x\boldsymbol{p} + y\boldsymbol{q}, \tag{57b}$$

Table 1 Multiplication of Anderson and Broyden Jacobian approximations corresponding to the example discussed in the text with a vector $\Delta \boldsymbol{a}$

$\Delta \boldsymbol{a}$	$\Delta \boldsymbol{b} = \partial_a^2 \boldsymbol{b}\ \Delta \boldsymbol{a}$	
	Anderson	Broyden
$\delta \boldsymbol{a}^1 = 1\boldsymbol{p} + 0\boldsymbol{q}$	$1\delta \boldsymbol{b}^1 + 0\delta \boldsymbol{b}^0$	$1\delta \boldsymbol{b}^1 + 0\delta \boldsymbol{b}^0$
$\delta \boldsymbol{a}^0 = x\boldsymbol{p} + y\boldsymbol{q}$	$0\delta \boldsymbol{b}^1 + 1\delta \boldsymbol{b}^0$	$x\delta \boldsymbol{b}^1 + \frac{y^2}{x^2+y^2}\delta \boldsymbol{b}^0$
$0\boldsymbol{p} + 1\boldsymbol{q}$	$-\frac{x}{y}\delta \boldsymbol{b}^1 + \frac{1}{y}\delta \boldsymbol{b}^0$	$0\delta \boldsymbol{b}^1 + \frac{y}{x^2+y^2}\delta \boldsymbol{b}^0$
$u\boldsymbol{p} + v\boldsymbol{q} + w\boldsymbol{s}$	$\left(u - \frac{vx}{y}\right)\delta \boldsymbol{b}^1 + \frac{v}{y}\delta \boldsymbol{b}^0$	$u\delta \boldsymbol{b}^1 + \frac{yv}{x^2+y^2}\delta \boldsymbol{b}^0$

where $\boldsymbol{p}$ and $\boldsymbol{q}$ are orthonormal vectors, and x and y real scalars.

In the Anderson approach ($m = k$), the approximated Jacobian is

$$\partial_a^2 \boldsymbol{b} = \boldsymbol{B}^2 \boldsymbol{A}^{2+}. \tag{58}$$

The QR decomposition of $\boldsymbol{A}^2$ is given by

$$\boldsymbol{R}_A^2 = \begin{bmatrix} 1 & x \\ 0 & y \end{bmatrix} \tag{59a}$$

$$\boldsymbol{Q}_A^2 = \begin{bmatrix} \boldsymbol{p} & \boldsymbol{q} \end{bmatrix}. \tag{59b}$$

With this decomposition, the pseudo-inverse is calculated

$$\boldsymbol{A}^{2+} = \left(\boldsymbol{A}^{2\mathrm{T}}\boldsymbol{A}^2\right)^{-1}\boldsymbol{A}^{2\mathrm{T}} = \boldsymbol{R}_A^{2\,-1}\boldsymbol{Q}_A^{2\,\mathrm{T}} = \frac{1}{y}\begin{bmatrix} y\boldsymbol{p}^{\mathrm{T}} - x\boldsymbol{q}^{\mathrm{T}} \\ \boldsymbol{q}^{\mathrm{T}} \end{bmatrix}. \tag{60}$$

Finally, the approximate Jacobian is given by

$$\partial_a^2 \boldsymbol{b} = \begin{bmatrix} \delta \boldsymbol{b}^1 & \delta \boldsymbol{b}^0 \end{bmatrix} \begin{bmatrix} \boldsymbol{p}^{\mathrm{T}} - \frac{x}{y}\boldsymbol{q}^{\mathrm{T}} \\ \frac{1}{y}\boldsymbol{q}^{\mathrm{T}} \end{bmatrix}. \tag{61}$$

In the Broyden approach ($m = 1$), the approximated Jacobian is

$$\partial_a^2 \boldsymbol{b} = \delta \boldsymbol{b}^1 \delta \boldsymbol{a}^{1+} + \partial_a^1 \boldsymbol{b}\left(\boldsymbol{I} - \delta \boldsymbol{a}^1 \delta \boldsymbol{a}^{1+}\right) \text{ with } \partial_a^1 \boldsymbol{b} = \delta \boldsymbol{b}^0 \delta \boldsymbol{a}^{0+}. \tag{62}$$

Note that the pseudo-inverse of a single column equals its transpose divided by its norm squared, such that

$$\delta \boldsymbol{a}^{1+} = \boldsymbol{p}^{\mathrm{T}} \tag{63a}$$

$$\delta \boldsymbol{a}^{0+} = \frac{x\boldsymbol{p}^{\mathrm{T}} + y\boldsymbol{q}^{\mathrm{T}}}{x^2 + y^2}. \tag{63b}$$

The resulting approximated Jacobian is given by

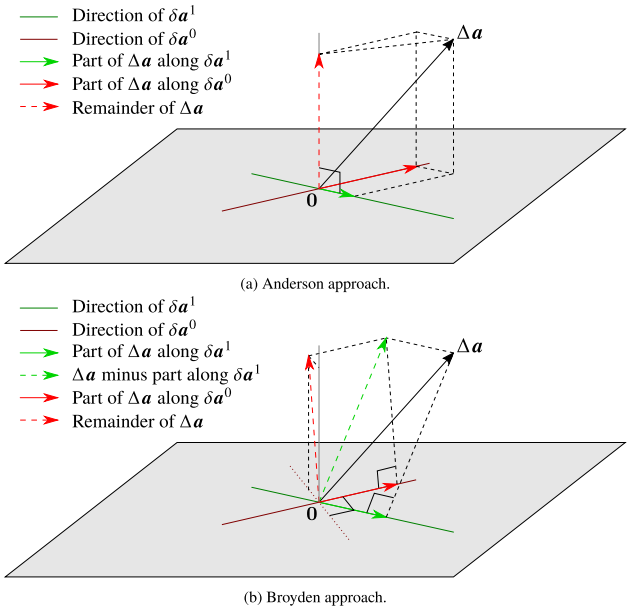


Fig. 5 The vector $\Delta \boldsymbol{a}$ is decomposed along the directions of the previously determined differences. The decomposition is different for the Anderson and Broyden approach. The green line is the direction of the lastly determined difference vector $\delta \boldsymbol{a}^1$, the red line corresponds to the one before last $\delta \boldsymbol{a}^0$. The red dotted vector is the remaining part after decomposition. The addition of the parts along $\delta \boldsymbol{a}^1$ and $\delta \boldsymbol{a}^0$ and the remaining part gives the original $\Delta \boldsymbol{a}$. (Color figure online)

$$\partial_a^2 \boldsymbol{b} = \begin{bmatrix} \delta \boldsymbol{b}^1 & \delta \boldsymbol{b}^0 \end{bmatrix} \begin{bmatrix} \boldsymbol{p}^{\mathrm{T}} \\ \frac{y}{x^2+y^2}\boldsymbol{q}^{\mathrm{T}} \end{bmatrix}. \tag{64}$$

Comparing Eq. (61) with Eq. (64), it is clear that the Jacobian approximations are different. Their inequality is analyzed in Table 1 by looking at their product with particular vectors $\Delta \boldsymbol{a}$.

For a vector $\Delta \boldsymbol{a}$ equal to the lastly added difference $\delta \boldsymbol{a}^1$, both approaches return the corresponding difference $\delta \boldsymbol{b}^1$, as expected. If the one before last vector $\delta \boldsymbol{a}^0$ is supplied, the results are different. The Anderson method simply returns $\delta \boldsymbol{b}^0$, as this method attempts to approximate $\Delta \boldsymbol{a}$ as closely as possible using the already available differences. In contrast, the Broyden method does not and returns a linear combination of $\delta \boldsymbol{b}^1$ and $\delta \boldsymbol{b}^0$. This approach gives priority to the lastly determined difference $\delta \boldsymbol{a}^1$ and uses the corresponding $\delta \boldsymbol{b}^1$ for the orthogonal projection of $\Delta \boldsymbol{a}$ on that difference $\delta \boldsymbol{a}^1$. For a result of the Broyden approach that lies along $\delta \boldsymbol{b}^0$, a difference orthogonal to the last difference $\delta \boldsymbol{a}^1$ needs to be supplied. Finally, the result for a general vector is given, where u, v and w are arbitrary scalars and $\boldsymbol{s}$ a unit vector orthogonal to $\boldsymbol{p}$ and $\boldsymbol{q}$.

Both methods decompose $\Delta \boldsymbol{a}$ in components along the previously determined vectors $\delta \boldsymbol{a}^i$, $0 < i < k - 1$, and

multiply the respective components with the corresponding vectors $\delta \boldsymbol{b}^i$. This is shown graphically in Fig. 5.

The Anderson method projects $\Delta \boldsymbol{a}$ on all previously determined differences. Therefore, the remaining part

$$\left(\boldsymbol{I} - \boldsymbol{A}^2\boldsymbol{A}^{2^{\mathrm{T}}}\right)\Delta \boldsymbol{a} = \left(\boldsymbol{I} - \boldsymbol{Q}_A^2\boldsymbol{Q}_A^{2^{\mathrm{T}}}\right)\Delta \boldsymbol{a} = \left(\boldsymbol{I} - \boldsymbol{p}\boldsymbol{p}^{\mathrm{T}} - \boldsymbol{q}\boldsymbol{q}^{\mathrm{T}}\right)\Delta \boldsymbol{a} \tag{65}$$

is orthogonal to these differences. The Broyden method projects $\Delta \boldsymbol{a}$ first on the lastly obtained difference, and the leftover part on the one before last, and so on. Therefore, the remaining part

$$\begin{aligned}&\left(\boldsymbol{I} - \delta \boldsymbol{a}^0\delta \boldsymbol{a}^{0+}\right)\left(\boldsymbol{I} - \delta \boldsymbol{a}^1\delta \boldsymbol{a}^{1+}\right)\Delta \boldsymbol{a}\\ &\quad = \left(\boldsymbol{I} - \boldsymbol{p}\boldsymbol{p}^{\mathrm{T}} - \left(\frac{x\boldsymbol{p} + y\boldsymbol{q}}{x^2 + y^2}\right)\boldsymbol{q}^{\mathrm{T}}\right)\Delta \boldsymbol{a}\end{aligned} \tag{66}$$

is not necessarily orthogonal to these differences. It will, however, always be orthogonal to the last difference onto which the projection was made, i.e., the oldest difference.

The difference between the two methods is essential to how nonlinearities in the secant information are dealt with. In general $\boldsymbol{\mathcal{B}}(\boldsymbol{a}^k)$ is nonlinear and its Jacobian is not constant, therefore the secant information will also contain nonlinear effects, especially when the step $\delta \boldsymbol{a}^i$ is large. Because the Broyden method prioritizes more recent secant information, it effectively ignores these nonlinearities, while the Anderson method does not, as it wants to approximate $\Delta \boldsymbol{a}$ as closely as possible using all available differences. This can lead to instabilities in the Anderson method. However, the Broyden method will also neglect small linear information, slowing down the convergence speed. More details and a method to remove nonlinearities from the secant information to stabilize Anderson are found in [49].

In the FSI community, the Anderson method is referred to as the *least-squares* approach and the Broyden method can be linked to the *multi-vector* approach. However, in FSI, fulfilling only the most recent secant equation as in the original Broyden method is typically not done and the multi-vector algorithms fulfill the secant equations in the most recent time step, using no-change conditions for older time steps. So in fact, the multi-vector approach is a generalized Broyden method, as will be explained in the following section.

5 Quasi-Newton Methods for FSI

In Sect. 3, different quasi-Newton schemes have been introduced. They required approximate Jacobians, which could be determined in different ways using information from previous iterations, as explained in Sect. 4. Up to this point, the focus was on solving the nonlinear equations in each time step separately. From here on, the distinction between different time steps will be necessary. Therefore, the superscript $n+1$ will be used to indicate the values from the current time step, meaning that these are the values that are currently calculated. This notation is similar to the superscript $k+1$, which indicates the current iteration.

In this section, the IQN-ILS, IBQN-LS, IQN-MVJ, MVQN, IQN-IMVLS and IQN-ILSM techniques will be derived and analyzed in the generalized Broyden framework. These techniques for partitioned FSI simulation have several differences, as summarized in Table 2.

The first difference is whether they use only the interface displacement as variables (IQN-ILS, IQN-MVJ, IQN-IMVLS, IQN-ILSM) or whether they are block iteration quasi-Newton methods using both interface displacement and load (IBQN-LS, MVQN). In the former case, they solve Eq. (12), in the latter they use Eq. (8), as explained in Sect. 3.

The second difference is related to how time stepping is handled, as most FSI simulations are time-dependent to capture a vibration or other dynamic behaviour. Assuming the inputs and outputs of the q previous time steps are stored, one can either impose the secant conditions from all time steps (IQN-ILS, IBQN-LS) or only for the latest time step,

Table 2 Main differences between the quasi-Newton methods

Variables	Conditions	
	Secant for all time steps	Only secant for last time step
Only interface displacement	IQN-ILS	IQN-MVJ, IQN-IMVLS, IQN-ILSM
Interface displacement and load	IBQN-LS	MVQN

Table 3 Overview of computational complexity and memory requirements for the different methods. Note that typically $m \ll n_x$.

Method	Computational complexity	Memory requirements
IQN-ILS	$\mathcal{O}(n_x m^2)$	$\sim n_x$
IBQN-LS	$\mathcal{O}(n_x m^2)$	$\sim n_x$
MVQN	$\mathcal{O}(n_x^2)$	$\sim n_x^2$
IQN-MVJ	$\mathcal{O}(n_x^2)$	$\sim n_x^2$
IQN-IMVLS[†]	$\mathcal{O}(n_x q \bar{k})$	$\sim n_x$
IQN-ILSM (reuse)[†]	$\mathcal{O}(n_x q \bar{k})$	$\sim n_x$

[†] $\bar{k}$ denotes the average number of coupling iterations per time step

combined with no-change conditions for previous time steps (IQN-MVJ, MVQN, IQN-IMVLS, IQN-ILSM). This second difference is thus related to the choice of the parameter m of generalized Broyden, as explained in Sect. 4. On the one hand, m can be set to ∞ (actually limited to q time steps), so all the info from q previous time steps is used together, without an old Jacobian, but only with an initial one to start the procedure. In fact, this corresponds with the Anderson approach and is typically termed *least-squares* approach in FSI. On the other hand, only the secant info from the current time step can be used, with an old approximate Jacobian that is the final one from the previous time step. This corresponds to $m = k$, called *multi-vector*, and is really generalized Broyden, and not one of the limiting cases (Anderson or Broyden).

The third difference is the amount of memory required for the storage of the approximate Jacobian(s) and the computational time required for the calculations related to the quasi-Newton steps. This will be explained more in detail for each method below and will be summarized in Table 3.

5.1 IQN-ILS

IQN-ILS is the abbreviation for Interface Quasi-Newton technique with an approximation for the Inverse of the Jacobian from a Least-Squares model [50]. The IQN-ILS technique performs an update of the input for the flow solver in each coupling iteration, using an approximation for the inverse of the Jacobian of the residual operator, so

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \Delta\boldsymbol{x}^k \quad \text{with} \quad \Delta\boldsymbol{x}^k = \widehat{\mathcal{R}'^{-1}}(\boldsymbol{x}^k)\Delta\boldsymbol{r}^k \tag{67}$$

and $\Delta\boldsymbol{r}^k = \boldsymbol{0} - \boldsymbol{r}^k$. The approximation for the inverse Jacobian $\widehat{\mathcal{R}'^{-1}}(\boldsymbol{x}^k) \equiv \partial_{\boldsymbol{r}}^k\boldsymbol{x}$ can be obtained directly by following the method explained in Sect. 4.3 for $\partial_{\boldsymbol{a}}^k\boldsymbol{b}$, with $\boldsymbol{a} = \boldsymbol{r}$ and $\boldsymbol{b} = \boldsymbol{x}$. Because the approximation needs to be full rank for a working quasi-Newton method, an initial Jacobian $\partial_{\boldsymbol{r}}^{k-m}\boldsymbol{x} = -\boldsymbol{I}$ is used, which is the Jacobian of Gauss–Seidel iteration, as explained below Eq. (21). In the literature, however, the approximation for the inverse of the Jacobian $\mathcal{R}'^{-1}(\boldsymbol{x}^k)$ is usually rewritten using the identity $\boldsymbol{r} = \tilde{\boldsymbol{x}} - \boldsymbol{x}$, giving

$$\widehat{\mathcal{R}'^{-1}}(\boldsymbol{x}^k) \equiv \partial_{\boldsymbol{r}}^k\boldsymbol{x} = \partial_{\boldsymbol{r}}^k(\tilde{\boldsymbol{x}} - \boldsymbol{r}) = \partial_{\boldsymbol{r}}^k\tilde{\boldsymbol{x}} - \boldsymbol{I}, \tag{68}$$

where the operator $\partial_{\boldsymbol{r}}^k\tilde{\boldsymbol{x}}$ is constructed as explained in Sect. 4.3, with $\boldsymbol{a} = \boldsymbol{r}$ and $\boldsymbol{b} = \tilde{\boldsymbol{x}}$. In this way of explaining, no initial Jacobian is used, so $\partial_{\boldsymbol{r}}^{k-m}\tilde{\boldsymbol{x}} = \boldsymbol{0}$. It is worth mentioning that if instead of the inverse, the Jacobian $\mathcal{R}'(\boldsymbol{x}^k)$ is approximated, the Interface Quasi-Newton Least-Squares method (IQN-LS) is retrieved [51].

For an FSI simulation with a single step (e.g., a steady simulation), the generalized Broyden formula in Eq. (52) is used with $m = k$ and without initial guess $\partial_{\boldsymbol{r}}^0\tilde{\boldsymbol{x}} = \boldsymbol{0}$, so

$$\partial_{\boldsymbol{r}}^k\tilde{\boldsymbol{x}} = \widetilde{\boldsymbol{X}}^k\boldsymbol{R}^{k+} \tag{69}$$

with

$$\boldsymbol{R}^k = \begin{bmatrix}\delta\boldsymbol{r}^{k-1} & \delta\boldsymbol{r}^{k-2} & \cdots & \delta\boldsymbol{r}^0\end{bmatrix} \tag{70a}$$

$$\widetilde{\boldsymbol{X}}^k = \begin{bmatrix}\delta\tilde{\boldsymbol{x}}^{k-1} & \delta\tilde{\boldsymbol{x}}^{k-2} & \cdots & \delta\tilde{\boldsymbol{x}}^0\end{bmatrix}. \tag{70b}$$

Note that $\partial_{\boldsymbol{r}}^k\tilde{\boldsymbol{x}}$ has at most rank k, while $\widehat{\mathcal{R}'^{-1}}(\boldsymbol{x}^k) \equiv \partial_{\boldsymbol{r}}^k\boldsymbol{x}$ is full-rank.[4]

In a time-dependent simulation, the secant information from the q previous time steps can be reused. As notation, the previous time steps are indicated with $n, n-1, \cdots, n+1-q$, for the time step that is being calculated the superscript $n+1$ is omitted and only the superscript k is used. The matrices $\boldsymbol{R}^{[k]}$ and $\widetilde{\boldsymbol{X}}^{[k]}$ are a concatenation of the matrices from the different time steps, giving

$$\boldsymbol{R}^{[k]} = \begin{bmatrix}\boldsymbol{R}^k & \boldsymbol{R}^n & \cdots \boldsymbol{R}^{n+1-q}\end{bmatrix} \tag{71a}$$

$$\widetilde{\boldsymbol{X}}^{[k]} = \begin{bmatrix}\widetilde{\boldsymbol{X}}^k & \widetilde{\boldsymbol{X}}^n & \cdots & \widetilde{\boldsymbol{X}}^{n+1-q}\end{bmatrix}. \tag{71b}$$

In this way, the information from each time step is treated equally, except when linear dependencies occur, because these are then removed by filtering, as will be explained below. The method thus satisfies all available secant conditions, i.e., from time step $n+1$ and the q previous time steps. Consequently, m is equal to the number of columns in $\boldsymbol{R}^{[k]}$ and $\widetilde{\boldsymbol{X}}^{[k]}$, which is

$$m = k^{n+1} + k^n + \cdots + k^{n+1-q}, \tag{72}$$

if no filtering is applied. As no initial Jacobian $\partial_{\boldsymbol{r}}^{k-m}\tilde{\boldsymbol{x}}$ is used, the information from earlier time steps $n < n+1-q$ is not considered. It is important to remark that the difference between the first $\boldsymbol{r}$ or $\tilde{\boldsymbol{x}}$ of a time step and the last one from the previous time step is not used. Only differences between vectors of the same time steps are taken into account. This approach ensures that the secant information matches with the meaning of the Jacobian that is being approximated, which is the derivative within a time step, and not between them.

The reuse parameter q has to be defined by the user. Reuse typically improves the performance, but too old data is no longer helping the convergence, and therefore an optimal value exists. In the literature, the existence of this parameter

[4] The term Jacobian is used for both $\partial_{\boldsymbol{r}}^k\boldsymbol{x}$ and $\partial_{\boldsymbol{r}}^k\tilde{\boldsymbol{x}}$, depending on which is relevant in the section.

is often cited as a drawback of IQN-ILS, because the performance of the method would be sensitive to this parameter. However, by using filtering, the performance of the method is rendered rather insensitive to this parameter around the optimum, as is shown by numerical tests in Sect. 6 and in other work [52, 53]. Moreover, the parameter q allows the user to control how many time steps can be considered relevant, which is important in cases with rapid changes from one time step to the next, e.g., with multi-phase flows [54].

Matrix-free implementation Equation (69) is a symbolic notation to write IQN-ILS in the generalized Broyden framework, but this matrix should never be constructed or stored in the computer's memory. One of the main benefits of IQN-ILS is its so-called matrix-free character, which means that no large square matrices need to be constructed or stored. The product of the approximation of the inverse of the Jacobian with $\Delta \boldsymbol{r}^k = -\boldsymbol{r}^k$ in Eq. (67) is symbolically calculated as

$$\Delta \boldsymbol{x}^k = \widehat{\mathcal{R}'^{-1}}(\boldsymbol{x}^k)\, \Delta \boldsymbol{r}^k = (\widetilde{\boldsymbol{X}}^{[k]} \boldsymbol{R}^{[k]+} - \boldsymbol{I})\, \Delta \boldsymbol{r}^k. \tag{73}$$

In practice, the product $\boldsymbol{c}^k = \boldsymbol{R}^{[k]+}\, \Delta \boldsymbol{r}^k$ of the pseudo-inverse of $\boldsymbol{R}^{[k]}$ with $\Delta \boldsymbol{r}^k$ is calculated using the economy-size QR decomposition and back-substitution, resulting in

$$\boldsymbol{R}_R^{[k]} \boldsymbol{c}^k = \boldsymbol{Q}_R^{[k]\mathrm{T}}\, \Delta \boldsymbol{r}^k. \tag{74}$$

as explained in Algorithm 1. The vector $\Delta \boldsymbol{r}^k$ is thus written as a linear combination of the $\delta \boldsymbol{r}^i$, resulting in coefficients $\boldsymbol{c}^k$. As each $\delta \boldsymbol{r}^i$ has a corresponding $\delta \tilde{\boldsymbol{x}}^i$, the change in $\tilde{\boldsymbol{x}}$ corresponding with $\Delta \boldsymbol{r}^k$ can be obtained by calculating $\widetilde{\boldsymbol{X}}^{[k]} \boldsymbol{c}^k$.

The complete procedure can be found in Algorithm 2, with a relaxation step with factor ω on line 7, for the case in which $\boldsymbol{R}^{[k]}$ and $\widetilde{\boldsymbol{X}}^{[k]}$ do not have any columns, e.g., at the beginning of the simulation.

Using $\boldsymbol{R}^{[k]} = \widetilde{\boldsymbol{X}}^{[k]} - \boldsymbol{X}^{[k]}$, where $\boldsymbol{X}^{[k]}$ is defined analogous to Eq. (71), Eq. (73) can be rewritten as

$$\Delta \boldsymbol{x}^k = \boldsymbol{X}^{[k]} \boldsymbol{R}^{[k]+} \Delta \boldsymbol{r}^k - (\boldsymbol{I} - \boldsymbol{R}^{[k]} \boldsymbol{R}^{[k]+}) \Delta \boldsymbol{r}^k \tag{75a}$$

$$= \boldsymbol{X}^{[k]} \boldsymbol{R}^{[k]+} \left[\boldsymbol{R}^{[k]} \boldsymbol{R}^{[k]+} \Delta \boldsymbol{r}^k \right] - \boldsymbol{I} \left[(\boldsymbol{I} - \boldsymbol{R}^{[k]} \boldsymbol{R}^{[k]+}) \Delta \boldsymbol{r}^k \right]. \tag{75b}$$

This shows that $\Delta \boldsymbol{r}^k$ is split into a part $\boldsymbol{R}^{[k]} \boldsymbol{R}^{[k]+} \Delta \boldsymbol{r}^k$ in the column span of $\boldsymbol{R}^{[k]}$ and a part $(\boldsymbol{I} - \boldsymbol{R}^{[k]} \boldsymbol{R}^{[k]+}) \Delta \boldsymbol{r}^k$ perpendicular to it. The secant-based approximate Jacobian $\boldsymbol{X}^{[k]} \boldsymbol{R}^{[k]+}$ is applied to the former, while Gauss–Seidel iteration with Jacobian $-\boldsymbol{I}$ is used for the latter.

Filtering When columns of $\boldsymbol{R}^{[k]}$ are linearly dependent up to a tolerance ϵ_f, the diagonal elements of $\boldsymbol{R}_R^{[k]}$ in Eq. (74) become small and this system can no longer be solved accurately. Hence, an essential component of IQN-ILS is filtering, especially when data from previous time steps is reused [50]. Columns of $\boldsymbol{R}^{[k]}$ that are linearly dependent up to the tolerance ϵ_f need to be removed together with the matching columns in $\widetilde{\boldsymbol{X}}^{[k]}$. As the newest information is stored on the left-hand side in $\boldsymbol{R}^{[k]}$, a $\delta \boldsymbol{r}^i$ that is a linear combination of newer $\delta \boldsymbol{r}^j$ $(j > i)$ is removed. Columns can be removed if $\left| R_{R,ii}^{[k]} \right| < \epsilon_f$ (QR0) or $\left| R_{R,ii}^{[k]} \right| < \epsilon_f \left\| \boldsymbol{R}_R^{[k]} \right\|_2$ (QR1), with $R_{R,ii}^{[k]}$ referring to a diagonal element of $\boldsymbol{R}_R^{[k]}$ [55]. The advantage of the first approach is that the tolerance ϵ_f can be set by perturbing $\boldsymbol{x}$ with smaller and smaller changes until the change in $\tilde{\boldsymbol{x}}$ is no longer smooth, but numerical noise. In this case, the tolerance ϵ_f can be considered as a measure of how accurate the flow solver and structural solver are calculating their solution. This filtering procedure is shown step by step in Algorithm 3. Obviously, it will be difficult to obtain convergence of the coupling iterations to a level that is lower than ϵ_f.

Alternative filtering approaches are algebraic QR filtering and POD filtering [55]. In the algebraic filtering method (QR2), a column is removed if the diagonal element $\left| R_{R,ii}^{[k]} \right| < \epsilon_f \left\| \boldsymbol{R}_{R,i}^{[k]} \right\|_2$, with $\boldsymbol{R}_{R,i}^{[k]}$ referring to column i of matrix $\boldsymbol{R}_R^{[k]}$. In the POD filtering, the eigenvalues of the autocorrelation matrix of $\boldsymbol{R}^{[k]}$ are used to truncate old data. The numerical tests in [55] showed that algebraic QR filtering worked better than POD filtering or filtering using $\left| R_{R,ii}^{[k]} \right| < \epsilon_f \left\| \boldsymbol{R}_R^{[k]} \right\|_2$. However, the comparison with $\left| R_{R,ii}^{[k]} \right| < \epsilon_f$ was not performed and remains as an interesting future work. Because the latter is directly related to the solver tolerances themselves, as explained above, it has been chosen for this work.

Another reason to do filtering is to limit the number of secant conditions in cases with few degrees of freedom on the interface. Typically, $m \ll n_x$, but with only few degrees of freedom on the interface, the oldest columns of $\boldsymbol{R}^{[k]}$ and $\widetilde{\boldsymbol{X}}^{[k]}$ need to be removed such that there are at most n_x columns, to avoid an overdetermined Jacobian.

Computational complexity and storage The additional storage required for the IQN-ILS method is the matrices $\boldsymbol{R}^{[k]}$ and $\widetilde{\boldsymbol{X}}^{[k]}$, both $\in \mathbb{R}^{n_x \times m}$. Temporary storage is necessary for $\boldsymbol{Q}_R^{[k]} \in \mathbb{R}^{n_x \times m}$ and $\boldsymbol{R}_R^{[k]} \in \mathbb{R}^{m \times m}$, and the small vector $\boldsymbol{c}^k \in \mathbb{R}^m$. The storage thus scales linearly with the number of degrees of freedom in the interface's discretization. Furthermore, m can be reduced compared to Eq. (72) due to filtering. A rule of thumb is that it is typically not beneficial to include more than 50 columns.

The economy-size QR decomposition of $\boldsymbol{R}^{[k]}$ has at most a complexity of $\mathcal{O}(n_x m^2)$ if the fast Givens method or the Householder method is used [38]. The matrix–vector product in the right-hand side of Eq. (74) has a computational complexity of $\mathcal{O}(n_x m)$ and solving the triangular system a complexity of $\mathcal{O}(m^2)$. Consequently, also the computational complexity scales linearly with n_x and is limited. In numerical tests, the IQN-ILS algorithm normally accounts for less than 1% of the total CPU time.

Algorithm 2 The Interface Quasi-Newton algorithm with an approximation for the Inverse of the Jacobian from a Least-Squares model (IQN-ILS) [50].

choose ϵ_x, ω, q and ϵ_f
$k = 0$
$\tilde{\boldsymbol{x}}^0 = \mathcal{S} \circ \mathcal{F}(\boldsymbol{x}^0)$
$\boldsymbol{r}^0 = \tilde{\boldsymbol{x}}^0 - \boldsymbol{x}^0$
while $\left\|\boldsymbol{r}^k\right\|_2 > \epsilon_x$ **do**
  if ($n = 0$ or $q = 0$) and $k = 0$ **then**
    $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \omega \boldsymbol{r}^k$
  else
    construct $\boldsymbol{R}^{[k]}$ and $\widetilde{\boldsymbol{X}}^{[k]}$ using Eqs. (71)
    calculate QR decomposition $\boldsymbol{R}^{[k]} = \boldsymbol{Q}_R^{[k]} \boldsymbol{R}_R^{[k]}$
    filter $\boldsymbol{R}^{[k]}$ and $\widetilde{\boldsymbol{X}}^{[k]}$ using Algorithm 3
    solve $\boldsymbol{R}_R^{[k]} \boldsymbol{c}^k = -\boldsymbol{Q}_R^{[k]\mathrm{T}} \boldsymbol{r}^k$
    $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \widetilde{\boldsymbol{X}}^{[k]} \boldsymbol{c}^k + \boldsymbol{r}^k$
  end if
  $\tilde{\boldsymbol{x}}^{k+1} = \mathcal{S} \circ \mathcal{F}(\boldsymbol{x}^{k+1})$
  $\boldsymbol{r}^{k+1} = \tilde{\boldsymbol{x}}^{k+1} - \boldsymbol{x}^{k+1}$
  $k = k + 1$
end while

Algorithm 3 A filtering procedure to remove linearly dependent columns and avoid an overdetermined system.

choose ϵ_f
$i = 1$
while $i \leq m$ **do**
  if $\left|\boldsymbol{R}_{R,ii}^{[k]}\right| < \epsilon_f$ **then**
    remove column i in $\boldsymbol{R}^{[k]}$ and $\widetilde{\boldsymbol{X}}^{[k]}$
    calculate QR decomposition $\boldsymbol{R}^{[k]} = \boldsymbol{Q}_R^{[k]} \boldsymbol{R}_R^{[k]}$
    $m = m - 1$
  else
    $i = i + 1$
  end if
end while
if $n_x < m$ **then**
  remove columns $n_x + 1$ to m from $\boldsymbol{R}^{[k]}$ and $\widetilde{\boldsymbol{X}}^{[k]}$
end if

5.2 IBQN-LS

IBQN-LS stands for Interface Block Quasi-Newton method with approximation of the Jacobians using Least-Squares models (initially called reduced-order models) [21]. It uses the formulation in Eq. (25) and solves Eq. (26) and Eq. (27) in turn for $\Delta \boldsymbol{x}^k$ and $\Delta \boldsymbol{y}^k$. Low-rank approximations for $\mathcal{F}'(\boldsymbol{x}^k)$ and $\mathcal{S}'(\boldsymbol{y}^k)$ are constructed using the generalized Broyden method with m as in Eq. (72) and without initial value for the Jacobian, so like in IQN-ILS. The reuse of previous time steps and the filtering are also applied in the same way as explained above. Consequently, this technique enforces all secant conditions from the current and q previous time steps, for both the flow solver and the structural solver.

For the approximate Jacobian of the flow solver $\widehat{\mathcal{F}'}(\boldsymbol{x}^k)$, the generalized Broyden framework is applied using $\boldsymbol{a} = \boldsymbol{x}$ and $\boldsymbol{b} = \tilde{\boldsymbol{y}}$. For m as in Eq. (72) and without initial value for the Jacobian this can be written symbolically as

$$\widehat{\mathcal{F}'}(\boldsymbol{x}^k) \equiv \partial_{\boldsymbol{x}}^k \tilde{\boldsymbol{y}} = \widetilde{\boldsymbol{Y}}^{[k]} \boldsymbol{X}^{[k]+} \tag{76}$$

with

$$\boldsymbol{X}^{[k]} = \left[\boldsymbol{X}^k \; \boldsymbol{X}^n \; \cdots \boldsymbol{X}^{n+1-q}\right] \tag{77a}$$

$$\widetilde{\boldsymbol{Y}}^{[k]} = \left[\widetilde{\boldsymbol{Y}}^k \; \widetilde{\boldsymbol{Y}}^n \; \cdots \; \widetilde{\boldsymbol{Y}}^{n+1-q}\right], \tag{77b}$$

where the secant information from the q previous time steps is combined with that from the current time step

$$\boldsymbol{X}^k = \left[\delta \boldsymbol{x}^{k-1} \; \delta \boldsymbol{x}^{k-2} \; \cdots \; \delta \boldsymbol{x}^0\right] \tag{78a}$$

$$\widetilde{\boldsymbol{Y}}^k = \left[\delta \tilde{\boldsymbol{y}}^{k-1} \; \delta \tilde{\boldsymbol{y}}^{k-2} \; \cdots \; \delta \tilde{\boldsymbol{y}}^0\right]. \tag{78b}$$

A symbolic formulation of the approximation $\widehat{\mathcal{S}'}$ in the generalized Broyden framework can be obtained in a similar way, using $\boldsymbol{a} = \boldsymbol{y}$ and $\boldsymbol{b} = \tilde{\boldsymbol{x}}$.

A disadvantage of this technique is that two linear systems need to be solved in each coupling iteration. In the original version, the $n_x \times n_x$ and $n_y \times n_y$ matrices corresponding with these systems were explicitly constructed using the symbolic notations as in Eq. (76) and they were solved with a direct linear solver [21]. However, by adopting an iterative linear solver like GMRES, only a procedure to calculate the product of the approximate Jacobians with a vector is required [56]. In practice, the number of iterations for the iterative solver is close to the number of columns used for the approximate Jacobians. Alternatively, the Woodbury matrix identity can be used to obtain a closed expression for the update [57].

Matrix-free implementation This matrix-free procedure will be explained here for the flow solver. When the product of $\widehat{\mathcal{F}}'(\boldsymbol{x}^k) \equiv \partial^k_{\boldsymbol{x}} \boldsymbol{y}$ with a vector $\Delta \boldsymbol{x}^k$ needs to be calculated during the iterative solution of Eqs. (26) or (27), this can symbolically be written as

$$\widehat{\mathcal{F}}'(\boldsymbol{x}^k)\, \Delta \boldsymbol{x}^k = \widetilde{\boldsymbol{Y}}^{[k]} \boldsymbol{X}^{[k]+} \Delta \boldsymbol{x}^k \tag{79}$$

For the practical implementation, Algorithm 1 is followed and this computation is split in two parts by the introduction of a coefficient vector $\boldsymbol{c}^k$, giving

$$\widehat{\mathcal{F}}'(\boldsymbol{x}^k)\, \Delta \boldsymbol{x}^k = \widetilde{\boldsymbol{Y}}^{[k]} \boldsymbol{c}^k \tag{80a}$$

with $\boldsymbol{c}^k$ the solution of

$$\boldsymbol{R}_X^{[k]} \boldsymbol{c}^k = \boldsymbol{Q}_X^{[k]\mathrm{T}} \Delta \boldsymbol{x}^k. \tag{80b}$$

The last part is the least-squares solution to an overdetermined system that can be solved efficiently by calculating the economy-size QR decomposition, followed by using back-substitution.

To summarize this procedure, the $\Delta \boldsymbol{x}^k$ is decomposed as a linear combination of the columns in $\boldsymbol{X}^{[k]}$, then the observation is made that columns in $\boldsymbol{X}^{[k]}$ and $\widetilde{\boldsymbol{Y}}^{[k]}$ with the same index form a secant pair, such that the result can be approximated as the same linear combination of the columns in $\widetilde{\boldsymbol{Y}}^{[k]}$, as shown in Eq. (80). The complete procedure can be found in Algorithm 4.

Computational complexity and storage Compared to IQN-ILS, IBQN-LS requires approximately twice the memory, as the data for two approximate Jacobians needs to be stored. Furthermore, even though the matrix-free procedure with the iterative linear solver is faster than explicit matrix construction and direct linear solver, the computing time is higher than for IQN-ILS, where none of this is required. Nevertheless, the time required for the coupling algorithm scales linearly with n_x and n_y and thus remains small compared to that of the actual solvers. The solution of the linear systems could be avoided by writing the solution to Eqs. (26) and (27) symbolically, using matrix inverses and applying the Woodbury matrix identity as in [57]. In this way, the size of the matrices that have to be inverted is reduced from n_x and n_y to m.

Algorithm 4 The Interface Block Quasi-Newton technique with approximations from Least-Squares models (IBQN-LS) [21].

choose ϵ_x, ω, q and ϵ_f
$k = 0$
$\tilde{\boldsymbol{y}}^0 = \mathcal{F}(\boldsymbol{x}^0)$
$\boldsymbol{y}^0 = \tilde{\boldsymbol{y}}^0$
$\tilde{\boldsymbol{x}}^0 = \mathcal{S}(\boldsymbol{y}^0)$
$\boldsymbol{r}^0 = \tilde{\boldsymbol{x}}^0 - \boldsymbol{x}^0$
while $\left\|\boldsymbol{r}^k\right\|_2 > \epsilon_x$ **do**
  if ($n = 0$ or $q = 0$) and $k = 0$ **then**
    $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \omega \boldsymbol{r}^k$
  else
    solve Eq. (26) for $\Delta \boldsymbol{x}^k$
    $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \Delta \boldsymbol{x}^k$
  end if
  $\tilde{\boldsymbol{y}}^{k+1} = \mathcal{F}(\boldsymbol{x}^{k+1})$
  construct $\boldsymbol{X}^{[k+1]}$ and $\widetilde{\boldsymbol{Y}}^{[k+1]}$
  calculate QR decomposition $\boldsymbol{X}^{[k+1]} = \boldsymbol{Q}_X^{[k+1]} \boldsymbol{R}_X^{[k+1]}$
  filter $\boldsymbol{X}^{[k+1]}$ and $\widetilde{\boldsymbol{Y}}^{[k+1]}$ as in Algorithm 3
  if ($n = 0$ or $q = 0$) and $k = 0$ **then**
    $\boldsymbol{y}^{k+1} = \tilde{\boldsymbol{y}}^{k+1}$
  else
    solve Eq. (27) for $\Delta \boldsymbol{y}^k$
    $\boldsymbol{y}^{k+1} = \boldsymbol{y}^k + \Delta \boldsymbol{y}^k$
  end if
  $\tilde{\boldsymbol{x}}^{k+1} = \mathcal{S}(\boldsymbol{y}^{k+1})$
  construct $\boldsymbol{Y}^{[k+1]}$ and $\widetilde{\boldsymbol{X}}^{[k+1]}$
  calculate QR decomposition $\boldsymbol{Y}^{[k+1]} = \boldsymbol{Q}_Y^{[k+1]} \boldsymbol{R}_Y^{[k+1]}$
  filter $\boldsymbol{Y}^{[k+1]}$ and $\widetilde{\boldsymbol{X}}^{[k+1]}$ as in Algorithm 3
  $\boldsymbol{r}^{k+1} = \tilde{\boldsymbol{x}}^{k+1} - \boldsymbol{x}^{k+1}$
  $k = k + 1$
end while

5.3 MVQN

MVQN is the abbreviation for Multi-Vector update Quasi-Newton [58]. This method is based on the IBQN-LS method and is even identical to it in the first time step. However, the differences appear when data from previous time steps is included. MVQN considers the current Jacobian as the sum of the Jacobian from the previous time step plus a rank-k update. This update is then determined by enforcing the secant conditions from the current time step and minimizing the Frobenius norm of the update. This coincides with the generalized Broyden method with $m = k^{n+1}$ and the initial Jacobian equal to the one from the previous time step.

Considering again the approximate Jacobian of the flow solver $\widehat{\mathcal{F}'}$, the generalized Broyden framework is applied using $\boldsymbol{a} = \boldsymbol{x}$ and $\boldsymbol{b} = \tilde{\boldsymbol{y}}$. The value m is now $k = k^{n+1}$ and the initial value for the Jacobian is the one for the previous time step, giving

$$\widehat{\mathcal{F}'}(\boldsymbol{x}^k) \equiv \partial_{\boldsymbol{x}}^k \tilde{\boldsymbol{y}} = \widetilde{\boldsymbol{Y}}^k \boldsymbol{X}^{k+} + \partial_{\boldsymbol{x}}^n \tilde{\boldsymbol{y}} \left(\boldsymbol{I} - \boldsymbol{X}^k \boldsymbol{X}^{k+} \right). \tag{81}$$

Using the definition of the pseudo-inverse in Eq. (36), this can be reformulated as

$$\widehat{\mathcal{F}'}(\boldsymbol{x}^k) = \partial_{\boldsymbol{x}}^n \tilde{\boldsymbol{y}} + \left(\widetilde{\boldsymbol{Y}}^k - \partial_{\boldsymbol{x}}^n \tilde{\boldsymbol{y}} \boldsymbol{X}^k \right) \left(\boldsymbol{X}^{k\mathrm{T}} \boldsymbol{X}^k \right)^{-1} \boldsymbol{X}^{k\mathrm{T}}, \tag{82}$$

which corresponds with the original formulation in [58]. However, analyzing this method is most straightforward when considering Eq. (81). If this approximate Jacobian is multiplied with a vector $\Delta\boldsymbol{x}$, then the secant information from the most recent time step is used for the part for which it is available, i.e., within the column span of the matrix $\boldsymbol{X}^k$. The previous approximate Jacobian is only multiplied with the leftover part of $\Delta\boldsymbol{x}$, i.e., the part orthogonal to the column span of $\boldsymbol{X}^k$. So, even if secant information from previous time steps is available for the first part of $\Delta\boldsymbol{x}$, it will not be used. This relates to the difference between the least-squares and multi-vector approach explained in Sect. 4.4. The matrix $\widehat{\mathcal{S}'}$ is constructed in a similar way

$$\widehat{\mathcal{S}'}(\boldsymbol{y}^k) = \partial_{\boldsymbol{y}}^n \tilde{\boldsymbol{x}} + \left(\widetilde{\boldsymbol{X}}^k - \partial_{\boldsymbol{y}}^n \tilde{\boldsymbol{x}} \boldsymbol{Y}^k \right) \left(\boldsymbol{Y}^{k\mathrm{T}} \boldsymbol{Y}^k \right)^{-1} \boldsymbol{Y}^{k\mathrm{T}}. \tag{83}$$

Computational complexity and storage The main benefits of this method are that no parameter q is required and that filtering with a tolerance ϵ_f is typically less essential as only a relatively small number of secant conditions from the most recent time step are considered. However, this comes at a significant cost, as the matrices $\widehat{\mathcal{F}'}$ and $\widehat{\mathcal{S}'}$ are constructed and stored in memory, such that the algorithm scales with n_x^2 and n_y^2, both in terms of memory use as in computational complexity. Combined with the linear systems that have to be solved in each coupling iteration, this becomes expensive compared to the actual solver for a reasonably large number of degrees of freedom on the interface (e.g., more than 10^4). A linearly scaling adaptation is the RandomiZed Multi-Vector Quasi-Newton method (MVQN-RZ) [57], which will be explained in more detail with the other linearly scaling multi-vector methods at the end of Sect. 5.4.

Algorithm 5 The Multi-Vector update Quasi-Newton method (MVQN) [58].

choose ϵ_x and ω
$k = 0$
$\tilde{\boldsymbol{y}}^0 = \mathcal{F}(\boldsymbol{x}^0)$
$\boldsymbol{y}^0 = \tilde{\boldsymbol{y}}^0$
$\tilde{\boldsymbol{x}}^0 = \mathcal{S}(\boldsymbol{y}^0)$
$\boldsymbol{r}^0 = \tilde{\boldsymbol{x}}^0 - \boldsymbol{x}^0$
while $\left\|\boldsymbol{r}^k\right\|_2 > \epsilon_x$ **do**
  if ($n = 0$ or $q = 0$) and $k = 0$ **then**
    $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \omega \boldsymbol{r}^k$
  else
    solve Eq. (26) for $\Delta\boldsymbol{x}^k$
    $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \Delta\boldsymbol{x}^k$
  end if
  $\tilde{\boldsymbol{y}}^{k+1} = \mathcal{F}(\boldsymbol{x}^{k+1})$
  construct $\boldsymbol{X}^{k+1}$ and $\widetilde{\boldsymbol{Y}}^{k+1}$
  construct $\widehat{\mathcal{F}'}(\boldsymbol{x}^{k+1}) \equiv \partial_{\boldsymbol{x}}^{k+1} \tilde{\boldsymbol{y}}$ using Eq. (82)
  if $n = 0$ and $k = 0$ **then**
    $\boldsymbol{y}^{k+1} = \tilde{\boldsymbol{y}}^{k+1}$
  else
    solve Eq. (27) for $\Delta\boldsymbol{y}^k$
    $\boldsymbol{y}^{k+1} = \boldsymbol{y}^k + \Delta\boldsymbol{y}^k$
  end if
  $\tilde{\boldsymbol{x}}^{k+1} = \mathcal{S}(\boldsymbol{y}^{k+1})$
  construct $\boldsymbol{Y}^{k+1}$ and $\widetilde{\boldsymbol{X}}^{k+1}$
  construct $\widehat{\mathcal{S}'}(\boldsymbol{y}^{k+1}) \equiv \partial_{\boldsymbol{y}}^{k+1} \tilde{\boldsymbol{x}}$ using Eq. (83)
  $\boldsymbol{r}^{k+1} = \tilde{\boldsymbol{x}}^{k+1} - \boldsymbol{x}^{k+1}$
  $k = k + 1$
end while

5.4 IQN-MVJ

The IQN-MVJ method is an acronym for Interface Quasi-Newton with Multi-Vector Jacobian [59]. This method adopts the idea for reuse from previous time steps proposed in MVQN and transfers it from the block iteration to the residual formulation quasi-Newton scheme, i.e., with only the interface displacement as variable. It is thus a generalized Broyden method which satisfies the secant conditions from the current time step while using the Jacobian from the previous time step as initial value. IQN-MVJ is linked with IQN-ILS in the same way as MVQN is linked with IBQN-LS. Except for the explicit construction of the approximate Jacobian, this method is identical to IQN-ILS in the first time step.

In IQN-MVJ, the approximation for the inverse of the Jacobian[5] is thus constructed as

$$\widehat{\mathcal{R}'^{-1}}(\boldsymbol{x}^k) \equiv \partial_{\boldsymbol{r}}^{n+1,k}\tilde{\boldsymbol{x}} - \boldsymbol{I} \tag{84}$$

with

$$\partial_{\boldsymbol{r}}^{n+1,k}\tilde{\boldsymbol{x}} = \widetilde{\boldsymbol{X}}^k \boldsymbol{R}^{k+} + \partial_{\boldsymbol{r}}^{n}\tilde{\boldsymbol{x}}\left(\boldsymbol{I} - \boldsymbol{R}^k \boldsymbol{R}^{k+}\right). \tag{85}$$

Using the definition of the pseudo-inverse in Eq. (36), this can be reformulated as

$$\partial_{\boldsymbol{r}}^{n+1,k}\tilde{\boldsymbol{x}} = \partial_{\boldsymbol{r}}^{n}\tilde{\boldsymbol{x}} + \left(\widetilde{\boldsymbol{X}}^k - \partial_{\boldsymbol{r}}^{n}\tilde{\boldsymbol{x}}\,\boldsymbol{R}^k\right)\left(\boldsymbol{R}^{k\mathrm{T}}\boldsymbol{R}^k\right)^{-1}\boldsymbol{R}^{k\mathrm{T}}, \tag{86}$$

which corresponds with the original formulation in [59].

Computational complexity and storage The main drawback of IQN-MVJ is that the square matrix with the approximation for the inverse of the Jacobian is constructed and stored in memory, such that computational cost and memory requirement scale with n_x^2. Consequently, like MVQN, this approach becomes expensive compared to the solvers for a reasonably large number of degrees of freedom on the interface (e.g., more than 10^4).

To avoid this scaling and achieve linear complexity in n_x and at the same time attempt to avoid a reuse parameter q, a matrix-free version of IQN-MVJ has been developed, named IQN-MVJ-RS-SVD [60]. Thereto, the approximation in Eq. (86) is reformulated as

$$\partial_{\boldsymbol{r}}^{n+1,k}\tilde{\boldsymbol{x}} = \partial_{\boldsymbol{r}}^{n}\tilde{\boldsymbol{x}} + \tilde{\widetilde{\boldsymbol{X}}}^k \boldsymbol{R}^{k+}, \tag{87}$$

with

$$\tilde{\widetilde{\boldsymbol{X}}}^k = \widetilde{\boldsymbol{X}}^k - \partial_{\boldsymbol{r}}^{n}\tilde{\boldsymbol{x}}\,\boldsymbol{R}^k. \tag{88}$$

To avoid storage of a square matrix, this can be written using a recursive formula

$$\partial_{\boldsymbol{r}}^{n+1,k}\tilde{\boldsymbol{x}} = \tilde{\widetilde{\boldsymbol{X}}}^k \boldsymbol{R}^{k+} + \tilde{\widetilde{\boldsymbol{X}}}^n \boldsymbol{R}^{n+} + \ldots + \tilde{\widetilde{\boldsymbol{X}}}^1 \boldsymbol{R}^{1+} \tag{89}$$

starting from $\partial_{\boldsymbol{r}}^{0}\tilde{\boldsymbol{x}} = \boldsymbol{0}$ at time 0. Obviously, this requires storage of matrices $\tilde{\widetilde{\boldsymbol{X}}}$ and $\boldsymbol{R}$ for each time step, which is beneficial as long as the total number of columns is significantly smaller than n_x. However, after a high number of time steps, this becomes prohibitively expensive and therefore three strategies are proposed in which the simulation is split into so-called chunks consisting of q' time steps after which a restart is performed [60]. The authors concluded that the IQN-MVJ-RS-SVD algorithm with a Singular Value Decomposition (SVD) restart strategy is the most promising. The approximate inverse Jacobian at the end of a chunk is then truncated by performing an SVD and truncating the singular values below a tolerance ϵ_f'. This truncated SVD is then the initial Jacobian for the following chunk. Furthermore, the SVD is efficiently updated at the end of each chunk. As opposed to the original IQN-MVJ, the IQN-MVJ-RS-SVD method has linear complexity in n_x, but the implementation is more elaborate than for IQN-MVJ. Compared to IQN-ILS, which has a reuse parameter q and QR filter tolerance ϵ_f and also has linear scaling in n_x, this method now requires a chunk size parameter q' and SVD filter tolerance ϵ_f', but with the claim that the performance is less sensitive to these parameters.

Another multi-vector method that achieves linear scaling memory requirements is the algorithm MVQN-RZ [57], which employs the randomized SVD not only to avoid the explicit construction of large square matrices, but also to circumvent the recursive reconstruction of the interface Jacobian. This algorithm has been applied to both block iteration quasi-Newton techniques and residual formulation quasi-Newton techniques. In every coupling iteration, the complexity of MVQN-RZ is $\mathcal{O}(z^2 n_x + k^2 n_x)$, where z is the number of decomposition modes. This compares to IQN-MVJ-RS-SVD with a complexity of $\mathcal{O}(z^2 n_x + k^2 n_x + Mkn_x + k^4)$ in every coupling iteration, with z referring to the number of eigenvalues left after truncation and M the simulation chunk size. Additionally, this method requires an SVD update after every M steps, which has a complexity of $\mathcal{O}(Mz^2 n_x)$. In providing these complexities, it is assumed that $n_x \gg k, z, M$.

[5] Some authors rewrite the update $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \widehat{\mathcal{R}'^{-1}}(\boldsymbol{x}^k)\,\boldsymbol{r}^k$ as $\boldsymbol{x}^{k+1} = \tilde{\boldsymbol{x}}^k - \partial_{\boldsymbol{r}}^{k}\tilde{\boldsymbol{x}}\,\boldsymbol{r}^k$, since $\widehat{\mathcal{R}'^{-1}}(\boldsymbol{x}^k) \equiv \partial_{\boldsymbol{r}}^{k}\tilde{\boldsymbol{x}} - \boldsymbol{I}$, and then $\partial_{\boldsymbol{r}}^{k}\tilde{\boldsymbol{x}}$ is called the approximation of the inverse Jacobian, but this notation is not used here to avoid confusion.

Algorithm 6 The Interface Quasi-Newton Multi-Vector Jacobian method (IQN-MVJ) [59].

choose ϵ_x and ω
$k = 0$
$\tilde{\boldsymbol{x}}^0 = \mathcal{S} \circ \mathcal{F}(\boldsymbol{x}^0)$
$\boldsymbol{r}^0 = \tilde{\boldsymbol{x}}^0 - \boldsymbol{x}^0$
while $\left\|\boldsymbol{r}^k\right\|_2 > \epsilon_x$ **do**
  if $n = 0$ and $k = 0$ **then**
    $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \omega \boldsymbol{r}^k$
  else
    construct $\boldsymbol{R}^k$ and $\widetilde{\boldsymbol{X}}^k$ as in Eqs. (70)
    construct $\widehat{\mathcal{R}'^{-1}}(\boldsymbol{x}^k)$ using Eq. (84) and Eq. (85)
    $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \widehat{\mathcal{R}'^{-1}}(\boldsymbol{x}^k)\, \boldsymbol{r}^k$
  end if
  $\tilde{\boldsymbol{x}}^{k+1} = \mathcal{S} \circ \mathcal{F}(\boldsymbol{x}^{k+1})$
  $\boldsymbol{r}^{k+1} = \tilde{\boldsymbol{x}}^{k+1} - \boldsymbol{x}^{k+1}$
  $k = k + 1$
end while

5.5 IQN-IMVLS

To mitigate the quadratic scaling in n_x of IQN-MVJ, the IQN-IMVLS (Interface Quasi-Newton Implicit Multi-Vector Least-Squares) method has been developed with linear complexity in n_x [53]. The first observation is that the factor $\widetilde{\boldsymbol{X}}^k - \partial_{\boldsymbol{r}}^n \tilde{\boldsymbol{x}}\, \boldsymbol{R}^k$ in Eq. (86) can be updated by adding one additional column in each coupling iteration, instead of recomputing it entirely

$$\widetilde{\boldsymbol{X}}^k - \partial_{\boldsymbol{r}}^n \tilde{\boldsymbol{x}}\, \boldsymbol{R}^k = [\delta\tilde{\boldsymbol{x}}^{k-1} - \partial_{\boldsymbol{r}}^n \tilde{\boldsymbol{x}}\, \delta\boldsymbol{r}^{k-1}, \widetilde{\boldsymbol{X}}^{k-1} - \partial_{\boldsymbol{r}}^n \tilde{\boldsymbol{x}}\, \boldsymbol{R}^{k-1}], \quad (90)$$

so only the product $\partial_{\boldsymbol{r}}^n \tilde{\boldsymbol{x}}\, \delta\boldsymbol{r}^{k-1}$ needs to be evaluated.

Determining the product of $\partial_{\boldsymbol{r}}^n \tilde{\boldsymbol{x}}$ with a vector requires a procedure to calculate a matrix–vector product. It is proven in [53] that $\partial_{\boldsymbol{r}}^n \tilde{\boldsymbol{x}}$ in Eq. (86) can be reformulated as

$$\partial_{\boldsymbol{r}}^n \tilde{\boldsymbol{x}} = \sum_{i=1}^{n} \widetilde{\boldsymbol{X}}^i \boldsymbol{R}^{i^+} \prod_{j=i+1}^{n} (\boldsymbol{I} - \boldsymbol{R}^j \boldsymbol{R}^{j^+}) \quad (91)$$

if the initial Jacobian is assumed to be zero. This recursive formulation can be truncated after q terms, giving

$$\partial_{\boldsymbol{r}}^n \tilde{\boldsymbol{x}} \approx \sum_{i=n+1-q}^{n} \widetilde{\boldsymbol{X}}^i \boldsymbol{R}^{i^+} \prod_{j=i+1}^{n} (\boldsymbol{I} - \boldsymbol{R}^j \boldsymbol{R}^{j^+}). \quad (92)$$

In IQN-IMVLS the reuse parameter q typically does not exhibit an optimum in terms of performance. Instead, both performance and computational cost grow with increasing values of q as the method converges towards the IQN-MVJ approach.

Computational complexity and storage As can be observed in Eq. (92), this procedure requires the storage of $\boldsymbol{R}^{j^+} = \left(\boldsymbol{R}^{j^{\mathrm{T}}} \boldsymbol{R}^j\right)^{-1} \boldsymbol{R}^{j^{\mathrm{T}}}$ for the q previous time steps. If the inverse of $\boldsymbol{R}^{j^{\mathrm{T}}} \boldsymbol{R}^j$ is calculated via the LU decomposition using partial pivoting with row interchanges [38], then this scales with n_x, but has slightly less robustness to bad conditioning than the Householder QR approach. As a result, the complete procedure has linear complexity in n_x, like IQN-ILS. In addition, the QR decomposition is only applied on the secant information from the most recent time step, as opposed to the matrix with the secant information from all time steps in IQN-ILS. As not all secant information is combined into one matrix, the sensitivity to (almost) linear dependencies is smaller. Furthermore, no restart is required, but the implementation is a bit more involved than IQN-ILS or IQN-MVJ. It was also observed in [53] that including the secant information from the previous time step in $\widetilde{\boldsymbol{X}}$ and $\boldsymbol{R}$ as well can accelerate the convergence, especially at the beginning of a time step. The complete procedure can be found in Algorithm 7.

Algorithm 7 The Interface Quasi-Newton Implicit Multi-Vector Least-Squares method (IQN-IMVLS) [53].

choose ϵ_x, ω and q
$k = 0$
$\tilde{\boldsymbol{x}}^0 = \mathcal{S} \circ \mathcal{F}(\boldsymbol{x}^0)$
$\boldsymbol{r}^0 = \tilde{\boldsymbol{x}}^0 - \boldsymbol{x}^0$
while $\left\|\boldsymbol{r}^k\right\|_2 > \epsilon_x$ **do**
  if $n = 0$ and $k = 0$ **then**
    $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \omega \boldsymbol{r}^k$
  else if $k = 0$ **then**
    calculate $\partial_{\boldsymbol{r}}^n \tilde{\boldsymbol{x}}\, \boldsymbol{r}^k$ using Eq. (92)
    $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \partial_{\boldsymbol{r}}^n \tilde{\boldsymbol{x}}\, \boldsymbol{r}^k + \boldsymbol{r}^k$
  else
    construct $\boldsymbol{R}^k$ and $\widetilde{\boldsymbol{X}}^k$ as in Eqs. (70)
    calculate $\partial_{\boldsymbol{r}}^n \tilde{\boldsymbol{x}}\, \boldsymbol{r}^k$ using Eq. (92)
    add column to $\widetilde{\boldsymbol{X}}^k - \partial_{\boldsymbol{r}}^n \tilde{\boldsymbol{x}}\, \boldsymbol{R}^k$ as in Eq. (90)
    solve $\boldsymbol{R}_R^k \boldsymbol{c}^k = -\boldsymbol{Q}_R^{k\,\mathrm{T}} \boldsymbol{r}^k$
    $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \partial_{\boldsymbol{r}}^n \tilde{\boldsymbol{x}}\, \boldsymbol{r}^k + \left(\widetilde{\boldsymbol{X}}^k - \partial_{\boldsymbol{r}}^n \tilde{\boldsymbol{x}}\, \boldsymbol{R}^k\right) \boldsymbol{c}^k + \boldsymbol{r}^k$
  end if
  $\tilde{\boldsymbol{x}}^{k+1} = \mathcal{S} \circ \mathcal{F}(\boldsymbol{x}^{k+1})$
  $\boldsymbol{r}^{k+1} = \tilde{\boldsymbol{x}}^{k+1} - \boldsymbol{x}^{k+1}$
  $k = k + 1$
end while
calculate $\boldsymbol{R}^{k+}$ using LU decomposition

5.6 IQN-ILSM

All techniques mentioned above use the flow solver and structural solver as black boxes, while sometimes the user has additional insight into the behaviour of the problem. This additional information can be incorporated to accelerate the convergence of the coupling iterations using the Interface Quasi-Newton algorithm with an approximation for the Inverse of the Jacobian from a Least-Squares model and additional Surrogate Model method (IQN-ILSM) [61]. In this technique, the secant information is combined with a so-called surrogate model, which behaves similarly to the actual solvers but is significantly faster. The origins of this technique can be found in the FreQ-LeSS algorithm for free-surface calculation, where secant conditions from previous flow solver iterations was combined with an analytical model of the problem [62, 63].

The surrogate model is denoted as $\mathcal{R}_s$ and the subscript s will be used to denote quantities related to the surrogate model. The inverse Jacobian of $\mathcal{R}_s$ with respect to $\tilde{\boldsymbol{x}}$ is referred to as the surrogate Jacobian, which is assumed to stay the same during the entire time step. A procedure to calculate the product of this matrix with a vector is sufficient, without requiring construction and storage. To emphasize this matrix-free aspect, the surrogate Jacobian is represented by a function $\partial_{\boldsymbol{r}} \tilde{\boldsymbol{x}}_s(\cdot)$. Furthermore, this surrogate Jacobian can be either full-rank or only a low-rank approximation, with column space $\boldsymbol{R}_s$. At the end of each time step, the surrogate model is synchronized with the original model by interpolating the solution from the latter to the former. This approach avoids large discrepancies between the original model and the surrogate model after some time steps.

To gain insight in the IQN-ILSM technique, Eq. (75b) should first be revisited. This equation shows how IQN-ILS splits $\Delta\boldsymbol{r}^k$ in a part $\boldsymbol{R}^k \boldsymbol{R}^{k+} \Delta\boldsymbol{r}^k$ in the column span of $\boldsymbol{R}^k$ and a part $(\boldsymbol{I} - \boldsymbol{R}^k \boldsymbol{R}^{k+}) \Delta\boldsymbol{r}^k$ orthogonal to it, using secant conditions for the former and Gauss–Seidel iteration for the latter. The term $(\boldsymbol{I} - \boldsymbol{R}^k \boldsymbol{R}^{k+}) \Delta\boldsymbol{r}^k$ for which no secant information is known can now be split once more into a part $\boldsymbol{R}_s \boldsymbol{R}_s^{+} \left(\boldsymbol{I} - \boldsymbol{R}^k \boldsymbol{R}^{k+}\right) \Delta\boldsymbol{r}^k$ for which the surrogate model has information and the remainder $\left(\boldsymbol{I} - \boldsymbol{R}_s \boldsymbol{R}_s^{+}\right) \left(\boldsymbol{I} - \boldsymbol{R}^k \boldsymbol{R}^{k+}\right) \Delta\boldsymbol{r}^k$ for which Gauss–Seidel iteration is the best option. Obviously, the latter is zero if the surrogate Jacobian is full-rank. The split of $\Delta\boldsymbol{r}^k$ and the approximate Jacobian used for each part can be written as

$$\begin{aligned} \Delta\boldsymbol{x}^k = {} & \boldsymbol{X}^k \boldsymbol{R}^{k+} \left[\boldsymbol{R}^k \boldsymbol{R}^{k+} \Delta\boldsymbol{r}^k\right] \\ & + \partial_{\boldsymbol{r}} \boldsymbol{x}_s \left(\left[\boldsymbol{R}_s \boldsymbol{R}_s^{+} \left(\boldsymbol{I} - \boldsymbol{R}^k \boldsymbol{R}^{k+}\right) \Delta\boldsymbol{r}^k\right] \right) \\ & - \boldsymbol{I} \left[\left(\boldsymbol{I} - \boldsymbol{R}_s \boldsymbol{R}_s^{+}\right) \left(\boldsymbol{I} - \boldsymbol{R}^k \boldsymbol{R}^{k+}\right) \Delta\boldsymbol{r}^k \right]. \end{aligned} \tag{93}$$

Note that this equation is written in terms of $\boldsymbol{X}^k$ rather than $\widetilde{\boldsymbol{X}}^k$ and that $\partial_{\boldsymbol{r}} \boldsymbol{x}_s(\cdot)$ is used rather than $\partial_{\boldsymbol{r}} \tilde{\boldsymbol{x}}_s(\cdot)$. Here $\partial_{\boldsymbol{r}} \boldsymbol{x}_s$ refers to $\partial_{\boldsymbol{r}} \tilde{\boldsymbol{x}}_s - \boldsymbol{R}_s \boldsymbol{R}_s^{+}$ and not to the full-rank Jacobian $\partial_{\boldsymbol{r}} \tilde{\boldsymbol{x}}_s - \boldsymbol{I}$ as was the case in Eq. (68). Equation (93) shows that the secant conditions have the highest priority, followed by the surrogate Jacobian and then Gauss–Seidel iteration. In the first coupling iteration, there are no secant conditions yet, so the

surrogate Jacobian plays an important role. The contribution of the secant conditions will then become more significant during the coupling iterations as the column span of $\boldsymbol{R}^k$ gradually increases. The expression in Eq. (93) can be simplified by using Jacobians with respect to $\tilde{\boldsymbol{x}}$ instead, giving

$$\Delta \boldsymbol{x}^k = \widetilde{\boldsymbol{X}}^k \boldsymbol{R}^{k^+} \Delta \boldsymbol{r}^k + \partial_{\boldsymbol{r}} \tilde{\boldsymbol{x}}_s \left(\left(\boldsymbol{I} - \boldsymbol{R}^k \boldsymbol{R}^{k^+} \right) \Delta \boldsymbol{r}^k \right) - \Delta \boldsymbol{r}^k. \tag{94}$$

These expressions can be expanded by including multiple surrogate models in a similar way.

Several types of surrogate models can be considered. A first type is a coarse grid version of the original problem, using the same solvers. When using such a surrogate model, it is important to note that the secant information obtained on the original grid and on the coarse grid is not combined. The surrogate Jacobian thus also uses the coarse grid and interpolation is performed when it needs to be multiplied with a vector from the original grid. A second option is to use solvers with simplified physics, such as solvers neglecting viscosity or nonlinearity. If the simplified physics solvers have a known Jacobian, e.g., because they are analytical functions, then the surrogate Jacobian will typically be constructed and stored but it will be full-rank and only a relatively small matrix.

Another option for the surrogate is reuse from previous time steps in a time-dependent simulation. As opposed to the previously mentioned surrogate types, this surrogate model does not require the solution of a separate problem and no synchronization at the end of each time step, so it is essentially free. The reuse of q previous time steps corresponds to q nested surrogate models, with decreasing importance when for older time steps, giving

$$\partial_{\boldsymbol{r}}^{n+1,k} \boldsymbol{x} = \widetilde{\boldsymbol{X}}^k \boldsymbol{R}^{k^+} + \sum_{i=n+1-q}^{n} \widetilde{\boldsymbol{X}}^i \boldsymbol{R}^{i^+} \left(\prod_{j=i+1}^{n} (\boldsymbol{I} - \boldsymbol{R}^j \boldsymbol{R}^{j^+}) \right) (\boldsymbol{I} - \boldsymbol{R}^k \boldsymbol{R}^{k^+}) - \boldsymbol{I} \tag{95}$$

with $\widetilde{\boldsymbol{X}}^i$ and $\boldsymbol{R}^i$ containing the secant information from time step i. This equation can be condensed to

$$\partial_{\boldsymbol{r}}^{n+1,k} \boldsymbol{x} = \sum_{i=n+1-q}^{n+1} \widetilde{\boldsymbol{X}}^i \boldsymbol{R}^{i^+} \prod_{j=i+1}^{n+1} \left(\boldsymbol{I} - \boldsymbol{R}^j \boldsymbol{R}^{j^+} \right) - \boldsymbol{I}. \tag{96}$$

When calculating the product $\prod_{i+1}^{n+1} (\boldsymbol{I} - \boldsymbol{R}^j \boldsymbol{R}^{j^+}) \Delta \boldsymbol{r}^k$ for each of the terms in the summation, the value in the previous term is stored and updated with one factor for the next term. Therefore the summation is in fact done in reverse order ($n+1 \rightarrow n+1-q$). The IQN-ILSM algorithm can be found in Algorithm 8 and an efficient calculation of $\Delta \boldsymbol{x}_s = \partial_{\boldsymbol{r}} \tilde{\boldsymbol{x}}_s(\cdot)$ in the case of reuse from previous time steps in Algorithm 9.

Computational complexity and storage While in IQN-ILS the QR decomposition is applied on the secant information from all time steps combined in each coupling iteration, IQN-ILSM with reuse as surrogate only applies the QR decomposition on the data from the current time step during the coupling iterations. The QR decomposition of the data from previous time steps is stored and does not need to be updated. This difference is reflected in the computational cost, which scales as $\mathcal{O}(n_x (q\bar{k})^2)$ for IQN-ILS and $\mathcal{O}(n_x \bar{k}^2)$ for IQN-ILSM, with $\bar{k}$ the number of coupling iterations averaged over the time steps. Moreover, as IQN-ILSM does not combine the secant conditions from all time steps in a single matrix, it typically does not require filtering, as opposed to IQN-ILS.

By comparing Eq. (96) with Eq. (92), it can be observed that the IQN-ILSM method with reuse as surrogate is identical to the IQN-IMVLS method, except for some implementation aspects mentioned in Sect. 5.5. IQN-ILSM can thus be considered as a generalization of IQN-IMVLS such that not only reuse of secant information from previous time steps, but also physics-based surrogate models can be used, although it was not developed as such. Furthermore, the IQN-ILSM method can be interpreted as part of a larger class of methods which combine data-driven relations with physics-based knowledge [64].

In addition to providing a surrogate Jacobian to accelerate the convergence of the coupling iterations, the surrogate can also provide an initial guess for the coupling iterations. This prediction can be found in line 5 of Algorithm 8. When reuse from previous time steps is the surrogate model, this corresponds to linear extrapolation.

Algorithm 8 The Interface Quasi-Newton algorithm with an approximation for the Inverse of the Jacobian from a Least-Squares model and additional Surrogate Model (IQN-ILSM) [61].

Ensure: given surrogate Jacobian $\partial_{\boldsymbol{r}} \tilde{\boldsymbol{x}}_s(\cdot)$
1: choose ϵ_x and ω
2: $k = 0$
3: solve $\mathcal{R}_s(\boldsymbol{x}_s) = 0$
4: determine $\partial_{\boldsymbol{r}} \tilde{\boldsymbol{x}}_s(\cdot)$
5: $\boldsymbol{x}^0 = \boldsymbol{x}^n + (\boldsymbol{x}_s - \boldsymbol{x}_s^n)$
6: $\tilde{\boldsymbol{x}}^0 = \mathcal{S} \circ \mathcal{F}(\boldsymbol{x}^0)$
7: $\boldsymbol{r}^0 = \tilde{\boldsymbol{x}}^0 - \boldsymbol{x}^0$
8: **while** $\left\|\boldsymbol{r}^k\right\|_2 > \epsilon_x$ **do**
9: **if** $k = 0$ **then**
10: $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \partial_{\boldsymbol{r}} \tilde{\boldsymbol{x}}_s(-\boldsymbol{r}^k) + \omega \boldsymbol{r}^k$
11: **else**
12: construct $\boldsymbol{R}^k$ and $\widetilde{\boldsymbol{X}}^k$
13: calculate QR decomposition $\boldsymbol{R}^k = \boldsymbol{Q}_R^k \boldsymbol{R}_R^k$
14: $\boldsymbol{d}^k = {\boldsymbol{Q}_R^k}^{\mathrm{T}} \boldsymbol{r}^k$
15: solve $\boldsymbol{R}_R^k \boldsymbol{c}^k = -\boldsymbol{d}^k$
16: $\boldsymbol{e}^k = \boldsymbol{r}^k - \boldsymbol{Q}_R^k \boldsymbol{d}^k$
17: $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \widetilde{\boldsymbol{X}}^k \boldsymbol{c}^k + \partial_{\boldsymbol{r}} \tilde{\boldsymbol{x}}_s(-\boldsymbol{e}^k) + \boldsymbol{r}^k$
18: **end if**
19: $\tilde{\boldsymbol{x}}^{k+1} = \mathcal{S} \circ \mathcal{F}(\boldsymbol{x}^{k+1})$
20: $\boldsymbol{r}^{k+1} = \tilde{\boldsymbol{x}}^{k+1} - \boldsymbol{x}^{k+1}$
21: $k = k + 1$
22: **end while**
23: synchronize $\mathcal{R}_s$ with $\mathcal{R}$

Algorithm 9 An efficient procedure to calculate the product of $\Delta \boldsymbol{x}_s = \partial_{\boldsymbol{r}} \tilde{\boldsymbol{x}}_s(-\boldsymbol{e}^k)$ on line 17 in the IQN-ILSM technique with reuse from previous time steps as surrogate [61].

Ensure: given $\widetilde{\boldsymbol{X}}^i, \boldsymbol{Q}_R^i, \boldsymbol{R}_R^i$ for time step $i = n, \ldots, n+1-q$; given $\boldsymbol{e}^k = \boldsymbol{r}^k - \boldsymbol{Q}_R^k \boldsymbol{d}^k$
1: set $\Delta \boldsymbol{x}_s = \boldsymbol{0}$
2: **for** $i = n, \ldots, n+1-q$ **do**
3: $\boldsymbol{d}^i = {\boldsymbol{Q}_R^i}^{\mathrm{T}} \boldsymbol{e}^k$
4: solve $\boldsymbol{R}_R^i \boldsymbol{c}^i = -\boldsymbol{d}^i$
5: $\Delta \boldsymbol{x}_s = \Delta \boldsymbol{x}_s + \widetilde{\boldsymbol{X}}^i \boldsymbol{c}^i$
6: $\boldsymbol{e}^k = \boldsymbol{e}^k - \boldsymbol{Q}_R^i \boldsymbol{d}^i$
7: **end for**

5.7 Other Algorithms

Besides the already mentioned techniques, there are many other variants, a selection of which is touched upon below.[6] This section ends with an outlook to the application of quasi-Newton coupling techniques outside of the field of fluid–structure interaction.

Jacobi iteration and high-performance computing The algorithms mentioned above are all using a sequential solution of the flow problem and structural problem. So, they can be considered related to Gauss–Seidel iteration, as opposed to Jacobi iteration which is characterized by simultaneous solution of both problems. For several of the quasi-Newton coupling techniques mentioned in this review, similar techniques are available based on Jacobi iteration [65]. For linear systems, Gauss–Seidel iteration inherently converge faster than Jacobi iteration, and this typically also holds for nonlinear systems, such as the FSI problem [66]. However, the higher number of iterations in a Jacobi-based method may be compensated by its better parallel scalability, although this is not guaranteed. Therefore, the difference in parallel scalability between both iterations types will be explained in the following lines. For Gauss–Seidel iteration, the flow and structure solvers are run sequentially, so both solvers can use all available CPU cores. However, the work load and the parallel efficiency of both solvers is typically very different, and often much higher for the flow solver. If the structure solver does not scale up well to a large number of cores, this leads

[6] A lot of work is being done on symmetric variants for optimization [80], but those are not appropriate for FSI as the Jacobian that needs to be approximated is not symmetric [81].

to a low parallel efficiency, or analogously a number of idle cores. This problem is overcome by Jacobi iteration where the flow and structure solvers are calculating simultaneously. The cores are distributed over the two solvers such that both solvers require the same amount of calculation time, i.e., they are perfectly balanced, but this load balancing may not be trivial. Furthermore, specific variants of quasi-Newton methods have been developed for High-Performance Computing (HPC) such as the Compact Interface Quasi-Newton method (CIQN) [67], which is a parallel adaptation of IQN-ILS focused on efficiently combining partitions to realize a scalable implementation.

Multi-solver variants Also Multi-Solver (MS) versions of both IQN-ILS and IBQN-LS have been developed [68]. Once an FSI simulation can no longer be accelerated by increasing the number of cores per solver, the multi-solver algorithms can be applied for an additional speed-up. These multi-solver algorithms reduce the calculation time by running multiple instances of the flow solver and structural solver, while keeping the number of cores per solver constant and running each instance on one or more cluster nodes. One instance of the flow solver and of the structural solver perform coupling iterations like in the normal IQN-ILS or IBQN-LS algorithm. However, data from previous time steps is not reused directly as explained in Sect. 5.1, because the relation between the columns of $\boldsymbol{R}^n$ and $\widetilde{\boldsymbol{X}}^n$ is only approximate at t^{n+1}. The additional instances of the flow solver and structural solver first recalculate the data from the previous time steps at the current time level, before including that data in a least-squares model. The columns of the matrix $\boldsymbol{X}^n$ contain specific combinations of the degrees of freedom on the interface that accelerated the convergence of the coupling iterations in the previous time step. Hence, it is expected that knowing the difference of the output at t^{n+1} due to the same difference of the input as used at time level t^n will improve the least-squares model for the approximate Jacobian.

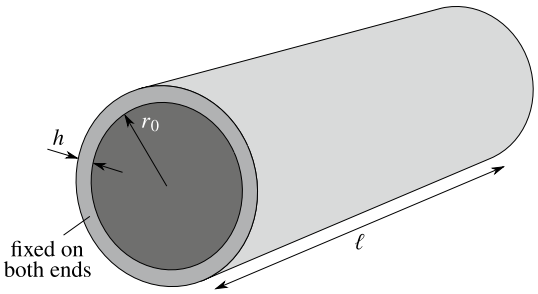


Fig. 6 Schematic representation of the tube geometry

Table 4 Parameter values of the 1D flexible tube case

Parameter	Value	Parameter	Value
ℓ	0.05 m	E	300,000 kg/m s^2
r_0	0.005 m	ν	0.3
h	0.001 m	ρ_f	1000 kg/m^3
Δt	0.0001 s	ρ_s	1200 kg/m^3

Multi-level variants Furthermore, there exist Multi-Level (ML) versions of IQN-ILS and IBQN-LS [69]. Those could be considered similar to IQN-ILSM with a coarse grid surrogate model. However, the multi-level algorithms have important disadvantages compared to IQN-ILSM and therefore the IQN-ILSM algorithm is recommended. First, the multi-level algorithms combine the secant conditions obtained on all grids, which gives them all the same priority. By contrast, IQN-ILSM gives the highest priority to the finest grid, with a diminishing contribution from the coarser grid(s) as the coupling iterations on the finest grid converge. Furthermore, the multi-level algorithms interpolate the secant conditions obtained on the coarser grid(s) to the finest grid level and store it at the highest resolution, while IQN-ILSM stores the secant conditions with the resolution at which they have been calculated.

Aitken relaxation In addition to the quasi-Newton algorithms, Aitken relaxation [70–72] is frequently used for partitioned FSI simulations. This technique uses a dynamically varying scalar relaxation factor ω^k for the Gauss–Seidel

Table 5 Average number of iterations required per time step for the different methods ($n_p = 100$)

Method	Number of iterations	Time (s)	Method	Number of iterations	Time (s)
Relaxation ($\omega = 0.01$)	820.98	241.40	Aitken relaxation	36.96	13.10
IQN-ILS ($q = 0$)	12.27	4.57	IBQN-LS ($q = 0$)	11.91	16.05
IQN-ILS ($q = 1$)	8.37	3.04	IBQN-LS ($q = 1$)	8.69	15.25
IQN-ILS ($q = 5$)	4.40	1.60	IBQN-LS ($q = 5$)	4.78	10.49
IQN-ILS ($q = 8$)	3.92	1.54	IBQN-LS ($q = 8$)	4.27	10.80
IQN-ILS ($q = 10$)	3.84	1.54	IBQN-LS ($q = 10$)	4.10	11.32
IQN-ILS ($q = 12$)	4.07	1.72	IBQN-LS ($q = 12$)	4.00	12.23
IQN-ILS ($q = 20$)	4.59	2.16	IBQN-LS ($q = 20$)	4.58	20.53
IQN-MVJ	4.19	1.62	MVQN	4.20	6.82
IQN-IMVLS	4.82	1.91			
IQN-ILSM (reuse)	4.19	2.17			
IQN-ILSM (coarse)	7.75	8.62			

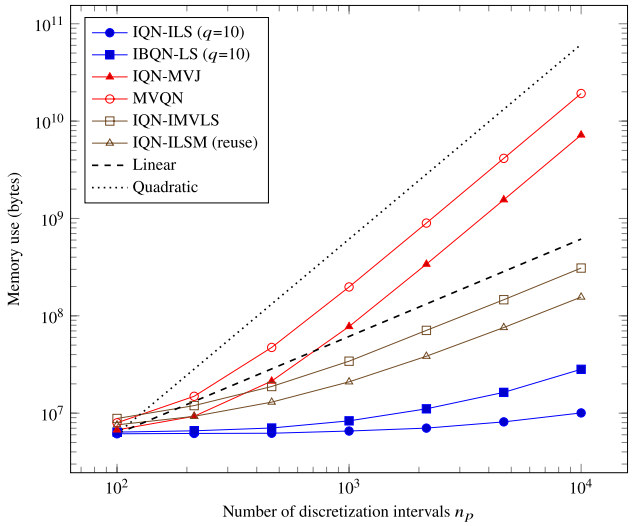


Fig. 7 Comparison of memory requirements for the different methods

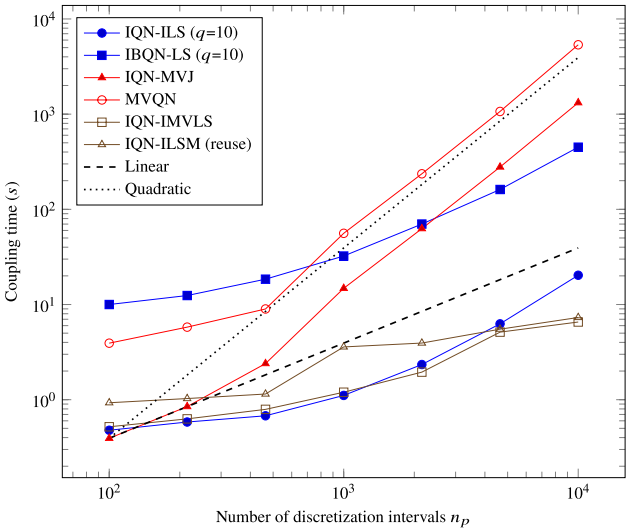


Fig. 8 Comparison of coupling time for the different methods

iterations within a time step. It can thus also be interpreted as an interface quasi-Newton technique: if the inverse of the Jacobian in Eq. (23) is approximated by $-\omega^k \boldsymbol{I}$, the Aitken relaxation method is retrieved.

$$\begin{aligned} \boldsymbol{x}^{k+1} &= \boldsymbol{x}^k + \omega^k \boldsymbol{r}^k \\ &= (1-\omega^k)\boldsymbol{x}^k + \omega^k \tilde{\boldsymbol{x}}^k \end{aligned} \tag{97}$$

The next input for $\mathcal{R}$ is thus a linear combination of the last output and the previous input. Therefore, the update of the interface's displacement is in the direction of the residual vector, as opposed to the update of the IQN-ILS method in Eq. (73). The value of ω^k is calculated recursively as

$$\omega^k = -\omega^{k-1} \frac{(\boldsymbol{r}^{k-1})^{\mathrm{T}}(\boldsymbol{r}^k - \boldsymbol{r}^{k-1})}{(\boldsymbol{r}^k - \boldsymbol{r}^{k-1})^{\mathrm{T}}(\boldsymbol{r}^k - \boldsymbol{r}^{k-1})} \tag{98}$$

which can be interpreted as the secant method for scalars directly applied to vectors and projected on $\boldsymbol{r}^k - \boldsymbol{r}^{k-1}$ [70]. By combining Eqs. (97) and (98), it can be seen that the update of the interface's displacement is given by

$$\begin{aligned} \boldsymbol{x}^{k+1} &= \boldsymbol{x}^k + \frac{(\boldsymbol{x}^k - \boldsymbol{x}^{k-1})^{\mathrm{T}}(\boldsymbol{r}^k - \boldsymbol{r}^{k-1})}{(\boldsymbol{r}^k - \boldsymbol{r}^{k-1})^{\mathrm{T}}(\boldsymbol{r}^k - \boldsymbol{r}^{k-1})}(-\boldsymbol{r}^k) \\ &= \boldsymbol{x}^k + \left[\frac{(\tilde{\boldsymbol{x}}^k - \tilde{\boldsymbol{x}}^{k-1})^{\mathrm{T}}(\boldsymbol{r}^k - \boldsymbol{r}^{k-1})}{(\boldsymbol{r}^k - \boldsymbol{r}^{k-1})^{\mathrm{T}}(\boldsymbol{r}^k - \boldsymbol{r}^{k-1})} - 1\right](-\boldsymbol{r}^k) \end{aligned} \tag{99}$$

for $k > 0$, which is similar to Eq. (73). If the Jacobian were created explicitly in the IQN-ILS algorithm and if the matrices $\boldsymbol{R}^k$ and $\widetilde{\boldsymbol{X}}^k$ were limited to their newest column, Eq. (73) would yield

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \left[\frac{(\tilde{\boldsymbol{x}}^k - \tilde{\boldsymbol{x}}^{k-1})(\boldsymbol{r}^k - \boldsymbol{r}^{k-1})^{\mathrm{T}}}{(\boldsymbol{r}^k - \boldsymbol{r}^{k-1})^{\mathrm{T}}(\boldsymbol{r}^k - \boldsymbol{r}^{k-1})} - \boldsymbol{I}\right](-\boldsymbol{r}^k). \tag{100}$$

Note the different location of the transpose sign in Eqs. (99) and Eq. (100). They are thus not identical because the coefficient of $-\boldsymbol{r}^k$ is a scalar in the first equation and a matrix in the second one. Consequently, Aitken relaxation is different from IQN-ILS, even when the latter is restricted to one column in the matrices $\boldsymbol{R}^k$ and $\widetilde{\boldsymbol{X}}^k$. While Aitken relaxation uses a single relaxation factor for all interface degrees of freedom, IQN-ILS assigns a different value to each one based on a combination of the previously determined modes, i.e., the columns of $\boldsymbol{R}^k$ and $\widetilde{\boldsymbol{X}}^k$.

Prediction In all the quasi-Newton algorithms, the coupling iterations begin from $\boldsymbol{x}^0$, which is an extrapolation or prediction. This can be a constant, linear or quadratic extrapolation based on $\boldsymbol{x}^n, \boldsymbol{x}^{n-1}, \ldots$. While there is an effect of the order of this prediction on the convergence of the coupling iterations, it is case dependent and it is thus difficult to state whether higher order is always faster [29]. In the IQN-ILSM algorithm, the surrogate model can also be used for a prediction, but this is also not always faster than a linear extrapolation [61].

Quasi-Newton methods with Robin–Neumann decomposition Instead of the typical Dirichlet–Neumann decomposition of the FSI problem, a Robin–Neumann decomposition can be used as well. This decomposition modifies the boundary condition in the flow solver to also include the pressure and traction forces. The effect is the introduction of Interface Artificial Compressibility (IAC), which allows to solve FSI

problems with enclosed incompressible fluids. To combine the idea of IAC with quasi-Newton methods a pressure correction has to be introduced to obtain corresponding inputs and outputs for the quasi-Newton technique [73]. However, by switching the order of the solvers and performing the quasi-Newton update on the pressure and traction forces instead of the displacements, this pressure correction can be avoided [74].

Applications in other fields Finally, it should be remarked that the quasi-Newton coupling techniques can also be used for coupled problems other than FSI. The main requirement for good results is significant interaction between both subproblems such that an approximate Jacobian stabilizes and accelerates the convergence compared to Gauss–Seidel iterations. The interaction should however not be so strong that an exact Jacobian satisfying all possible secant conditions is required to achieve convergence.

An example of a suitable problem is Conjugate Heat Transfer (CHT), where the partitioned spatial regions are each modelled by independent heat transfer codes and linked by temperature and flux matching conditions at the common boundaries [75]. A furnace radiation model can be coupled with a melt crystal growth model to investigate growth processes [76]. Another example is soil–structure interaction problems. The effect of excavations on the frame of a building can be studied by coupling a model of the soil's behaviour with a code for nonlinear structural dynamics [77]. In this case, the interaction between the models occurs at a relatively small number of points. Finally, they can even be used to calculate the combustion in a fluidized bed reactor under pressure [78]. The calculation of the carbon and oxygen concentrations is performed separately from the calculation of the temperature field. In this last example, there is only one domain and data are exchanged throughout the domain, as opposed to the other examples, where the domains do not overlap and data are only exchanged at the common boundary of the subdomains. Therefore, the cost of the coupling is no longer negligible compared to the solution of the subproblems, which necessitates coupling techniques with low computational complexity and low memory requirements.

6 Numerical Tests

Several comparisons of quasi-Newton techniques can be found in the literature [52, 56, 59]. This section aims at providing an idea about the relative performance of the described techniques using a test case that can be reproduced in a straightforward way, but without claim of general applicability. It focuses on the scaling with increasing number of degrees of freedom on the interface.

6.1 Test Case

The six quasi-Newton techniques presented in Sect. 5 are now evaluated on a one-dimensional (1D) flexible tube, through which an incompressible fluid flows. This test case runs quickly and still features the destabilizing added mass phenomenon [35].

The straight tube has a length ℓ, a circular cross-section with nominal inner diameter r_0 and wall thickness h. Its geometry is sketched in Fig. 6.

The ratio of the fluid density ρ_f to the structure density ρ_s is close to one, resulting in a large added mass. The structure has a modulus of elasticity E and a Poisson's ratio ν. The values of these parameters are reported in Table 4.

On the inlet, a pressure pulse of 1333.2 Pa is applied for a duration of 0.003 s. Due to the flexible tube wall, the pulse travels at a finite velocity, despite the incompressible fluid. The pressure at the outlet of the tube is atmospheric pressure, i.e., 0 Pa. The total simulation time is 0.01 s, divided in 100 time steps.

The flow solver solves the nonlinear continuity and momentum equation in which viscosity and gravity are neglected. The deformation of the tube wall is calculated by the structure solver, which only considers radial displacements, so the length of the tube stays constant. The tube is discretized in n_p equal intervals. For the details regarding the governing equations and applied discretizations, the reader is referred to [35].

The FSI problem is solved with the open-source code *CoCoNuT*. This code allows to couple different software packages, which are treated as black boxes. It has the advantage of being modular and flexible in combining interpolators, solvers and coupling algorithms. Moreover, due to its comprehensive structure and implementation in Python, the code can be adapted to the user's own requirements, if needed. The most recent code can be found on GitHub in the repository https://github.com/pyfsi/coconut/. The version used in this paper as well as scripts and result data are available for download on the Zenodo platform [79]. Both a flow and structure solver were written for the 1D flexible tube case described above and this code is equally available online.

6.2 Results

As the solvers are typically very expensive in FSI calculations, the focus is on the number of solver executions per time step. The term iteration will be used to denote a subsequent flow and structure solver call. Table 5 shows the number of iterations for the different quasi-Newton methods discussed above, as well as for a fixed relaxation factor and Aitken relaxation. Time data is included for completeness. A relative convergence tolerance is used, so that a

time step is considered converged when $\|\boldsymbol{r}^k\|_2 < 10^{-6}\|\boldsymbol{r}^0\|_2$, with $\boldsymbol{r}^0$ the residual at the start of the time step.

Dynamic Aitken relaxation improves the convergence greatly compared to simple relaxation, but does not perform as well as the quasi-Newton methods. The block iteration and residual formulation methods perform similarly, whereas the least-squares techniques turn out to be somewhat more efficient compared to the multi-vector techniques for this test case. IQN-ILS ($q = 0$) corresponds to the method without reuse from previous time steps, but the IQN-ILS method performs best for q equal to 10. This shows the importance of reusing information from previous time steps. Note that the average number of coupling iterations per time step is not so sensitive to the value of q around the optimum. The indications *(reuse)* and *(coarse)* for the IQN-ILSM algorithm, refer respectively to the use of the previous time steps, and the use of an identical problem with half as many discretization intervals (without reuse) as surrogate model.

Next, Fig. 7 shows the memory requirements, for different number of discretization intervals n_p. The data depict the peak memory usage as determined using the Python module guppy3.[7] The reported values are the total memory use minus that of the solvers. The results clearly show quadratically scaling memory requirements for the multi-vector techniques with explicit Jacobian construction, in contrast to the linear scaling for the methods with matrix-free implementation. Starting from 10^4 degrees of freedom, the memory requirements of MVQN and IQN-MVJ become much higher than those of the other coupling techniques. Furthermore, the block iteration methods require more memory compared to their corresponding residual formulation techniques, as two Jacobian matrices are approximated. Finally, IQN-IMVLS and IQN-ILSM (reuse) require more memory compared to the other matrix-free methods, because they retain information form every time step ($q = 100$), instead of only from the last few ($q = 10$). However, this could be reduced by applying a truncation after q time steps in the recursive formulas.

Lastly, Fig. 8 reports the coupling time, for different number of discretization intervals n_p. The coupling time is calculated as the run time, excluding initialization, minus the time spent in both solvers and the interpolation routine. It is important to note that the code generating these results is not optimized for speed, such that care has to be taken when interpreting the results. Nonetheless, some relevant conclusions can be made. For example, remark that the quadratically scaling methods IQN-MVJ and MVQN require considerably more time than their linearly scaling counterparts. The same can be concluded for the block iterations methods, which are implemented here with an iterative procedure to obtain the quasi-Newton updates. Using the Woodbury matrix identity instead is expected to reduce the computational cost, but not to the extent that they will become cheaper than residual formulation techniques. Finally, the coupling time for the linearly scaling residual formulation techniques are in the order of 10 s for the investigated range of discretization intervals. These values remain low compared to the typical cost of solving the subproblems. Therefore, the difference in computational cost of the coupling for these methods is only moderately important.

7 Conclusion

The IQN-ILS, IBQN-LS, MVQN, IQN-MVJ, IQN-IMVLS and IQN-ILSM can all be reformulated as quasi-Newton techniques using generalized Broyden techniques for the approximate (inverse of the) Jacobian. In a time-dependent simulation with reuse from previous time steps, IQN-ILS and IBQN-LS enforce the secant conditions from all time steps in the same way, while MVQN, IQN-MVJ, IQN-IMVLS and IQN-ILSM only enforce the secant conditions from the latest time step directly. The block iteration quasi-Newton techniques like IBQN-LS and MVQN achieve similar performance in terms of number of coupling iterations per time step compared to the equivalent residual formulation quasi-Newton techniques IQN-ILS and IQN-MVJ, but the block iteration quasi-Newton techniques require more memory, more computational time for the coupling and a longer source code, entailing more involved implementation. Hence, it can be stated that, in general terms, it is better to use a residual formulation quasi-Newton method instead of the block iteration quasi-Newton techniques. For real-scale applications with a reasonable number of degrees of freedom on the interface, linear scaling of the coupling technique is essential, so MVQN and IQN-MVJ can only be used for smaller problems. The IQN-ILS technique has a short implementation and linear scaling thanks to the matrix-free procedure, but it requires filtering when reusing secant conditions from previous time steps. Finally, IQN-ILSM can be seen as a generalization of IQN-IMVLS, such that not only reuse from previous time steps can be applied but also other prior knowledge can be included to accelerate the convergence.

Acknowledgements The authors humbly acknowledge the contribution of Jan Vierendeels (1968–2018) to the field of partitioned simulation of fluid–structure interaction.

Funding N. Delaissé gratefully acknowledges the funding received from the Research Foundation–Flanders (FWO) as part of the SBO CONTACTLUB project (S006519N).

[7] See GitHub repository https://github.com/zhuyifei1999/guppy3/.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Bazilevs Y, Hsu MC, Kiendl J, Wuechner R, Bletzinger KU (2010) 3D simulation of wind turbine rotors at full scale. Part II: fluid–structure interaction modeling with composite blades. Int J Numer Methods Fluids 65(1–3):236–253. https://doi.org/10.1002/fld.2454
2. Santo G, Peeters M, Van Paepegem W, Degroote J (2019) Dynamic load and stress analysis of a large horizontal axis wind turbine using full scale fluid–structure interaction simulation. Renew Energy 140:212–226. https://doi.org/10.1016/j.renene.2019.03.053
3. Paidoussis M (2016) Fluid–structure interactions: volume 2 slender structures and axial flow, 2nd edn. Elsevier, Amsterdam
4. Billah KY, Scanlan RH (1991) Resonance, Tacoma narrows bridge failure, and undergraduate physics textbooks. Am J Phys 59(2):118–124. https://doi.org/10.1119/1.16590
5. Hillewaere J, Degroote J, Lombaert G, Vierendeels J, Degrande G (2015) Wind–structure interaction simulations of ovalling vibrations in silo groups. J Fluids Struct 59:328–350. https://doi.org/10.1016/j.jfluidstructs.2015.09.013
6. Narayanan NK, Wüchner R, Degroote J (2019) Coupling of structural solver and volume-conserving solver for form-finding of membrane structures subjected to ponding. In: 8th international conference on computational methods for coupled problems in science and engineering, Sitges. pp 1–12. http://hdl.handle.net/1854/LU-8639251
7. Kamensky D, Hsu MC, Schillinger D, Evans JA, Aggarwal A, Bazilevs Y, Sacks MS, Hughes TJR (2015) An immersogeometric variational framework for fluid–structure interaction: application to bioprosthetic heart valves. Comput Methods Appl Mech Eng 284:1005–1053. https://doi.org/10.1016/j.cma.2014.10.040
8. Peskin CS, McQueen DM (1989) A three-dimensional computational method for blood flow in the heart I. Immersed elastic fibers in a viscous incompressible fluid. J Comput Phys 81(2):372–405. https://doi.org/10.1016/0021-9991(89)90213-1
9. Schott B, Ager C, Wall WA (2019) A monolithic approach to fluid–structure interaction based on a hybrid Eulerian-ALE fluid domain decomposition involving cut elements. Int J Numer Methods Eng 119(3):208–237. https://doi.org/10.1002/nme.6047
10. Ryzhakov PB, Marti J, Dialami N (2022) A unified arbitrary Lagrangian–Eulerian model for fluid–structure interaction problems involving flows in flexible channels. J Sci Comput. https://doi.org/10.1007/s10915-021-01748-w
11. Farhat C, van der Zee KG, Geuzaine P (2006) Provably second-order time-accurate loosely-coupled solution algorithms for transient nonlinear computational aeroelasticity. Comput Methods Appl Mech Eng 195(17–18):1973–2001. https://doi.org/10.1016/j.cma.2004.11.031
12. Lesoinne M, Farhat C (1998) A higher-order subiteration free staggered algorithm for non-linear transient aeroelastic problems. Am Inst Aeronaut Astronaut J 36(9):1754–1756. https://doi.org/10.2514/3.14041
13. van Brummelen EH (2009) Added mass effects of compressible and incompressible flows in fluid–structure interaction. J Appl Mech 76(2):021206. https://doi.org/10.1115/1.3059565
14. Boilevin-Kayl L, Fernandez MA, Gerbeau JF (2019) A loosely coupled scheme for fictitious domain approximations of fluid–structure interaction problems with immersed thin-walled structures. SIAM J Sci Comput 41(2):351–374. https://doi.org/10.1137/18m1192779
15. Causin P, Gerbeau JF, Nobile F (2005) Added-mass effect in the design of partitioned algorithms for fluid–structure problems. Comput Methods Appl Mech Eng 194(42–44):4506–4527. https://doi.org/10.1016/j.cma.2004.12.005
16. Förster C, Wall WA, Ramm E (2007) Artificial added mass instabilities in sequential staggered coupling of nonlinear structures and incompressible viscous flows. Comput Methods Appl Mech Eng 196(7):1278–1293. https://doi.org/10.1016/j.cma.2006.09.002
17. Le Tallec P, Mouro J (2001) Fluid structure interaction with large structural displacements. Comput Methods Appl Mech Eng 190(24–25):3039–3067. https://doi.org/10.1016/s0045-7825(00)00381-9
18. Gerbeau JF, Vidrascu M, Frey P (2005) Fluid–structure interaction in blood flows on geometries based on medical imaging. Comput Struct 83(2–3):155–165. https://doi.org/10.1016/j.compstruc.2004.03.083
19. Förster C, Wall WA, Ramm E (2006) The artificial added mass effect in sequential staggered fluid–structure interaction algorithms. In: Wesseling P, Oñate E, Périaux J (eds) 4th European conference on computational fluid dynamics, Egmond aan Zee. pp 1–20. http://resolver.tudelft.nl/uuid:f6316ca6-b02a-4a6f-b5ad-4e2f6744c7de
20. Riemslagh K, Vierendeels J, Dick E (2000) An efficient coupling procedure for flexible wall fluid–structure interaction. In: 3th European congress on computational methods in applied sciences and engineering, Barcelona. pp 1–13. http://hdl.handle.net/1854/LU-128926
21. Vierendeels J, Lanoye L, Degroote J, Verdonck PR (2007) Implicit coupling of partitioned fluid–structure interaction problems with reduced order models. Comput Struct 85(11–14):970–976. https://doi.org/10.1016/j.compstruc.2006.11.006
22. Taelman L, Bols J, Degroote J, Muthurangu V, Panzer J, Vierendeels J, Segers P (2015) Differential impact of local stiffening and narrowing on hemodynamics in repaired aortic coarctation: an FSI study. Med Biol Eng Comput 54(2–3):497–510. https://doi.org/10.1007/s11517-015-1336-1
23. Dolfen H, De Ridder J, Brockmeyer L, Merzari E, Kennedy G, Van Tichelen K, Degroote J (2022) A multi-stage approach of simulating turbulence-induced vibrations in a wire-wrapped tube bundle for fretting wear prediction. J Fluids Struct 109:103460. https://doi.org/10.1016/j.jfluidstructs.2021.103460
24. Delcour L, Bral A, Van Langenhove L, Degroote J (2022) Investigating the influence of compressibility on the second mode flutter instability of a clamped-free cylinder in axial flow using fluid–structure interaction simulations with the Chimera technique. J Fluids Struct 109:103469. https://doi.org/10.1016/j.jfluidstructs.2021.103469
25. Badia S, Nobile F, Vergara C (2008) Fluid–structure partitioned procedures based on Robin transmission conditions. J Comput Phys 227(14):7027–7051. https://doi.org/10.1016/j.jcp.2008.04.006

26. Deuse M, Sandberg RD (2020) Implementation of a stable high-order overset grid method for high-fidelity simulations. Comput Fluids 211:104449. https://doi.org/10.1016/j.compfluid.2020.104449
27. Zhang ZQ, Liu GR, Khoo BC (2012) A three dimensional immersed smoothed finite element method (3D IS-FEM) for fluid–structure interaction problems. Comput Mech 51(2):129–150. https://doi.org/10.1007/s00466-012-0710-1
28. Zorrilla R, Rossi R, Wüchner R, Oñate E (2020) An embedded finite element framework for the resolution of strongly coupled fluid–structure interaction problems. Application to volumetric and membrane-like structures. Comput Methods Appl Mech Eng 368:113179. https://doi.org/10.1016/j.cma.2020.113179
29. Erbts P, Düster A (2012) Accelerated staggered coupling schemes for problems of thermoelasticity at finite strains. Comput Math Appl 64(8):2408–2430. https://doi.org/10.1016/j.camwa.2012.05.010
30. Joosten MM, Dettmer WG, Perić D (2009) Analysis of the block Gauss–Seidel solution procedure for a strongly coupled model problem with reference to fluid–structure interaction. Int J Numer Methods Eng 78(7):757–778. https://doi.org/10.1002/nme.2503
31. Idelsohn SR, Del Pin F, Rossi R, Oñate E (2009) Fluid–structure interaction problems with strong added-mass effect. Int J Numer Methods Eng 80(10):1261–1294. https://doi.org/10.1002/nme.2659
32. Formaggia L, Gerbeau JF, Nobile F, Quarteroni A (2001) On the coupling of 3D and 1D Navier–Stokes equations for flow problems in compliant vessels. Comput Methods Appl Mech Eng 191(6–7):561–582. https://doi.org/10.1016/s0045-7825(01)00302-4
33. Vierendeels J, Dumont K, Dick E, Verdonck PR (2005) Analysis and stabilization of fluid–structure interaction algorithm for rigid-body motion. Am Inst Aeronaut Astronaut J 43(12):2549–2557. https://doi.org/10.2514/1.3660
34. Degroote J, Bruggeman P, Haelterman R, Vierendeels J (2008) Stability of a coupling technique for partitioned solvers in FSI applications. Comput Struct 86(23–24):2224–2234. https://doi.org/10.1016/j.compstruc.2008.05.005
35. Degroote J, Annerel S, Vierendeels J (2010) Stability analysis of Gauss–Seidel iterations in a partitioned simulation of fluid–structure interaction. Comput Struct 88(5–6):263–271. https://doi.org/10.1016/j.compstruc.2009.09.003
36. Degroote J (2013) Partitioned simulation of fluid–structure interaction: coupling black-box solvers with quasi-Newton techniques. Arch Comput Methods Eng 20(3):185–238. https://doi.org/10.1007/s11831-013-9085-5
37. van Brummelen EH (2010) Partitioned iterative solution methods for fluid–structure interaction. Int J Numer Methods Fluids 65(1–3):3–27. https://doi.org/10.1002/fld.2465
38. Golub GH, Van Loan CF (1996) Matrix computations, 3rd edn. Johns Hopkins University Press, Baltimore https://doi.org/10.56021/9781421407944
39. Trefethen LN, Bau DI (1997) Numerical linear algebra. Cambridge. https://doi.org/10.1137/1.9780898719574
40. Broyden CG (1965) A class of methods for solving nonlinear simultaneous equations. Math Comput 19(92):577–593. https://doi.org/10.1090/s0025-5718-1965-0198670-6
41. Fang HR, Saad Y (2009) Two classes of multisecant methods for nonlinear acceleration. Numer Linear Algebr Appl 16(3):197–221. https://doi.org/10.1002/nla.617
42. Johnson DD (1988) Modified Broyden's method for accelerating convergence in self-consistent calculations. Phys Rev B 38(18):12807–12813. https://doi.org/10.1103/physrevb.38.12807
43. Vanderbilt D, Louie SG (1984) Total energies of diamond (111) surface reconstructions by a linear combination of atomic orbitals method. Phys Rev B 30(10):6118–6130. https://doi.org/10.1103/physrevb.30.6118
44. Eyert V (1996) A comparative study on methods for convergence acceleration of iterative vector sequences. J Comput Phys 124(2):271–285. https://doi.org/10.1006/jcph.1996.0059
45. Anderson DG (1965) Iterative procedures for nonlinear integral equations. J ACM 12(4):547–560. https://doi.org/10.1145/321296.321305
46. van Leuken H (1991) Electronic structure of metallic multilayers. PhD thesis, University of Amsterdam
47. Scheufele K (2015) Robust quasi-Newton methods for partitioned fluid–structure simulations. Master's thesis, University of Stuttgart. https://doi.org/10.13140/RG.2.2.28442.08648
48. Uekermann BW (2016) Partitioned fluid–structure interaction on massively parallel systems. PhD thesis, Technische Universität München. https://doi.org/10.14459/2016md1320661
49. Demeester T, Delaissé N, van Brummelen EH, Haelterman R, Degroote J (2022) On the effect of nonlinearity and Jacobian initialization on the convergence of the generalized Broyden quasi-Newton method. Int J Numer Methods Eng 123(17):4054–4072. https://doi.org/10.1002/nme.6998
50. Degroote J, Bathe KJ, Vierendeels J (2009) Performance of a new partitioned procedure versus a monolithic procedure in fluid–structure interaction. Comput Struct 87(11–12):793–801. https://doi.org/10.1016/j.compstruc.2008.11.013
51. Haelterman R, Degroote J, Van Heule D, Vierendeels J (2009) The quasi-Newton least squares method: a new and fast secant method analyzed for linear systems. SIAM J Numer Anal 47(3):2347–2368. https://doi.org/10.1137/070710469
52. Gallinger T, Bletzinger KU (2010) Comparison of algorithms for strongly coupled partitioned fluid–structure interaction—efficiency versus simplicity. In: Pereira JCF, Sequeira A (eds) 5th European conference on computational fluid dynamics, Lisbon. pp 1–20
53. Spenke T, Hosters N, Behr M (2020) A multi-vector interface quasi-Newton method with linear complexity for partitioned fluid–structure interaction. Comput Methods Appl Mech Eng 361:112810. https://doi.org/10.1016/j.cma.2019.112810
54. Degroote J, Souto-Iglesias A, Van Paepegem W, Annerel S, Bruggeman P, Vierendeels J (2010) Partitioned simulation of the interaction between an elastic structure and free surface flow. Comput Methods Appl Mech Eng 199(33–36):2085–2098. https://doi.org/10.1016/j.cma.2010.02.019
55. Haelterman R, Bogaers AEJ, Scheufele K, Uekermann B, Mehl M (2016) Improving the performance of the partitioned QN-ILS procedure for fluid–structure interaction problems: filtering. Comput Struct 171:9–17. https://doi.org/10.1016/j.compstruc.2016.04.001
56. Degroote J, Haelterman R, Annerel S, Bruggeman P, Vierendeels J (2010) Performance of partitioned procedures in fluid–structure interaction. Comput Struct 88(7–8):446–457. https://doi.org/10.1016/j.compstruc.2009.12.006
57. Zorrilla R, Rossi R (2023) A memory-efficient multivector quasi-Newton method for black-box fluid–structure interaction coupling. Comput Struct 275:106934. https://doi.org/10.1016/j.compstruc.2022.106934
58. Bogaers AEJ, Kok S, Reddy BD, Franz T (2014) Quasi-Newton methods for implicit black-box FSI coupling. Comput Methods Appl Mech Eng 279:113–132. https://doi.org/10.1016/j.cma.2014.06.033
59. Lindner F, Mehl M, Scheufele K, Uekermann B (2015) A comparison of various quasi-Newton schemes for partitioned fluid–structure interaction. In: Schrefler B, Oñate E, Papadrakakis M (eds) 6th international conference on computational methods for coupled problems in science and engineering. pp 477–488. http://hdl.handle.net/2117/191193

60. Scheufele K, Mehl M (2017) Robust multisecant quasi-Newton variants for parallel fluid–structure simulations and other multiphysics applications. SIAM J Sci Comput 39(5):404–433. https://doi.org/10.1137/16m1082020
61. Delaissé N, Demeester T, Fauconnier D, Degroote J (2022) Surrogate-based acceleration of quasi-Newton techniques for fluid–structure interaction simulations. Comput Struct 260:106720. https://doi.org/10.1016/j.compstruc.2021.106720
62. Demeester T, Brummelen EH, Degroote J (2020) An efficient quasi-Newton method for two-dimensional steady free surface flow. Int J Numer Methods Fluids 92(7):785–801. https://doi.org/10.1002/fld.4806
63. Demeester T, van Brummelen EH, Degroote J (2021) An efficient quasi-Newton method for three-dimensional steady free surface flow. Int J Numer Methods Fluids 93(8):2581–2610. https://doi.org/10.1002/fld.4989
64. De Groote W, Kikken E, Hostens E, Van Hoecke S, Crevecoeur G (2022) Neural network augmented physics models for systems with partially unknown dynamics: application to slider-crank mechanism. IEEE/ASME Trans Mechatron 27(1):1–11. https://doi.org/10.1109/tmech.2021.3058536
65. Mehl M, Uekermann B, Bijl H, Blom D, Gatzhammer B, van Zuijlen AH (2016) Parallel coupling numerics for partitioned fluid–structure interaction simulations. Comput Math Appl 71(4):869–891. https://doi.org/10.1016/j.camwa.2015.12.025
66. Cervera M, Codina R, Galindo M (1996) On the computational efficiency and implementation of block-iterative algorithms for nonlinear coupled problems. Eng Comput 13(6):4–30. https://doi.org/10.1108/02644409610128382
67. Santiago A, Zavala-Aké M, Borrell R, Houzeaux G, Vázquez M (2020) HPC compact quasi-Newton algorithm for interface problems. J Fluids Struct 96:103009. https://doi.org/10.1016/j.jfluidstructs.2020.103009
68. Degroote J, Vierendeels J (2011) Multi-solver algorithms for the partitioned simulation of fluid–structure interaction. Comput Methods Appl Mech Eng 200(25–28):2195–2210. https://doi.org/10.1016/j.cma.2011.03.015
69. Degroote J, Vierendeels J (2012) Multi-level quasi-Newton coupling algorithms for the partitioned simulation of fluid–structure interaction. Comput Methods Appl Mech Eng 225–228:14–27. https://doi.org/10.1016/j.cma.2012.03.010
70. Küttler U, Wall WA (2008) Fixed-point fluid–structure interaction solvers with dynamic relaxation. Comput Mech 43(1):61–72. https://doi.org/10.1007/s00466-008-0255-5
71. Mok DP, Wall WA (2001) Partitioned analysis schemes for the transient interaction of incompressible flows and nonlinear flexible structures. In: Schweizerhof K, Wall WA, Bletzinger KU (eds) Trends in computational structural mechanics, Barcelona. pp 688–698
72. Mok DP, Wall WA, Ramm E (2001) Accelerated iterative substructuring schemes for instationary fluid–structure interaction. In: Bathe KJ (ed) Computational fluid and solid mechanics. Elsevier, Amsterdam, pp 1325–1328. https://doi.org/10.1016/b978-008043944-0/50907-0
73. Bogaers AEJ, Kok S, Reddy BD, Franz T (2015) Extending the robustness and efficiency of artificial compressibility for partitioned fluid–structure interactions. Comput Methods Appl Mech Eng 283:1278–1295. https://doi.org/10.1016/j.cma.2014.08.021
74. Spenke T, Make M, Hosters N (2022) A Robin–Neumann scheme with quasi-Newton acceleration for partitioned fluid–structure interaction. Int J Numer Methods Eng 124(4):979–997. https://doi.org/10.1002/nme.7151
75. Rüth B, Uekermann B, Mehl M, Birken P, Monge A, Bungartz HJ (2020) Quasi-Newton waveform iteration for partitioned surface-coupled multiphysics applications. Int J Numer Methods Eng 122(19):5236–5257. https://doi.org/10.1002/nme.6443
76. Yeckel A, Lun L, Derby JJ (2009) An approximate block Newton method for coupled iterations of nonlinear solvers: theory and conjugate heat transfer applications. J Comput Phys 228(23):8566–8588. https://doi.org/10.1016/j.jcp.2009.08.003
77. Jahromi HZ, Izzuddin BA, Zdravkovic L (2009) A domain decomposition approach for coupled modelling of nonlinear soil–structure interaction. Comput Methods Appl Mech Eng 198(33–36):2738–2749. https://doi.org/10.1016/j.cma.2009.03.018
78. Artlich S, Mackens W (1995) Newton-coupling of fixed point iterations. In: Hackbusch W, Wittum G (eds) 11th GAMM-seminar: numerical treatment of coupled systems. Notes on numerical fluid mechanics, vol 51. Vieweg, Braunschweig, Kiel, pp 1–10. https://doi.org/10.1007/978-3-322-86859-6_1
79. Delaissé N, Demeester T, Haelterman R, Degroote J (2023) Quasi-Newton methods for partitioned simulation of fluid–structure interaction reviewed in the generalized Broyden framework: code and data. https://doi.org/10.5281/zenodo.7565680
80. Boutet N, Haelterman R, Degroote J (2021) Secant update generalized version of PSB: a new approach. Comput Optim Appl 78(3):953–982. https://doi.org/10.1007/s10589-020-00256-1
81. Matthies HG, Niekamp R, Steindorf J (2006) Algorithms for strong coupling procedures. Comput Methods Appl Mech Eng 195(17–18):2028–2049. https://doi.org/10.1016/j.cma.2004.11.032