Surrogate-based acceleration of quasi-Newton techniques for fluid-structure interaction simulations

Nicolas Delaissé∗, Toon Demeester, Dieter Fauconnier, Joris Degroote

*Department of Electromechanical, Systems and Metal Engineering, Ghent University
Sint-Pietersestraat 41, 9000 Ghent, Belgium

Soete Laboratory, Department of Electromechanical, Systems and Metal Engineering, Ghent University
Technologiepark Zwijnaarde 46, 9052 Zwijnaarde, Belgium

Flanders Make @ UGent – Core Lab EEDT-MP

Abstract

Quasi-Newton methods have proven to be an efficient way to couple partitioned solvers in fluid-structure interaction problems, as they are able to stabilize cases with high added-mass, as well as accelerate the convergence. However, these methods assume that the coupled system is a complete black-box, whereas often, its behavior is well approximated by a surrogate model. Such a model may be obtained by coarsening the system, simplifying the physics, reverting to analytical approximations or considering the system at a previous point in time. The principal idea of this work is to use an initial solution and a Jacobian provided by the surrogate model, to expedite the convergence even further. This article presents a new framework for the inclusion of surrogate models in quasi-Newton methods and positions several existing methods with respect to it.

Keywords: Fluid-structure interaction, Quasi-Newton method, Surrogate model, Partitioned algorithm

1. Introduction

The interaction between a fluid and a movable or deformable structure is crucial in a wide range of engineering disciplines. Few examples include the design of slender constructions and structures [1, 2], the fracturing of vessels and pipes under impulsive loads [3], the flow of blood in the cardiovascular system [4] and the flutter of aircraft wings [5]. The widespread occurrence of these fluid-structure interactions (FSI) explains the much received attention in literature.

There are two approaches to solving FSI problems. They can either be solved in a monolithic way, by solving all equations simultaneously with a dedicated solver, or using a partitioned approach, which employs separate solvers for the fluid and structure subdomains. In practice, the latter is often selected, because it allows to use existing, optimized and validated solvers. As additional benefit to the partitioned approach, the discretization and solution techniques can be tailored to each subdomain. The most important disadvantage is the need for iterations between the flow and structure solver to impose the equilibrium conditions at the fluid-structure interface.

The most straightforward partitioned solution technique is the so-called Gauss-Seidel iteration. Within such an iteration, the structure and flow solver are evaluated sequentially until convergence is reached. With a Dirichlet-Neumann decomposition of the interface conditions, for example, the pressure and traction calculated by the flow solver are used as boundary conditions for the structure solver, and the displacement resulting from the structure calculation is in turn used to update the mesh of the flow domain and as boundary condition for the flow calculation.
A large part of literature focuses on FSI simulations with incompressible flow. These problems are particularly challenging, because they exhibit numerical difficulties due to the added-mass effect, especially when the densities of fluid and structure are the same order of magnitude \([6, 7]\). As a result, Gauss-Seidel iterations have a poor convergence behavior in these cases, if they converge at all. Several analyses have been conducted to examine the stability of Gauss-Seidel iterations \([6, 8, 9]\). Interestingly, they show that for incompressible FSI with high added-mass, only a small number of modes are unstable and that these unstable modes correspond to the error components with the lowest wavenumbers.

To stabilize the coupling, derivative information can be included in the iteration update with the use of a quasi-Newton method. Such a method performs Newton-Raphson iterations using a Jacobian that is approximated based on the secant information obtained from consecutive input-output pairs. It allows the solvers to be so-called black-box solvers, which do not allow access to the source code. Moreover, quasi-Newton methods perform well because the instability due to added-mass is caused by only a small number of modes in the error, as was discovered in the stability analyses mentioned above. Therefore, the low-rank approximation of the Jacobian, obtained from secant information, is sufficient to stabilize the coupling. It controls the unstable modes, while the other modes are still handled by Gauss-Seidel iterations. All quasi-Newton techniques are based on this observation and use input-output pairs from previous solver evaluations to construct and continuously improve the approximation of the Jacobian.

In literature, several quasi-Newton techniques have been proposed. Overall, these methods can be divided in two classes, based on the iteration scheme \([10]\). The block methods require the approximation of two Jacobians: one for each solver. The other class of methods is of the (displacement) residual formulation type. They have the advantage that only one (inverse) Jacobian approximation is required. This article focuses on the residual formulation methods, as their implementation is more straightforward. However, the discussed techniques and conclusions are equally valid for block variants.

Within the class of the residual formulation methods, the first was the interface quasi-Newton method with approximation for the inverse of the Jacobian from a least-squares model (IQN-ILS) \([11]\). Later followed the interface quasi-Newton multiple vector Jacobian method (IQN-MVJ) \([12]\). Some variations of these multi-vector methods have been proposed that focus on linearly scaling memory requirements, by avoiding the storage of large square matrices \([13, 14]\). Although the essence of all these quasi-Newton methods is the same, they differ in the way they use the derivative information, gathered throughout the calculation, to construct the Jacobian approximation.

Exploiting derivative information to stabilize and accelerate the coupling is common among all quasi-Newton techniques. However, at the start of the calculation, there are no previous solver evaluations and hence a Jacobian approximation is not yet available. Therefore, a quasi-Newton technique typically starts with a (relaxed) Gauss-Seidel iteration. Moreover, the first iteration of a time step requires an initial guess, which is typically obtained as an extrapolation of the results of previous time steps, while the initial guess in the first time step is usually the undeformed structure. This approach thus assumes there is no knowledge of the system beforehand. In many cases, however, information is either immediately available or straightforwardly obtained before the actual calculation. This work will use the general term surrogate model to denote the source of this type of prior information. As the references to the literature below will demonstrate, the use of such a surrogate model can be twofold. On the one hand, it can provide an approximation of the actual solution at the start of every time step. On the other hand, it can also provide an approximation of the Jacobian, i.e. of the derivative information.

A surrogate model may be any source of useful information with respect to the problem that is either readily available or straightforward to obtain. However, it is vital that such a model is cheaper and faster to evaluate than the actual solvers. Typically, this means that it does not resolve all physically relevant phenomena or is less accurate. The use of a surrogate model is only beneficial if it decreases the number of coupling iterations of the actual solvers in an amount that outweighs the additional cost of its own evaluation, such that the overall calculation is computationally less demanding. Moreover, one should evaluate the time required to construct the surrogate model relative to the gain over all its use cases.

Although not always denoted as such, the idea of employing a surrogate model is not new and has already been explored by different authors. Often, only the approximated solution is transferred, such as with Piperno and Farhat \([15, 16]\), who increased the accuracy and stability of loosely coupled calculations with the help of prediction methods using an estimation of the structural behavior. Similarly, van Zuijlen et al. \([17]\) employed a coarse grid correction of the initial guess at the start of a time step. Other multi-grid techniques accelerate Gauss-Seidel iterations by providing a correction for the smooth error components on the fine grid through coarse grid iterations \([17, 18]\). These techniques
only rely on the similarity of the solution between the surrogate and the actual model, but never exchange derivative information.

One of the first methods that benefited from the similarity of the derivatives on different grid levels is the multi-level acceleration technique of van Zuijlen and Bijl [19]. It hinges on the observation that the dynamic relaxation factor in Aitken relaxation does not differ significantly between the coarse and fine grid levels. The multi-level variant of IQN-ILS by Degroote and Vierendeels [20] goes a step further, by constructing the Jacobian on the coarse grid and subsequently improving it on the finer grid levels.

More recently, mapping techniques from the field of optimization have been applied on FSI simulations with the goal to relate a low-fidelity and high-fidelity solver. Scholcz et al. [21] use aggressive space mapping, in which a space mapping function is constructed, using black-box input from both solvers. This function keeps track of the differences between the actual model and the surrogate model, and allows to map the solution of the surrogate model to the one of the actual model. Blom et al. [22], on the other hand, use manifold mapping. This technique constructs a manifold mapping function relating the output of the surrogate and actual models. Both mapping techniques perform an optimization of the surrogate model in every iteration, the result of which is subsequently used to update the inverse Jacobian of the respective mapping functions. A disadvantage of this technique is that, if the surrogate is coarser than the actual model, a restriction and interpolation operator is required to link the input of both models. This results in a non-vanishing high-frequency error in the final solution, because the iteration update is constructed with differences from the coarse grid level. Finally, the inclusion of a surrogate model within a quasi-Newton method has also been realized outside of the field of FSI simulations [23].

This work provides a general framework, called quasi-Newton method with approximation of the inverse Jacobian from a least-squares model and surrogate model (IQN-ILSM), for the use of a surrogate model both as predictor, providing an initial guess at the start of a time step, and as supplementary Jacobian. Moreover, by doing so, the relation with existing techniques is revealed. Further, the performance of four different types of surrogate models is discussed. The first type includes coarser versions of the solvers. The second type makes use of other low fidelity solvers. As third type, analytical variants are used. And, finally, in a transient problem, reuse of previous time steps is considered.

The remainder of this work is structured as follows. The next section introduces the basic FSI equations and is followed by Section 3, which discusses the methods used to interpolate between different solvers. The general IQN-ILSM algorithm and a detailed discussion of the use of the different surrogate models are presented in Section 4. Further, Section 5 introduces two example cases and the solvers that will be used to illustrate the new framework. In Section 6, the performance of the different surrogate models is examined for these two cases, in steady and transient conditions as well as in two and three dimensions. Finally, a conclusion is presented in Section 7.

2. FSI equations

An FSI problem consists of two domains $\Omega_f$ and $\Omega_s$, with respective boundaries $\Gamma_f$ and $\Gamma_s$. The subscripts $f$ and $s$ refer to the fluid and structure. These two domains have a common boundary $\Gamma_i = \Gamma_f \cap \Gamma_s$, which is called the fluid-structure interface and through which the interaction between the fluid and structure domain occurs. The governing equations are discretized in space in both the flow and structure domain. In the case of a transient simulation, a discretization in time is performed as well.

In a partitioned approach, the flow equations are solved in $\Omega_f$ using a certain displacement of the fluid-structure interface as boundary condition. Thereafter, the resulting pressure and traction on the interface are used to determine a boundary condition for the solution of the structure equations in $\Omega_s$. This is repeated until the equilibrium conditions on the fluid-structure interface are fulfilled up to a certain tolerance.

For the coupling algorithm, only the variables on the common boundary are of interest. The displacement of the fluid-structure interface with respect to the initial position is represented by the vector $x \in \mathbb{R}^n$. Likewise, the vector $y$ represents the pressure and traction components on the interface and will also be referred to as the load on the interface. Using these vector notations, the flow and structure solver are concisely written as

$$\tilde{y} = \mathcal{F}(x) \text{ and } \tilde{x} = \mathcal{S}(y),$$

respectively, in which all dependencies on previous time steps and on the variables outside of the common boundary are hidden. The output of a solver is indicated with a tilde, because it is not necessarily equal to the input of the other
solver. These functions are considered to be black-box solvers, meaning that the input and output are restricted to the fluid-structure interface and the Jacobian information is not accessible.

This work focuses on the displacement residual formulation, which writes the FSI problem as a function of the interface displacement $x$ only, by passing the output of the flow solver $\tilde{y}$ directly to the structure solver: $\tilde{x} = \mathcal{S} \circ \mathcal{F}(x)$. In this way, the dynamic equilibrium condition is exactly satisfied in every iteration. Conversely, the output $\tilde{x}$ of the structure solver may be modified to $x$, before passing it on as input to the flow solver. This modification can be a relaxation or can be based on an approximate Jacobian, as will be explained in Section 4.1. The consecutive solution of flow and structure solver is considered to be one coupling iteration and the corresponding residual $r$ is defined by the difference between $\tilde{x}$ and $x$. The residual operator $\mathcal{R}$, defined by

$$
\mathcal{R}(x) = \mathcal{S} \circ \mathcal{F}(x) - x = \tilde{x} - x, \quad (2)
$$

returns the residual $r = \mathcal{R}(x)$ as a function of the input $x$ of an iteration. The calculation has converged if the residual $r$ drops below a certain tolerance. This approach reduces the FSI problem to finding the root of the non-linear system of equations $\mathcal{R}(x)$.

### 3. Interpolation

The grids of the fluid and structure domain are typically non-conforming, meaning that the fluid-structure interface is discretized differently on both sides. Moreover, a surrogate model typically uses its own dedicated grid which differs from that of the flow or structure solver. The exchange of information on the fluid-structure interface thus requires the interpolation of data on different discretizations on this interface.

There are numerous manners to achieve interpolation. However, in accordance with the black-box characteristics of the solvers, the interpolation method should only require access to the data on the fluid-structure interface. One example is the use of radial basis mappers [24, 25, 26, 27], which is the technique applied in this article.

This interpolation method is often opted for, because of its high accuracy, combined with limited calculation time. As additional benefit, it can be applied for an arbitrary number of dimensions $d$. This technique constructs the interpolant by superimposing radial basis functions centered around the points with known values. Such a radial basis function $\phi : \mathbb{R}_+ \to \mathbb{R}$ with center $\vec{c} \in \mathbb{R}^d$ satisfies $\phi(\vec{x} - \vec{c}) = \phi(||\vec{x} - \vec{c}||)$, where $\vec{x}$ is a point in $\mathbb{R}^d$ and $||\cdot||$ the Euclidean norm. In this work, the norm of a vector is always calculated using the Euclidean norm, also called $L_2$ norm.

Many such functions exist, but in this work the one introduced by Wendland [26] is used, which is given by

$$
\phi(||\vec{x}||) = \begin{cases} 
(1 - ||\vec{x}||/r)^4 (4 ||\vec{x}||/r + 1) & \text{for } 0 \leq ||\vec{x}|| < r \\
0 & \text{for } r \leq ||\vec{x}|| 
\end{cases} \quad (3)
$$

The scalar $r$ is a scaling factor controlling the shape. As this radial basis function is zero for $||\vec{x}|| \geq r$, it has so-called compact support, which has the advantage of speeding up the calculation of the interpolation coefficients, as the matrices become sparse. Still, even with compact support, the resulting linear system would remain large if all known points would be used to construct an interpolant for a single unknown point. Therefore, for each unknown point, a local interpolant is constructed using only the $m$ nearest known points. Here, the value of $m$ is 9 in the one-dimensional (1D) and two-dimensional (2D) case, and 81 in the three-dimensional (3D) case. Moreover, the value of the support radius $r$ has a large impact on the accuracy and stability of the system [24]. A too low value results in an inaccurate radial basis approximation with narrow peaks, whereas a too high value increases the condition number exponentially. In this work, the support radius $r$ is set to twice the distance to the furthest of the $m$ nearest points, such that the support of every radial basis function includes each of the other $m$ nearest points.

Consider the interpolation of a general variable $z$, which is, for example, a component of displacement or a residual. For the interpolation of $z$ from grid $a$ to a point $\vec{x}_{b,k}$ on grid $b$, the interpolant is given by

$$
z(\vec{x}) = \sum_{j=1}^{m} \alpha_j \phi(||\vec{x} - \vec{x}_{a,j}||), \quad (4)
$$
where \( \vec{x}_{a,j} \), \( j = 1, \ldots, m \), are the \( m \) points on grid \( a \) that are closest to \( \vec{x}_b \). The scalars \( \alpha_j \) scale the radial basis functions in each point \( \vec{x}_{a,j} \) and are determined such that the interpolant satisfies the given values on grid \( a \). Expressed as an equation, this results in

\[
z(\vec{x}_{a,j}) = z_{a,j}, \quad j = 1, \ldots, m,
\]

or when written in matrix form

\[
\alpha = \Phi^{-1} z,
\]

where \( z, \alpha \in \mathbb{R}^{m \times 1} \) are two vectors containing \( z_{a,j} \) and \( \alpha_j \), and \( \Phi \) is a symmetric matrix with elements \( \phi_{i,j} = \phi \| \vec{x}_{a,i} - \vec{x}_{a,j} \| \). Consequently, the interpolated value at \( \vec{x}_{b,k} \) is given by

\[
z(\vec{x}_{b,k}) = \sum_{j=1}^{m} \alpha_j \phi(\| \vec{x}_{b,k} - \vec{x}_{a,j} \|),
\]

or in matrix form

\[
z(\vec{x}_{b,k}) = \Phi_b \alpha,
\]

where \( \Phi_b \in \mathbb{R}^{1 \times m} \) is a row vector with elements \( \phi(\| \vec{x}_{b,k} - \vec{x}_{a,j} \|) \).

Combining Eq. (5) and Eq. (7), the interpolation is given by

\[
z(\vec{x}_{b,k}) = \Phi_b \Phi^{-1} z = c^k z.
\]

In this equation, the row vector \( c^k \) only depends on the point \( \vec{x}_{b,k} \) in grid \( b \) and the \( m \) closest points in grid \( a \) and hence the same coefficients are used for any variable \( z \) interpolated from grid \( a \) to the point \( \vec{x}_{b,k} \) on grid \( b \).

In summary, for each point \( \vec{x}_{b,k} \) on grid \( b \), three steps are followed to interpolate data. First, the \( m \) points on grid \( a \) closest to \( \vec{x}_{b,k} \) are identified. In the case of sharp corners or very close surfaces, care should be taken that no incorrect points are included in the selection. Then, the interpolation coefficients \( c^k \) are calculated and finally the actual interpolation is performed using Eq. (8). The first two steps only need to be done once, as the interpolation coefficients do not depend on the interpolated variable \( z \). Therefore the computational cost is very low. In this work, the coefficients are calculated with the initial grid position only. However, if required they can be recalculated after a certain number of coupling iterations.

Some of the surrogate models used in this work have one or more dimensions less in comparison with the actual solvers. In those cases, the data transfer between the actual solvers and the surrogate model requires the interpolation from a mesh with a higher number of dimensions to a mesh with a lower number of dimensions and vice versa.

The specific interpolation method to deal with this change in number of dimensions depends on the case at hand. Nevertheless, the general principle is always the same: by averaging in a certain direction, a dimension is removed, whereas by assuming constant values along a direction, a dimension can be added. In this work, these principles are used for an axisymmetric case, explained in detail in Section 5.

4. Quasi-Newton methods with surrogate models

This work discusses multiple types of surrogate models, all of which can be used within a general framework: the quasi-Newton method with approximation of the inverse Jacobian from a least-squares model and surrogate model (IQN-ILSM). This method uses a surrogate to provide both an initial guess in each time step as well as a Jacobian, which is used as additional information throughout the coupling iterations of a time step.

Before presenting this algorithm, the IQN-ILS method without reuse [11] is derived, be it in an alternative manner. This method is a typical quasi-Newton method that uses the secant information from input-output pairs of a black-box to approximate its Jacobian. More details on the surrogate models are provided at the end of this section.

4.1. A quasi-Newton technique without surrogate: IQN-ILS

The IQN-ILS algorithm starts from the displacement residual formulation, which reduces the FSI problem to finding the root of the non-linear system of equations \( \mathcal{R}(x) \), defined by Eq. (2). This root-finding problem can be solved by performing Newton-Raphson iterations as follows

\[
solve \; \mathcal{R}^k \Delta x^k = \Delta r^k = 0 - r^k \quad \text{for} \quad \Delta x^k
\]

from which

\[
x^{k+1} = x^k + \Delta x^k.
\]
In this equation, the superscript $k + 1$ denotes the current iteration within the current time step $n + 1$. As done here, the superscript $n + 1$ is omitted if no confusion is possible. The vector $\Delta r^k$ is defined as the difference between the desired residual, i.e. $0$, and the current residual $r^k$. Furthermore, the notation $\mathcal{R}^k$ refers to the Jacobian of $\mathcal{R}$ evaluated at $x^k$. This Jacobian is not known, as the solvers are considered black-boxes. Therefore, it is approximated, as will be explained further.

From here on, the notation $\partial^k_r x$ will be used to denote the approximation of the Jacobian $\mathcal{R}^k$. In this expression, the variable $r$ refers to the function from which the Jacobian is calculated and the subscript $x$ denotes the variable with respect to which the Jacobian is determined. The superscript $k$ indicates the iteration. Throughout the article, this notation will be used more generally to refer to the approximated Jacobian of different functions with respect to different variables.

Note that instead of approximating $\mathcal{R}'$, it is more appropriate to approximate the inverse Jacobian $\mathcal{R}^{-1}$ directly in order to avoid the solution of the linear system Eq. (9a). Consequently, the quasi-Newton iterations can be written as

$$x^{k+1} = x^k - \partial^k_r x r^k,$$

where, using the notation introduced above, $\partial^k_r x \in \mathbb{R}^{u \times u}$ denotes an approximation of the inverse of the Jacobian of $\mathcal{R}$, evaluated at $x^k$.

The approximation itself is based on input-output pairs from previous iterations. At the start of iteration $k + 1$, a number of residual vectors $r^i (i = 0, \ldots, k)$ and corresponding displacement vectors $x^i (i = 0, \ldots, k)$ from the current time step are available. From these vectors, $k$ differences are determined:

$$\delta r^i = r^{i+1} - r^i \quad \text{for} \quad i = 0, \ldots, k - 1,$$

$$\delta x^i = x^{i+1} - x^i \quad \text{for} \quad i = 0, \ldots, k - 1.\quad (11a)$$

These differences are stored as columns of the matrices $R^k$ and $X^k \in \mathbb{R}^{u \times k}$. The $\delta$ notation refers to the difference between consecutive iterations, in contrast to the $\Delta$ notation, which refers to the desired change in every iteration. Only for the input $x$, $\Delta x^k$ and $\delta x^k$ will be identical. Next, the approximated Jacobian is determined such that it fulfills the $k$-secant equations

$$\delta x^i = \partial^k_r x \delta r^i \quad \text{for} \quad i = 0, \ldots, k - 1\quad (12)$$

or in matrix form

$$X^k = \partial^k_r x R^k.\quad (13)$$

If $\mathcal{R}$ were linear, the inverse Jacobian would be constant and satisfy these equations exactly. Given these $k$-conditions, there is still some freedom in the definition of $\partial^k_r x$, since the number of columns $k$ is typically much smaller than the number of rows $u$. A reasonable choice is to impose a minimum norm requirement on the approximated Jacobian. If the columns of $\mathcal{R}^k$ are linearly independent, the resulting minimum norm solution is given by

$$\partial^k_r x = X^k (R^k)^T (R^k)^{-1} R^k \quad (14)$$

where $R^k$ is a shorthand notation for the pseudo-inverse of $R^k$, with the property $R^k R^k = I$, where $I$ is the identity matrix. In this way, the quasi-Newton update becomes

$$\Delta x^k = X^k R^k \Delta r^k.\quad (15)$$

However, with this Jacobian, the update $\Delta x^k$ would be a linear combination of the columns of $X^k$, which consist of previously evaluated differences $\delta x^i$. Then, updates outside of the vector space spanned by the columns of $X^k$ would remain unreachable. Moreover, the rank of $\partial^k_r x$ is limited to the number of columns $k$ of $R^k$. For the quasi-Newton iterations to work properly, the approximation $\partial^k_r x$ should be full rank.

Therefore, the vector $\Delta r^k$ is split in a part $R^k \Delta r^k$ within the column space of $R^k$ and a part $(I - R^k R^k)$ of $r^k$ orthogonal to this vector space, with the use of the orthogonal projector $R^k R^k$, as illustrated in...
The vector $\Delta r^k$ is split in part within the column space of $R^k$ and a part orthogonal to it, using the orthogonal projector $R^k R^{k+}$. Figure 1. This matrix projects an arbitrary vector onto the column space or range of $R^k$, such that the Euclidean distance between the original vector and its projection is minimal.

That this is indeed the case can be seen by trying to approximate the vector $\Delta r^k$ as a linear combination of the columns of $R^k$: $\Delta r^k \approx R^k c^k$. The minimum norm least-squares solution to this problem is given by $c^k = R^k R^{k+} \Delta r^k$, such that the closest approximation of $\Delta r^k$ within the column space of $R^k$ is given by $R^k c^k$ or $R^k R^{k+} \Delta r^k$. Note that the summation of both parts indeed retrieves the vector $\Delta r^k$ and that their scalar product equals zero. For a more elaborate and rigorous discussion of orthogonal projectors refer to [28].

For the first part of the residual $\Delta r^k$, within the column space of $R^k$, information has been gathered in the preceding iterations and Eq. (15) is used. However, for the second part, orthogonal to the range of $R^k$, no information is available. This is verified by observing that the product of the purely secant-based Jacobian $X^k R^{k+}$ with this part equals zero. Therefore, it is not possible to reduce the part of the residual $\Delta r^k$ orthogonal to the span of the already obtained differences $\delta r^i$, when using Eq. (15) alone. To overcome this and to obtain a Jacobian that is full rank, simple Gauss-Seidel iterations are performed for the second part. A Gauss-Seidel iteration corresponds to taking a step equal to the residual $\Delta x^k = r^k = -\Delta r^k$. In other words, an inverse Jacobian equal to $-I$ is used. Finally, this results in the quasi-Newton step

$$\Delta x^k = X^k R^{k+} \left[ R^k R^{k+} \Delta r^k \right] - I \left[ (I - R^k R^{k+}) \Delta r^k \right]$$

(16a)

$$= X^k R^{k+} \Delta r^k - (I - R^k R^{k+}) \Delta r^k,$$

(16b)

in which the approximated Jacobian is given by

$$\delta r^k x = X^k R^{k+} - (I - R^k R^{k+}).$$

(17)

The number of columns can be different from $k$, if filtering is applied (removal of almost linearly dependent columns) or if reuse is performed (addition of columns from previous time steps).

If columns are linearly dependent, filtering has to be applied. This will be explained further in this section.
Furthermore, if a third matrix $\tilde{X}^k$ is defined, containing the corresponding differences of the output $\tilde{x}$ of the structure solver,\\
\[ \delta \tilde{x}^k = \tilde{x}^{k+1} - \tilde{x}^k \quad \text{for} \quad i = 0, \ldots, k - 1, \]

as columns, then it follows from Eq. (2) that $\tilde{X}^k = R^k + X^k$. Combined with Eqs. (16), the expression for a quasi-Newton step in IQN-ILS is retrieved.

Both Eqs. (16) and Eq. (19) result in the same iteration update given a residual $\Delta r^k$; they are only different in the way they are derived. The former uses $\partial^k_i x$, which is the approximation of the inverse of the Jacobian of $R$ with respect to $x$. As described above this approximation consists of a secant term and a Gauss-Seidel term. In contrast, the latter uses $\delta^k_i \tilde{x}$, which is the approximation of the inverse of the Jacobian of $R$ with respect to $\tilde{x}$ and is given by
\[ \delta^k_i \tilde{x} = \tilde{X}^k R^k. \]

The relation between the two is given by
\[ \partial^k_i x = \delta^k_i \tilde{x} - \delta^k_i r = \partial^k_i \tilde{x} - I. \]

Hence, the identity matrix has to be subtracted from $\partial^k_i \tilde{x}$ in order to obtain an approximation of the inverse of the Jacobian with respect to $x$.

As discussed above, for $\partial^k_i x$, the application of Gauss-Seidel iterations for the part orthogonal to the columns of $R^k$ corresponds to using an inverse Jacobian equal to $-I$. However, for the second approach, the inverse Jacobian corresponding to Gauss-Seidel iterations is equal to $I$, as a Gauss-Seidel step, $\Delta x^k = r^k = -\Delta r^k$, corresponds to a step $\Delta \tilde{x}^k = \Delta x^k + \Delta r^k = 0$. This observation will greatly simplify the expressions in the next section.

Furthermore, it is important to emphasize that Eqs. (16) or Eq. (19) are only symbolic representations of the IQN-ILS procedure and should never be implemented as such. In practice, the minimum norm problem in Eq. (14) is solved using the economy-size QR-decomposition of $R^k$

\[ R^k = Q_R^k R_R^k, \]

where $Q_R^k \in \mathbb{R}^{u \times k}$ is a matrix with orthonormal columns and $R_R^k \in \mathbb{R}^{k \times k}$ an upper triangular matrix. The resulting quasi-Newton update becomes

\[ \Delta x^k = X^k R_R^{-1} Q_R^k T \Delta r^k - (I - Q_R^k Q_R^k T) \Delta r^k = \tilde{X}^k R_R^{-1} Q_R^k T \Delta x^k - \Delta r^k, \]

where Eq. (23a) employs the approximate Jacobian with respect to $x$ and Eq. (23b) the approximate Jacobian with respect to $\tilde{x}$. Note that with the QR-decomposition, the orthogonal projector $R_R^{-1} Q_R R_R^{-1} Q_R T$ is given by $Q_R^k Q_R^k T$. Moreover, matrices of the form $\tilde{X}^k R_R^{-1} Q_R^k T$ and $Q_R^k Q_R^k T$ should never be calculated explicitly, as only the product with a vector needs to be determined. In this way, matrix-matrix products and the construction of large dense square matrices are avoided. Further, as $R_R^k$ is an upper triangular matrix, the inverse $R_R^{-1}$ does not have to be calculated explicitly either, as the corresponding triangular system can be solved using back-substitutions. The complexity of the resulting algorithm is determined by the QR-decomposition and is linear with respect to the number of degrees of freedom $u$. In the following, the notations with the pseudo-inverse will be used rather than the implementation with the QR-decomposition for the sake of brevity and clarity, but it is essential to keep in mind that those notations are only a symbolic representation.

When the number of columns in the matrix $R^k$ grows, some columns might become (almost) linearly dependent. The corresponding diagonal element in $R_R^k$ will then be (almost) zero and the triangular system cannot be solved. To avoid this, filtering can be applied, which removes older linearly dependent columns from $X^k$, $R^k$ and $\tilde{X}^k$ [29, 30]. Likewise, in problems with a small number of degrees of freedom, the oldest columns can be removed to keep the number of columns smaller than the number of rows $u$. 

8
Finally, note that during the first iteration \((k = 0)\), no differences can be determined. Therefore, there is no secant information to fulfill and a (relaxed) Gauss-Seidel iteration is performed. When using IQN-ILS for a transient problem, each time step starts from a prediction based on previous time steps. For the very first time step an initial guess is made by the user, typically zero deformation.

### 4.2. Addition of a surrogate: IQN-ILSM

The quasi-Newton algorithm described above is a completely black-box approach: it assumes nothing is known a priori about the solvers or the complete coupled system. However, in many cases, there is some information readily available or can straightforwardly be obtained. This information can take many forms, but is denoted here with the general term surrogate model. Before taking a closer look to different types of surrogate models, this section elaborates on how they can be used. The benefit of incorporating a surrogate model in the coupling algorithm is twofold: it can offer both an approximated solution as well as an additional approximation of the Jacobian. This additional Jacobian is used throughout the coupling iterations of a time step, with diminishing importance as more secant information becomes available. Both the approximate solution and the Jacobian may accelerate the convergence and increase the stability of the coupling.

The first improvement is the inclusion of a surrogate Jacobian to complement the secant information gathered throughout convergence. In the original quasi-Newton algorithm, the Jacobian is constructed based on input-output pairs, as explained previously. For the part of \(\Delta r^k\) that is orthogonal to the column space of \(\hat{R}^k\), it is assumed that no derivative information is known and \(-J\) is employed as inverse Jacobian. Using a surrogate Jacobian instead can improve stability and accelerate convergence. Moreover, it avoids the need for an initial (relaxed) Gauss-Seidel iteration.

Second, most surrogate models also provide an approximation of the solution or surrogate prediction, which can be used as initial guess. This is especially important for the initial value of a steady simulation or for the first time step of a transient calculation. Nevertheless, also in every other time step of a transient simulation, the use of a surrogate model may provide a better prediction than the pure extrapolation of the solution from previous time steps.

Algorithm 1 shows the procedure in detail. In the most general form, the surrogate model may be denoted by a surrogate residual operator \(\mathcal{R}_s\), which represents a surrogate FSI problem analogous to \(\mathcal{R}\). The solution \(x_s\) of \(\mathcal{R}_s(x_s) = 0\) is an approximation of the actual solution \(x\) of \(\mathcal{R}(x) = 0\). At the start of each time step, the surrogate FSI problem is solved and its solution is used to determine the initial value of the current time step \(x^{n+1,0}\). Instead of applying the surrogate solution directly, the change of the surrogate solution with respect to the previous time step \(n\) is added to the actual solution of the previous time step \(x^n\). This decision is based on the expectation that the change in solution between two time steps is better approximated by the surrogate model than the actual solution itself. Indeed, this approach proved to be moderately more potent. In Algorithm 1, the time step indication \(n + 1\) has been dropped for brevity.

Next, on line 3, the inverse Jacobian of \(\mathcal{R}_s\) with respect to \(\hat{x}\) is determined. In the remainder of this work, it will be simply referred to as the surrogate Jacobian. As it is sufficient to calculate its product with a vector, it does not need to be an explicit matrix and a more general linear expression is sufficient. To emphasize this, the surrogate Jacobian will be represented by a function \(\partial_s \hat{x}_s(\cdot)\). Furthermore, its value may be exact, a full-rank approximation or only a low-rank approximation. Lastly, note that the surrogate Jacobian is considered fixed throughout a time step.

All steps up to here, including the evaluation of the surrogate model on line 2 and the determination of the surrogate Jacobian on line 3, are additional steps compared to an existing quasi-Newton technique, such as IQN-ILS. Therefore, it is imperative that their cost remains sufficiently low, such that the obtained speed-up outweighs the additional cost. Note that these steps are done only once per time step.

Conversely, the next steps are done in every iteration. For instance, on line 15, the iteration update \(\Delta x^k = x^{k+1} - x^k\) is determined with contributions of both the secant Jacobian and the surrogate Jacobian. The methodology is clearly depicted by Eq. (24a), where the Jacobians with respect to \(x\) are used. The residual \(\Delta r^k = 0 - r^k\) is split in a part \(R^k R^{k+1} \Delta r^k\) within the column space of the secant Jacobian and a part \((I - R^k R^{k+1}) \Delta r^k\) orthogonal to it, as explained in Section 4.1. The key idea is to use the surrogate Jacobian for this last part, instead of using Gauss-Seidel iterations.

In the most general case, however, the surrogate Jacobian is only a low-rank approximation, such that the resulting total Jacobian may again be defective. Therefore, \((I - R^k R^{k+1}) \Delta r^k\) is split once more in two parts similar to the
Algorithm 1 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).

1: \( k = 0 \)
2: solve \( \mathcal{R}_s(x_s) = 0 \)
3: determine \( \partial_r \tilde{x}_s(\cdot) \)
4: \( x^0 = x^n + (x_s - x^n) \)
5: \( r^0 = \mathcal{R}(x^0) \)
6: while \( \| r^k \| > \epsilon \) do
7: \( k = 0 \) if \( r^k = 0 \) then
8: \( x^{k+1} = x^k - \partial_r \tilde{x}_s(r^k) + r^k \)
9: else
10: construct \( \mathcal{R}_k^k \) and \( \tilde{X}^k \)
11: calculate QR-decomposition \( \tilde{R} = Q\tilde{R}_k \)
12: \( b^k = Q^T_k r^k \)
13: solve \( \tilde{R}_k c^k = b^k \)
14: \( r'^k = r^k - Q_k b^k \)
15: \( x^{k+1} = x^k + \tilde{X}^k c^k - \partial_r \tilde{x}_s(r'^k) + r^k \)
16: end if
17: \( r^{k+1} = \mathcal{R}(x^{k+1}) \)
18: \( k = k + 1 \)
19: end while
20: synchronize \( \mathcal{R}_s \) with \( \mathcal{R} \)

procedure from Section 4.1 and Gauss-Seidel iterations are performed for the part that is outside of the column space of the surrogate Jacobian. This part is written as \((I - R_s R_s^+) (I - \tilde{R}^k \tilde{R}^k) \Delta r^k\) in analogy with previous section. Just as before, the matrix \( R_s R_s^+ \) is an orthogonal projector onto the columns space of the surrogate Jacobian. As a result \( \Delta x^k \) is composed out of three terms, as illustrated in Figure 2. In Eq. (24a), the different parts of \( \Delta r^k \) were written in full for clarity.

\[
\Delta x^k = \tilde{X}^k \tilde{R}^k + \partial_r \tilde{x}_s(\cdot) (\tilde{R}^k \tilde{R}^k + \Delta r^k)
\]

\[
\Delta x^k = \tilde{X}^k \tilde{R}^k + \partial_r \tilde{x}_s(\cdot) (I - \tilde{R}^k \tilde{R}^k + \Delta r^k)
\]

This equation can be simplified to Eq. (24b) by using the Jacobians with respect to \( \tilde{x} \) instead. In that case, the Gauss-Seidel iterations for the last part do not require a separate term or, stated differently, they are included in the last term. The implementation in Algorithm 1 also uses this equation, but with the QR-decomposition. In this way, it demonstrates the practical implementation of the algorithm. Because there is no need for the explicit construction of large square matrices or matrix-matrix multiplication, the computational cost and memory requirements are linear with respect to the number of degrees of freedom \( u \).

Note that the secant Jacobian has priority over the surrogate Jacobian. In other words, as the calculation continues, the part of the residual in the range of the secant Jacobian will increase, and by consequence, so will the contribution of the secant term, which uses the actual information from previous iterations. On the other hand, the role of the surrogate Jacobian will diminish while convergence is being reached. This is in fact desirable, as the surrogate model inherently contains some error, which becomes more important close to the final solution. In the first iteration on line 8, there are no secant equations to fulfill, such that the next step is calculated entirely using the surrogate Jacobian \( \partial_r \tilde{x}_s(\cdot) \).

Finally, at the end of a time step, on line 20, the surrogate model is synchronized with the actual solvers. This
Figure 2: The vector $\Delta r^k$ is split in two parts with orthogonal projector $R^k R^k R^k$. The part orthogonal to the column space of $R^k$ is divided again in two parts with the orthogonal projector $R_s R_s$. The pure secant Jacobian $X_k R^k$ is used for the first part (1), the surrogate Jacobian $\partial_r x_s(\cdot)$ is used for the second part (2) and Gauss-Seidel iterations are used for the last part (3).

means that all internal variables of the surrogate model are updated with the correct solution from the actual solvers. In case $R_s$ is built up from separate flow and structure solvers, this synchronization can be done by setting all the variables in the entire flow and structure domain to their values in the actual solution through interpolation. However, if the solvers are black-boxes, this is clearly not possible. A more general approach to achieve synchronization in the case of a fully black-box surrogate is solving the surrogate residual operator $R_s$ once more using the final solution of the actual solvers $x$. In other words, $R_s(x)$ is calculated. The cost of this synchronization step is small, as it consists of only one surrogate evaluation. Synchronization may be especially important in transient simulations with high number of time steps and large deformations, because the surrogate and actual coupled system risk to diverge. Nevertheless, in other cases, it can also have a positive influence on performance, which outweighs its small cost. The exact effect of synchronization will be problem dependent.

The flow chart on the next page illustrates the procedure. In the figure, the surrogate Jacobian is determined using IQN-ILS, however another technique for this step is equally suitable as long as it allows for the extraction of an approximated Jacobian. As before, the superscript $n + 1$ denotes the current time step, while $n$ and $n - 1$ denote the two previous time steps. Once more, the time step indication $n + 1$ has been dropped for brevity. The different shades of gray indicate the different steps of a single time step: first the surrogate calculation, then the calculation with the actual solvers and finally the synchronization. Note that the surrogate part is performed only once per time step.

Despite the apparent effortlessness with which the algorithm above performs mathematical operations between variables on the mesh of $R_s$ and $R$, these meshes typically differ, possible even with a different number of dimensions, and therefore interpolation is required, as explained in Section 3. As will be explained in the following paragraphs, not only the surrogate and actual solutions need to be interpolated, also the use of derivative information from the surrogate model requires interpolation.

Although it is not necessary, the surrogate model $R_s$ itself may consist of the combination of a surrogate flow solver $F_s$ and a surrogate structure solver $S_s$. Then, $R_s$ is related to these surrogate solvers by

$$R_s(x_s) = S_s \circ F_s(x_s) - x_s.$$  (25)

If the grids used by these solvers are not identical, calculating the surrogate solution requires the interpolation between the meshes of $F_s$ and $S_s$. Generally, interpolation is required between three sets of grids in three locations, as shown in Figure 4.

First, displacement and load data are interpolated between the grids of the actual flow and structure solver. This occurs within the residual operator $R$. Typically one of the grids will serve as reference grid, to which the data of the other grid is transferred. In Figure 4, the fluid grid is chosen as reference. This grid is then also used for the input and output of the residual operator $R$.

Second, displacement and load data are interpolated between the grids of the surrogate flow and structure solver. This is done within the surrogate residual operator $R_s$ and is completely analogous to the interpolation within $R$. 

11
Figure 3: Graphical representation of the procedure of IQN-ILSM. The surrogate calculation is performed using IQN-ILS. The solvers and vectors corresponding to the surrogate model $R_s$ are denoted with a subscript $s$.

**surrogate**

1. $k = 0$
2. Linear prediction $x_s^k = x_s^0 + (x_s^0 - x_s^{k-1})$
3. $r_s = R_s(x_s)$
4. Surrogate flow solver $\hat{y}_s = F_s(x_s)$
5. $y_s = \hat{y}_s$
6. Surrogate structure solver $\hat{x}_s = S_s(y_s)$
7. $r_s = \hat{x}_s - x_s$

**actual solvers**

1. $k = 0$
2. Surrogate prediction $x^k = x^0 + (x_s - x_s^0)$
3. $r^k = R(x^k)$
4. $\hat{y} = F(x)$
5. $y = \hat{y}$
6. Structure solver $\hat{x} = S(y)$
7. $r = \hat{x} - x$

**synchronization**

1. Surrogate synchronization $R_s(x)$

**end time step $n + 1$**

$n = n + 1$
Figure 4: Graphical representation of the different grids and locations of interpolation. In this example the flow solver grid serves as reference for the actual and surrogate solvers. The solvers and vectors corresponding to the surrogate model $\mathcal{R}_s$ are denoted with a subscript $s$. 
Third, displacement and residual data are interpolated between the residual operators $\mathcal{R}$ and $\mathcal{R}_s$. As the residual is calculated as a difference between two displacements, residual and displacement data are discretized identically. However, the grids used in $\mathcal{R}$ and $\mathcal{R}_s$ typically differ and hence data transfer between them requires interpolation. The discretization used in $\mathcal{R}$ serves as reference and the in- and output of $\mathcal{R}_s$ are interpolated from and to that reference grid. At the start of each time step, the surrogate solution $x_s$ is interpolated to the grid of $\mathcal{R}_s$ to perform a prediction, whereas at the end of each time step, the final solution $x$ is interpolated back to the grid of $\mathcal{R}_s$ for synchronization purposes. Further, the derivative information $\partial_r \tilde{x}_s(\cdot)$ on the surrogate grid, needs to be used on the grid of $\mathcal{R}$. This is achieved by interpolating its argument $\Delta r$ to the surrogate grid, and interpolating the result $\Delta x_s = \partial_r \tilde{x}_s(\Delta r)$ obtained on this surrogate grid, back to the reference grid. In this way, only the second term in Eq. (24b) is interpolated from the surrogate grid. All other operations in the coupling loop are performed on the grid of $\mathcal{R}$. Any loss of information incurred from going through the surrogate grid is captured by the third term $-\Delta r^k$, which assures that for any part of the residual that is not treated by any of the two previous terms, Gauss-Seidel iterations are performed. Failure to take all of the residual $\Delta r^k$ into account results in a non-vanishing error. For example, the calculation of the iteration update on a coarse grid will result in a non-vanishing high-frequency error on the fine grid, as was observed by Scholz et al. [21].

As a final note, the solution of $\mathcal{R}_s(x_s)$ on line 2 can be achieved with any type of method, both monolithic and partitioned. However, the determination of a surrogate Jacobian should be possible. In that aspect, a partitioned quasi-Newton technique has the benefit that it automatically provides a low-rank Jacobian. In IQN-ILS, for example, the low-rank surrogate Jacobian is given by $X_s R_s^+$, where the subscript $s$ is used to refer the surrogate model. Note that the solution technique for the surrogate has its own extrapolation method and convergence criteria.

Furthermore, the surrogate can even be solved with IQN-ILS, in other words, with the help of another surrogate model. In this way, a nesting of surrogate models is achieved. This does not only accelerate solving of the surrogate model, but may provide a more complete surrogate Jacobian to use for the coupling of the actual solvers. For example, if the surrogate model $\mathcal{R}_s$ is solved with another surrogate model $\mathcal{R}_{ss}$, the iteration update for the actual solvers would be given by

$$\Delta x^k = \tilde{X}^k R^{k+} \Delta r^k + \partial_r \tilde{x}_s \left( (I - R^k R^{k+}) \Delta r^k \right) + \partial_r \tilde{x}_{ss} \left( (I - R_s R_s^+) (I - R^k R^{k+}) \Delta r^k \right) - \Delta r^k,$$

(26)

where $\partial_r \tilde{x}_{ss}(\cdot)$ is the surrogate Jacobian of $\mathcal{R}_{ss}$. The first term in this equation represents the contribution from the secant Jacobian, the second and third term are the contribution from the first and second surrogate model, respectively, and the application of Gauss-Seidel iterations for the remainder of the residual is included in the last term. The nesting of surrogate models is straightforwardly extended to even more surrogate models.

Now that the general framework has been presented, the remainder of this section will dive deeper into the different types of surrogates that can be used inside this framework. The inclusion of surrogate information results in different methods depending on the type of surrogate model. Some of these methods already exist, others are improvements to existing methods and still others are new.

### 4.3. Coarsened solvers as surrogate model

A first option is a surrogate residual operator $\mathcal{R}_s$ that consists of two surrogate solvers as in Eq. (25), where $\mathcal{F}_s$ and $\mathcal{S}_s$ are identical to the actual solvers, but defined on a coarser grid. The root-finding problem of the surrogate residual operator $\mathcal{R}_s$ is solved using IQN-ILS, such that the surrogate Jacobian is given by $\tilde{X}_s R^s$, resulting in

$$\partial_r x^k = \tilde{X}^k R^{k+} + \tilde{X}_s R^s (I - R^k R^{k+}) - I.$$

(27)

This idea is not new, but has been introduced by Degroote and Vierendeels as the multi-level concept [20]. In essence, the modes defined on coarser grid levels are used to speed-up the calculations on finer grid levels. This methodology is supported by the fact that for incompressible FSI with high added-mass the unstable modes correspond to the error components with the lowest wavenumber, which can be resolved on a coarser grid. However, there are three important differences between the presented framework in this work and the implementation by Degroote and Vierendeels.

First of all, the surrogate derivative information is treated differently. In the algorithm by Degroote and Vierendeels, the differences, such as $\delta r^i$, obtained on different grid levels are all stored in one common matrix. However, an important note is that this matrix does not contain differences between the displacements or residuals on different
grid levels, because these are not a pure representation of the relation between residual and displacement, but are influenced by additional features becoming visible on the finer grid. Nevertheless, by storing all differences in a single matrix, the same importance is given to each difference. By contrast, in the new method separate matrices are constructed for each level: there are matrices for the actual residual operator and matrices for the surrogate residual operators. This means that priority can be given to the information on the finer grid level, which is not only more accurate because of the finer mesh, but also due to the fact that they are obtained closer to the final solution and are, therefore, more appropriate for the approximation of the inverse Jacobian of the non-linear equations. The finest grid in the approach of Degroote and Vierendeels corresponds to the residual operator $R$. The use of one surrogate model $R_s$ corresponds to their multi-level approach with two grid levels. As illustrated by Eq. (26), more than two grid levels in the multi-level approach are naturally obtained by using multiple surrogate models that are increasingly more coarse. As before, the algorithm starts on the coarser grids, but will give priority to the information on the finer grids when it becomes available.

The second difference with the multi-level method by Degroote and Vierendeels is the absence of a universal coupling grid. Its necessity disappears as differences are stored for each grid level separately. In this way, the information obtained on the coarser grids does not have to be stored in the same matrix as the differences on the finest grid and can be stored in a much smaller matrix. Furthermore, in the multi-level algorithm, loads and displacements are interpolated between the solver and the universal coupling grid, on which all coupling iterations are performed. Even the coupling between two solvers on a coarser grid is performed on this coupling grid. In the new method only differences in residuals and displacements are interpolated between the grid levels. Therefore, within a residual operator, no additional interpolation step via a coupling grid is required.

Lastly, the determination of the initial guess at the start of a time step is done differently. The multi-level algorithm by Degroote and Vierendeels applies the surrogate solution directly on the finer grid, the new IQN-ILSM method determines the change in the solution of the coarser grid level with respect to the previous time step and uses this change to determine the value on the new grid, as was explained in Section 4.2.

4.4. Simplified physics as surrogate model

Second, the surrogate residual operator $R_s$ may consist of two surrogate solvers as in Eq. (25), where $F_s$ and $S_s$ omit a part of the physics and are therefore easier to solve. Again, the root-finding problem of the surrogate residual operator $R_s$ is solved using IQN-ILS, resulting in Eq. (27). This type of surrogate model is very close to the previous one, but due to the fact that the difference between actual and surrogate solvers are no longer just the mesh resolution, the surrogate solution $x_s$ and derivative information $\partial_r \tilde{x}_s$ will typically be somewhat less accurate. On the other hand, the cost reduction may be significantly higher. Examples include solvers that neglect features such as viscosity for a flow solver or non-linear material behavior for a structure solver. Another important group of examples are solvers that are reduced in number of dimensions. In fact, many simulations in 3D have physics that are predominantly 2D or even 1D, e.g. wave propagation problems. Therefore, those problems can benefit from a 1D or 2D surrogate model.

As an illustration, this work will consider an example from the last subcategory in Section 6. Although these lower-dimensional solvers do not cover all physics, they can certainly provide useful data, especially when the physical phenomena occur predominantly in one dimension, while being cheap to evaluate, as the number of degrees of freedom is orders of magnitude smaller. An advantage of this type of surrogate model is that they can reduce the number of degrees of freedom significantly, without changing the resolution in the predominant flow direction, and therefore are still able to capture the physics very well.

4.5. Simplified physics as surrogate model with known Jacobian

Thirdly, a surrogate residual operator $R_s$ can be used for which the Jacobian $\partial_r \tilde{x}_s$ is known exactly. This surrogate model will typically be an analytical model, with simplified assumptions similar to the previous category. If the analytical model is linear, the solution is found directly. But also in the case of a non-linear analytical surrogate model, the solution of $R_s$ is reached quickly by performing a small number of Newton-Raphson iterations with the exact Jacobian, determined in each iteration. Constructing a Jacobian based on secant information is hence not required. The downside is that the surrogate Jacobian will typically be constructed explicitly. Therefore, this method is only suited for surrogate solvers with a lower number of degrees of freedom.
Possibly, only the Jacobians for the separate solvers \( \mathcal{F}_s \) and \( \mathcal{S}_s \) are known. The surrogate Jacobian can then be determined as follows:

\[
\partial_{r} \tilde{x}_s = \partial_{r} x_s + I = (\partial_{x_s} \tilde{x}_s - I)^{-1} + I = (\mathcal{S}_s'(x_s)) \mathcal{F}_s'(x_s) - I \right)^{-1} + I.
\]

Here \( \mathcal{F}_s'(x_s) \) and \( \mathcal{S}_s'(y_s) \) represent the Jacobians of the surrogate flow and structure solver, respectively. Note that \( \partial_{x} \tilde{x}_s \) is full rank on the surrogate grid. Nevertheless, if the surrogate grid is coarser with respect to the grid of \( \mathcal{R} \), there is still a part of the residual for which Gauss-Seidel iterations have to be performed.

Once \( x_s \) is found, the analytically determined surrogate Jacobian \( \partial_{r} \tilde{x}_s \) is kept constant throughout the time step for the determination of the iteration update in the main loop. Alternatively, one could argue for an update of the surrogate Jacobian in each iteration of the IQN-ILSM algorithm. However, the benefit is expected to be small, because the surrogate Jacobian is only an approximation of the actual inverse Jacobian of \( \mathcal{R} \), and because the surrogate will typically only change slightly, as the solution \( x_s \) is considered to be quite close to the final solution. Moreover, this would require additional interpolations of \( x^k \) to the surrogate mesh and back.

### 4.6. Reuse of previous time steps

The final and fourth example is only applicable in a transient FSI calculation, where there is another source of profitable information that can be used as surrogate model, namely the previous time steps. The largest benefit is that this type of surrogate model is essentially free, because lines 2 to 3 in the algorithm correspond to the solution of the previous time step and do not represent an additional cost. In that case, the surrogate solution in the current time step \( x^{n+1}_s \) equals the solution in the previous time step \( x^n \) and the surrogate Jacobian from the current time step \( \partial_{r} x^{n+1}_s(\cdot) \) is given by the approximation of the inverse Jacobian from the previous time step \( \partial_{r} x^n(\cdot) \). Further, no synchronization is required, as the surrogate model consists of previous time steps, and the prediction on line 4 corresponds to linear extrapolation.

The reuse of \( q \) of the \( n \) previous time steps corresponds to the use of \( q \) nested surrogate models.\(^3\) Rewriting Eq. (26) towards this objective results in

\[
\partial_{r} x^{n+1,k} = \overline{X}^{n+1,k} R^{n+1,k} + \sum_{i=n+1-q}^{n} \overline{X}^i R^i \left( \prod_{j=i+1}^{n} (I - R^j R^j) \right) (I - R^{n+1,k} R^{n+1,k})
\]

where the matrices \( \overline{X}^i \) and \( R^i \) correspond to the calculation of time step \( i \) and, as before, time step \( n+1 \) is the current time step.

The idea to reuse information from previous time steps is not new. It has been applied to IQN-ILS by Degroote et al. [11] by including the differences from previous time steps in the matrices \( R^{n+1} \) and \( \overline{X}^{n+1} \) of the current time step. This requires the introduction of a reuse parameter \( q \) denoting the number of time steps to reuse, for which the optimal value is problem dependent. If the number of reused time steps is too low, useful information might be lost, whereas a too high value leads to contradicting and irrelevant data. However, the application of filtering [29, 30] may resolve this problem, see the explanation on IQN-ILS in Section 4.1.

Later followed IQN-MVJ [12, 31], which does not require a reuse parameter. In this last quasi-Newton method, the explicitly constructed Jacobian from the previous time step \( \partial_{r} x^n \) is used according to Eq. (30a).

\[
\partial_{r} x = \partial_{r} x^n + (\overline{X}^k - \partial_{r} x^n R^k) R^k
\]

Eq. (30b) rewrites the same equation in terms of the Jacobian with respect to \( x \), such that the comparison with the IQN-ILSM method is easily made. The largest downside of this IQN-MVJ method is the explicit matrix construction,

\(^3\)The value of this parameter is not troublesome, since a larger value generally results in a better convergence, as will be explained further.
which results in a quadratic complexity in both operations and storage. Scheufele and Mehl [13] presented a matrix-free alternative, by applying a restart after a certain number of time steps involving a singular value decomposition. However, this method again requires reuse parameters.

Conversely, the IQN-ILSM method with reuse surrogate model described here has linear complexity, as will be shown below, and requires no problem-dependent reuse parameters. In fact, it turns out to be identical to the recently introduced algorithm by Spenke et al. [14], called interface quasi-Newton implicit multi-vector least-squares method (IQN-IMVLS). This method is another matrix-free variant of IQN-MVJ in which the inverse Jacobian is determined recursively, as in Eq. (29). If the iteration superscript $k$ is omitted, this equation can be condensed to

$$\partial \bar{r}^{n+1} x = \sum_{i=n+1-q}^{n+1} \tilde{X}^i R_i^{+} \prod_{j=i+1}^{n+1} (I - R_i^j R_i^{+}) - I. \quad (31)$$

Besides the linear complexity, it has the advantage that the contribution of each previous time step is contained within one separate term. Hence, it is possible to only include information from a limited number of previous time steps $q$, in this way limiting the cost of calculating the product of the Jacobian with a vector.

The linear cost with respect to the number of time steps of the product of Eq. (31) with a vector $\Delta R^k$ is the result of an efficient implementation. Instead of recalculating the product $\prod_{j=i+1}^{n+1} (I - R_i^j R_i^{+}) \Delta R^k$ for each reused time step, it suffices to update it with one additional factor. The procedure is illustrated by Algorithm 2, which evaluates the expression $\partial \bar{r} x_s(r^k)$ on line 15 of Algorithm 1 in the case of a reuse surrogate model.

**Algorithm 2** An efficient procedure for the evaluation of $\Delta x_s = \partial \bar{r} x_s(r^k)$ on line 15 in Algorithm 1 in the case of a reuse surrogate model.

1: $\tilde{X}^i, Q_i^R, R_i^R$ for time step $i = n, \ldots, n+1-q$ are available
2: for $i = n, \ldots, n+1-q$ do
3: \hspace{0.5cm} $b^i = Q_i^R r^k$
4: \hspace{0.5cm} solve $R_i^T e^i = -b^i$
5: \hspace{0.5cm} $\Delta x_s = \Delta x_s + \tilde{X}^i e^i$
6: \hspace{0.5cm} $r^k = r^k - Q_i^R b^i$
7: end for

Limiting reuse to a certain number of time steps is very similar to the approach of Degroote et al. However, their procedure treats all reuse information identically. As a result, appropriate filtering is required to remove contradicting information. With recursive reuse of previous time steps, this problem is avoided. The surrogate model approach clearly shows how information from more relevant, closer, time steps is favored over data from older time steps: the information from older time steps is only used for the part of the residual for which there is no other information available.

Furthermore, both methods might suffer from the inclusion of information from previous time steps, which is no longer relevant. However, the effect on the recursively determined Jacobian will be much smaller because the newer time steps are prioritized. Therefore, the performance generally increases if data from more time steps are reused [14], whereas the approach of Degroote et al. is only robust with respect to the reuse parameter if filtering is applied.

Moreover, with reuse, the cost of calculating the iteration update is higher for IQN-ILS. The cost is determined by the reduced QR-decomposition, which has a complexity of $O(u(kq)^2)$, where $k$ is the average number of iterations per time step. Additionally, if filtering is applied, the QR-decomposition is typically recalculated several times, driving up the cost. In the IQN-ILSM method with reuse, a QR-decomposition is also performed for the determination of the pseudo-inverse $R^+$. However, the cost is limited to $O(u(kq)^2)$, as it is calculated only once per time step and subsequently stored. The remaining steps in the Algorithm consist of matrix-vector products with $O(u(kq))$, repeated $q$ times. If $q > k$, the final complexity of IQN-ILSM is only $O(u(kq))$. Admittedly, one may argue that the importance of this cost is quite limited, as it remains small compared to the cost of the solvers themselves.

Finally, instead of determining the number of reused time steps $q$ arbitrarily, the IQN-ILSM method with reuse surrogate offers the possibility to choose this value dynamically. For instance, one may halt the inclusion of previous
time steps once the norm of the remainder of $\Delta r^k$ becomes lower than a certain tolerance $\epsilon_r$, such that $q$ is the smallest integer for which

$$\left\| \prod_{j=n+1-q}^{n+1} \left( I - R^i R^j \right) \Delta r^k \right\| < \epsilon_r \left\| \Delta r^k \right\|.$$  

(32)

From this point onward, the change to $\partial_r^{n+1,k} x$, resulting from the inclusion of more time steps, becomes insignificant.

5. Case setup and models

The different types of surrogate models are tested on two example cases. The first is a classic FSI test case that considers the incompressible flow through a flexible tube, as illustrated in Figure 5. The adopted parameters are given in Table 1. The tube has length $\ell$ and radius $r_0$. Further, the tube wall is described by its thickness $h$, structural density $\rho_s$, modulus of elasticity $E$ and Poisson ratio $\nu$, and the fluid is identified by its density $\rho_f$ and dynamic viscosity $\mu$.

The tube is used for transient as well as steady simulations. The transient case considers a pressure increase applied at one end, resulting in a pressure pulse that travels through the tube. The steady case increases the pressure at both ends leading to an inflation of the tube. In both cases, the fluid is free to flow in and out of the tube at both ends and the tube wall is clamped at its extremities.

The second example is the so-called breaking dam test case, which simulates the deflection of a top-fixed flexible gate due to the variable pressure imposed by a liquid column, initially at rest and located behind the gate in a free-surface tank. Figure 6 provides a schematic representation. This case has been examined numerically and experimentally by Antoci et al. [32]. The liquid under consideration is water with density $\rho_f$ and dynamic viscosity $\mu_f$. The flexible gate is made of rubber with density $\rho_s$, modulus of elasticity $E$ and Poisson ratio $\nu$. The gas phase is
Figure 6: Schematic representation of the breaking dam test case. The clearance beneath the flexible gate has been exaggerated for clarity.

Table 2: Parameter values for the breaking dam case.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>0.1 m</td>
<td>$H$</td>
<td>0.14 m</td>
</tr>
<tr>
<td>$L$</td>
<td>0.079 m</td>
<td>$S$</td>
<td>0.005 m</td>
</tr>
<tr>
<td>$G$</td>
<td>0.0025 m</td>
<td>$\rho_s$</td>
<td>1 100 kg/m$^3$</td>
</tr>
<tr>
<td>$E$</td>
<td>$10^7$ N/m$^2$</td>
<td>$\nu$</td>
<td>0.49</td>
</tr>
<tr>
<td>$\rho_f$</td>
<td>1 000 kg/m$^3$</td>
<td>$\mu_f$</td>
<td>0.001 Pa s</td>
</tr>
<tr>
<td>$\rho_a$</td>
<td>1.225 kg/m$^3$</td>
<td>$\mu_a$</td>
<td>$1.7894 \times 10^{-5}$ Pa s</td>
</tr>
</tbody>
</table>

Simulations will be performed using different combinations of solvers and surrogate models with the IQN-ILSM algorithm. Before proceeding to the actual results, these different solvers are introduced first.

5.1. Main solvers and surrogate models

This section describes the different solvers that will be used for the actual calculations and as surrogate models. In some cases, a certain solver may be used as the actual solver and in other cases as surrogate solver. The tube simulations will be performed with 1D, 2D or 3D solvers and surrogate models. The breaking dam simulation will only make use of 2D solvers. The number of dimensions refers to the computational domain of the solvers, depicted in Figure 7. In this work, the flow and structure solver being coupled within a residual operator always have the same number of dimensions. In other words, a change in dimension only occurs between the residual operator $R$ and the surrogate residual operator $R_s$. However, it would be equally possible to couple, for example, a 2D flow solver with a 3D structure solver with the same interpolation methods.

In the subsequent paragraphs, the solver combinations used for the tube problem will be discussed first. Thereafter, the 2D solvers for the breaking dam case are elaborated on. The final part will deal with how the evaluation cost of different solvers and surrogate models can be compared.

As first solver model for the tube case, consider the combination of a 1D flow and structure model, obtained by averaging over radial sections. These 1D solvers will solely be used as surrogate model. The flow model solves for air with density $\rho_a$ and dynamic viscosity $\mu_a$. The values of these as well as the geometrical parameters are given in Table 2.
Figure 7: Computational domains of the tube case in 1D, 2D and 3D for the structure (green) and flow solvers (blue). The interface is indicated in red. The table gives an overview of the non-zero displacements components ($d_x$, $d_y$, $d_z$) and the non-zero load variables: pressure ($p$) and traction components ($t_x$, $t_y$, $t_z$).
pressure and axial velocity, while the structure model solves for cross-sectional area, from which the radial displacement is derived. Both models have a high degree of simplification: the fluid is assumed incompressible and inviscid, and the axial deformation of the structure is not considered. They do not provide accurate solutions, but they still give realistic results that are suitable to be used as surrogate model. Moreover, they calculate very quickly. For details and equations refer to [9]. Here, the Newmark time discretization for the structure model has been replaced with a backward Euler scheme. This way, both structure and flow solvers have the same time discretization, which is important to avoid pressure fluctuations in time [33]. In the remainder of this section, the notation 1D refers to the combination of these solvers with a division of the tube in 300 segments. The notation 1DA denotes the same solvers, but then the surrogate model \( R_s \) is solved with the analytically determined Jacobian. For details on the analytical calculation of this Jacobian refer to Appendix A.

The second solver combination consists of a 2D flow and structure solver, both of which use an axisymmetric representation of the tube. The flow calculation is performed with Ansys Fluent using a second order upwind scheme for the convective terms. The structure calculation is realized with Abaqus Standard using quadratic axisymmetric elements (CAX8H). For both solvers, time discretization is performed using a backward Euler scheme. In order to obtain the same number of nodes in the axial direction for both solvers, the number of structural elements is half of the number of flow cells, in that direction. Because the calculation is 2D, the flow model calculates three variables (pressure, axial and radial velocity), while the structure model calculates two (axial and radial displacement). In the following, the notations 2DC and 2DF denote the combination of these solvers with a coarse mesh and fine mesh, respectively. The coarse fluid mesh has 100 cells in the axial direction and the fine mesh is obtained by refining the coarse one with a factor three in all directions.

Third is the combination of a 3D flow and structure solver. Similar to the 2D solvers, Ansys Fluent and Abaqus Standard are used for the flow and structure calculation, respectively. Here, quadratic shell elements with reduced integration (S8R) are used. Again, the number of nodes is kept equal in the flow and structure calculation. The flow solver calculates the pressure and all three components of the velocity, while the structure solver determines the three displacement components. Analogously to before, the notations 3DC and 3DF will be used to denote these solvers with a coarse and fine mesh, with respectively 100 and 300 fluid cells in the axial direction. Once more, the fine mesh is obtained by refining the coarse mesh with a factor three in all directions.

Figure 7 provides an overview of the computational domains, interfaces and non-zero variables for the different solvers. As explained in Section 3, the data transfer between solvers with a different number of dimensions, requires extension and averaging.

For the 1D solvers, the interface is identical to the solver domains and is therefore also 1D. In the 2D case, the interface is a line in 2D space. However, with respect to the interpolation, it can be considered 1D, because the interpolation is based on the initial, undeformed coordinates (no y-coordinate). For interpolation between 1D and 2D, the interfaces match. Therefore, the interpolation of the radial component of the displacement or residual is accomplished as before. As there is no axial component in the 1D case, it is neglected (2D to 1D) or set to zero (1D to 2D).

For the 3D solvers, the interface is a surface in 3D space. This interface is different from the 1D or 2D case even if the initial coordinates are considered. When interpolating to 1D or 2D, this is overcome by averaging the axial component of the displacement or residual in the circumferential direction. A similar approach is used for the two other components, but it is important to take rotation into account: the radial component is averaged considering its circumferential position, whereas the circumferential component is disregarded. When interpolating to 3D, the displacement or residual are copied in the circumferential direction, again taking rotation into account for the radial component. There is no circumferential component.

The breaking dam problem is 2D and only a 2D solver combination will be introduced. The multiphase flow calculation is performed with Ansys Fluent using the volume of fluid (VOF) method. Abaqus Standard is used to calculate the structure, which is represented using eight-node plane strain elements (CPE8R). Both solvers use a backward Euler scheme for time discretization. This solver combination is denoted by 2DF on a fine mesh. The notation 2DC refers to its coarser counterpart.

In order to compare the evaluation cost of the different solvers introduced above, the total number of degrees of freedom is determined for each case. Table 3 provides the number of variables for which is solved, and the number of locations of these variables, for both the flow and structure solver. The flow solver uses a finite volume approach in which the variables are stored in the cell centers, whereas the structure solver stores its variables in the nodes,
Table 3: Summary of the different residual operators, which consist of a combination of a flow and structure solver. For both, the number of variables and the number of locations of these variables are shown. The last column gives the sum of the total number of degrees of freedom of both solvers and is representative for the evaluation cost of one iteration of the residual operator.

(a) Tube case.

<table>
<thead>
<tr>
<th>Flow domain</th>
<th>Flow variables</th>
<th>Structure domain</th>
<th>Structure variables</th>
<th>Total degrees of freedom</th>
</tr>
</thead>
<tbody>
<tr>
<td>1D</td>
<td>300</td>
<td>300</td>
<td>1</td>
<td>900</td>
</tr>
<tr>
<td>2DC</td>
<td>1 000</td>
<td>861</td>
<td>2</td>
<td>4 722</td>
</tr>
<tr>
<td>2DF</td>
<td>9 000</td>
<td>7 081</td>
<td>2</td>
<td>41 162</td>
</tr>
<tr>
<td>3DC</td>
<td>128 000</td>
<td>4 864</td>
<td>3</td>
<td>526 592</td>
</tr>
<tr>
<td>3DF</td>
<td>3 456 000</td>
<td>43 392</td>
<td>3</td>
<td>13 954 176</td>
</tr>
</tbody>
</table>

(b) Breaking dam case.

<table>
<thead>
<tr>
<th>Flow domain</th>
<th>Flow variables</th>
<th>Structure domain</th>
<th>Structure variables</th>
<th>Total degrees of freedom</th>
</tr>
</thead>
<tbody>
<tr>
<td>2DC</td>
<td>3 566</td>
<td>85</td>
<td>2</td>
<td>10 868</td>
</tr>
<tr>
<td>2DF</td>
<td>32 298</td>
<td>1 057</td>
<td>2</td>
<td>99 008</td>
</tr>
</tbody>
</table>

the 1D structure solver being an exception. In the residual formulation approach, the flow and structure solvers are always evaluated consecutively, considered as one iteration of a residual operator. With this in mind, the last column provides the total amount of degrees of freedom for the combination of both solvers. Note that not all these degrees of freedom are calculated simultaneously. As a result, a reasonable methodology is obtained to compare the iteration cost for different solver combinations, namely by evaluating the ratio of these total values. For example, the cost of an evaluation of the tube 2DC solvers relative to the tube 2DF solvers is about 4 722/41 162 or approximately 11.5%.

In a partitioned approach, by far most of the calculation time is spent on the individual solvers. Therefore, the number of coupling iterations is a good indication for the performance of the coupling algorithm and will be used to compare different techniques.

As a calculation using IQN-ILSM concerns both iterations of the residual operator and the surrogate residual operator, a total, equivalent number of iterations is determined by multiplying the number of surrogate iterations with the corresponding ratio of total number of degrees of freedom. This approach assumes that all solver costs scale linearly with the number of degrees of freedom, which is not exactly true, but reasonable. Returning to the example of the tube 2DC and 2DF solvers, the equivalent number of iterations for a simulation where the tube 2DC solvers serve as surrogate model for the tube 2DF solvers, is obtained by adding 11.5% of the number of 2DC iterations to the number of 2DF iterations.

Furthermore, the surrogate solvers are black-boxes, which means that the synchronization is done with an additional surrogate iteration. This cost is also taken into account in the equivalent number of iterations.

In this work, the coarse and reduced-physics surrogate models are solved using IQN-ILS and the initial value of a time step is obtained with a linear extrapolation from the solutions of previous time steps. They are considered converged if the norm of the residual becomes smaller than $\epsilon_s = \sqrt{u_s / u}$, where $\epsilon$ is the absolute tolerance of the $L_2$ norm of the residual operator, and $u$ and $u_s$ are the number of degrees of freedom on the interface for $R$ and $R_s$, respectively. This convergence definition maintains the same accuracy of the displacement variable independent of the interface resolution.

As a final note, in each of the following cases, it is important to stress that the solution is independent of the coupling algorithm up to the coupling tolerance. The goal is to find the coupling technique that obtains this solution in the smallest equivalent number of coupling iterations.

5.2. Coupling code: CoCoNuT

The in-house Coupling Code for Numerical Tools, or CoCoNuT, realizes the coupling of the different solvers. So far, this work has employed different software packages for the solution of the flow and structure equations. However,
1.31 1.05 0.79 0.53 0.26 -0.14 kPa

Figure 8: Illustration of the pressure pulse traveling through the tube ($t = 0.0040$ s). The pressure field is shown on the deformed fluid grid.

Table 4: Averaged number of iterations over all time steps required for the transient tube case with 2DF solvers, for the use of a coarse version of the solver as surrogate model (2DC), in different combinations with reuse (R), with and without surrogate prediction. The comparison factor is approximately 11.5%.

<table>
<thead>
<tr>
<th>Jacobian approximation</th>
<th>Solver</th>
<th>Surrogate</th>
<th>Sync.</th>
<th>Equivalent</th>
<th>Solver</th>
<th>Surrogate</th>
<th>Sync.</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>13.18</td>
<td>14.45</td>
<td>1</td>
<td>14.95</td>
<td>14.60</td>
<td>1</td>
<td>14.60</td>
<td>1</td>
</tr>
<tr>
<td>2DC</td>
<td>7.60</td>
<td>14.45</td>
<td>1</td>
<td>9.37</td>
<td>8.72</td>
<td>14.45</td>
<td>1</td>
<td>10.49</td>
</tr>
<tr>
<td>2DC + R</td>
<td>5.49</td>
<td>4.50</td>
<td>1</td>
<td>6.12</td>
<td>6.69</td>
<td>4.48</td>
<td>1</td>
<td>7.32</td>
</tr>
<tr>
<td>R</td>
<td>4.12</td>
<td>14.45</td>
<td>1</td>
<td>5.89</td>
<td>4.58</td>
<td>-</td>
<td>-</td>
<td>4.58</td>
</tr>
<tr>
<td>R + 2DC</td>
<td>4.10</td>
<td>14.45</td>
<td>1</td>
<td>5.87</td>
<td>4.77</td>
<td>14.45</td>
<td>1</td>
<td>6.54</td>
</tr>
<tr>
<td>R + 2DC + R</td>
<td>3.95</td>
<td>4.43</td>
<td>1</td>
<td>4.57</td>
<td>4.54</td>
<td>4.51</td>
<td>1</td>
<td>5.17</td>
</tr>
</tbody>
</table>

Table 6.23 in order to obtain a solution of the FSI problem, a third package is necessary to take care of the coupling of the different solvers. The coupling and the implementation of the IQN-ILSM algorithm is achieved with open-source software CoCoNuT. This Python-based coupling code is developed with an emphasis on modularity, flexibility and understandability. It works out of the box, but is straightforwardly adapted to one’s own needs, as it allows for an ad hoc combination of interpolators, solvers and coupling algorithms. In this work, this has proven especially useful for the calculation of the residual operator and surrogate residual operator, and the different interpolations. The full code can be found in the GitHub repository pyfsi/coconut.

6. Numerical results

6.1. Tube case: transient 2D

The transient tube case considers a static pressure pulse of 1333.2 Pa applied to one end of the tube during 0.003 s, after which the pressure drops to ambient pressure. The total simulation time is 0.01 s in time steps of 0.0001 s. At the outlet, ambient pressure is imposed. These simulations are performed with the 2DF solvers, using different surrogate models: first a coarser version of the same solver (2DC) and thereafter a solver that is reduced in dimension (1D). Further, they are combined with a reuse model, using all available time steps. The tolerance $\epsilon$ of the 2DF residual operator is $10^{-9}$ m. Because the ratio of fluid and structure density is close to one, the added-mass effect plays an important role. An illustration of the pressure pulse traveling through the tube is provided by Figure 8.

Table 4 summarizes the results for the use of a coarse version of the solver as surrogate model (2DC), in different combinations with reuse of previous time steps (R). The first column indicates the type of surrogate model used for the Jacobian approximation. In these notations, the priority decreases from left to right. For example, R + 2DC + R represents the IQN-ILSM algorithm where a reuse surrogate model is applied first, followed by a 2DC surrogate for the remainder of the residual and this 2DC surrogate model itself is also solved using reuse. Further, the table consists of two parts. In the left part a prediction is done with the 2DC surrogate model, as on line 4 in Algorithm 1. In the right
Table 5: Averaged number of iterations over all time steps required for the transient tube case with 2DF solvers, for the use of a less accurate solver as surrogate model (1D), in different combinations with reuse (R), with and without surrogate prediction. The comparison factor is approximately 2.2%.

<table>
<thead>
<tr>
<th>Jacobian approximation</th>
<th>Solver</th>
<th>Surrogate</th>
<th>Sync.</th>
<th>Equivalent</th>
<th>Solver</th>
<th>Surrogate</th>
<th>Sync.</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>16.44</td>
<td>10.76</td>
<td>1</td>
<td>16.70</td>
<td>14.60</td>
<td>-</td>
<td>-</td>
<td>14.60</td>
</tr>
<tr>
<td>1D</td>
<td>9.04</td>
<td>10.76</td>
<td>1</td>
<td>9.30</td>
<td>8.67</td>
<td>10.76</td>
<td>1</td>
<td>8.93</td>
</tr>
<tr>
<td>1D + R</td>
<td>7.82</td>
<td>3.59</td>
<td>1</td>
<td>7.92</td>
<td>7.23</td>
<td>3.59</td>
<td>1</td>
<td>7.33</td>
</tr>
<tr>
<td>R</td>
<td>4.81</td>
<td>10.76</td>
<td>1</td>
<td>5.07</td>
<td>4.58</td>
<td>-</td>
<td>-</td>
<td>4.58</td>
</tr>
<tr>
<td>R + 1D</td>
<td>4.66</td>
<td>10.76</td>
<td>1</td>
<td>4.92</td>
<td>4.47</td>
<td>10.76</td>
<td>1</td>
<td>4.73</td>
</tr>
<tr>
<td>R + 1D + R</td>
<td>4.47</td>
<td>3.59</td>
<td>1</td>
<td>4.57</td>
<td>4.33</td>
<td>3.59</td>
<td>1</td>
<td>4.43</td>
</tr>
</tbody>
</table>

part, on the other hand, the prediction is done with a linear extrapolation. For both, the number of iterations, averaged over all time steps, are given for the residual operator, surrogate residual operator and synchronization, respectively. The last column of each part gives the equivalent number of iterations as explained in Section 5.1. The comparison factor in this case is approximately 11.5%, meaning that the cost of each surrogate iteration corresponds to about one-ninth of the cost of an iteration with the actual solvers.

Some of these cases correspond with existing algorithms. Among the cases without surrogate prediction, the one without Jacobian approximation and the one with reuse correspond to IQN-ILS (without reuse) and IQN-MVJ, respectively. In those cases no surrogate residual operator has to be solved and there are no surrogate nor synchronization iterations. Note that the inclusion of reuse leads to a reduction of the average number of iterations with 69%. Further, IQN-ILSM 2DC corresponds to the multi-level algorithm presented by Degroote and Vierendeels [20], taking into account the differences as detailed in Section 4. Without surrogate prediction, this results in a 28% reduction in the average number of iterations. However, by including surrogate prediction, an additional reduction of 8% is obtained, without requiring any further calculations.

First of all, observe that the effect of the surrogate prediction is positive with respect to the required number of iterations of the actual solvers. However, as the cost of the surrogate is non-negligible, the appropriate criterion for comparison is the equivalent number of iterations. The surrogate prediction has a positive effect on these as well, with the exception of the algorithm without surrogate Jacobian and IQN-ILSM R. This is due to the fact that for these methods without surrogate 2DC prediction (which in fact correspond to IQN-ILS and IQN-MVJ), no 2DC surrogate model has to be solved. In other words, the gain of including 2DC surrogate prediction does not outweigh the cost of the 2DC surrogate evaluations in these particular cases.

Second, looking at the effect of the different Jacobian approximations, there is a general trend of decreasing number of actual solver iterations from top to bottom. This is also true for the equivalent number of iterations, again with the exception of IQN-ILSM R. The required number of equivalent iterations for the surrogate models R + 2DC and R + 2DC + R is higher compared to R alone in the case without surrogate prediction, due to the fact that the benefit of including the 2DC surrogate model does not outweigh its cost. Nonetheless, it is clear that the inclusion of more surrogate models leads to fewer residual operator iterations. Especially the introduction of reuse results in a significant gain. This is no surprise, as the variation of both the Jacobian and the change in solution from one time step to another is limited. Moreover, the reuse surrogate model is essentially free, as no surrogate model has to be solved. Although the cost of the operations increases if more time steps are reused, this cost remains small compared to the evaluation cost of the solvers, in particular due to its efficient implementation with linear complexity.

Analogously to the previous table, Table 5 represents the results in case a reduced-physics solver is used as surrogate model (1D), in different combinations with reuse (R). As the surrogate model is only 1D, the comparison factor, equal to 2.2%, is significantly lower. Note that the entries for IQN-ILS and IQN-MVJ are identical to before, as in those cases no additional surrogate model has to be solved.

In contrast to the 2DC surrogate model, the cases without prediction perform consistently better. Here, the surrogate solution is too far from the actual solution to contribute to the performance with a prediction, in contrast to the
2DC surrogate model where the surrogate prediction did accelerate the convergence. Although the 1D surrogate model should not be used for prediction, the 1D surrogate Jacobian does realize an important improvement. As before, the number of iterations decreases from top to bottom in the table, both for the iterations of the actual solvers as the equivalent number of iterations. IQN-ILS R without surrogate prediction is again the only exception. While including an additional 1D surrogate Jacobian does decrease the number of iterations of the actual solvers, the overall cost, represented by the equivalent number of iterations, increases. However, if this 1D surrogate model is solved using reuse, there is an overall improvement.

Finally, the performance of the 1D surrogate model is compared with the 2DC surrogate model. It is clear that the 2DC surrogate model with prediction provides the greatest reduction in the number of iterations of the actual solvers. However, with respect to the equivalent number of iterations, the 1D surrogate model without prediction performs generally better, due to the larger cost associated to the 2DC surrogate model evaluations. The comparison of both surrogate models without surrogate prediction reveals that the use of the 1D surrogate Jacobian generally results in a higher performance gain. This is presumably linked to the discretization in the predominant, axial flow direction. The 1D solvers and 2DF solvers both have 300 degrees of freedom in the axial direction, whereas 2DC only has 100.

The aforementioned results demonstrate the powerful performance of the reuse surrogate model. The more time steps are reused, the more it will dominate. Towards the end of the simulation many time steps are available, in contrast to the first time step, when there are none. Figure 9 shows how the addition of a surrogate model can greatly improve the convergence in the first step. The convergence of the actual solvers for both surrogate models is compared in Figure 9a. For each surrogate, the case without surrogate prediction (no SP), and without surrogate Jacobian (no SJ) are included as well. The improvement with respect to IQN-ILS is clear. In contrast to the results averaged over all time steps, the 2DC surrogate model outperforms the 1D surrogate model in each case and the best result for each surrogate model is obtained by using both the surrogate Jacobian and surrogate prediction. Note that in the case of the 1D surrogate with prediction, the initial norm of the residual is not much lower than the case without prediction, which shows that the 1D surrogate solution is not very accurate. Figure 9b shows the convergence of the surrogate models. Whereas all data on the left figure shows the convergence of the residual operator 2DF, the figure on the right contains the convergence of two different residual surrogate operators, which should therefore not be compared.

6.2. Tube case: transient 3D

The second example considers the same transient case, which is now solved using the 3DF solvers and a 1D, 1DA or 3DC surrogate model. The results are summarized in Table 6. The comparison factors relative to 3DF are 0.01%, 0.01% and 3.77% for the 1D, 1DA and 3DC surrogate models, respectively and the tolerance ε of the 3DF residual operator is $10^{-9}$ m.
Table 6: Required number of iterations in the first time step of the transient tube case with 3DF solvers, for the use of a 1D, 1DA and 3DC surrogate model. The comparison factors for the 1D, 1DA and 3DC surrogate models are approximately 0.01%, 0.01% and 3.77%, respectively.

<table>
<thead>
<tr>
<th>Jacobian approximation</th>
<th>Solver</th>
<th>Surrogate</th>
<th>Sync.</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>IQN-ILS</td>
<td>17.00</td>
<td>-</td>
<td>-</td>
<td>17.00</td>
</tr>
<tr>
<td>IQN-ILSM 1D</td>
<td>10.00</td>
<td>14.00</td>
<td>1</td>
<td>10.00</td>
</tr>
<tr>
<td>IQN-ILSM 1DA</td>
<td>10.00</td>
<td>5.00</td>
<td>1</td>
<td>10.00</td>
</tr>
<tr>
<td>IQN-ILSM 3DC</td>
<td>9.00</td>
<td>17.00</td>
<td>1</td>
<td>9.68</td>
</tr>
</tbody>
</table>

Observe that the cost of the surrogate models compared to the actual solvers is significantly lower than in the previous 2D case. Before, the coarse solver (2DC) still had an important cost, which could outweigh its induced benefit. Here, however, the comparison factor is only 3.77%, which is about three times lower.

Further, Figure 10 visualizes the convergence in the first time step. It shows that the performance of the 1DA surrogate Jacobian is very close to that of the 1D case, which is a demonstration of the quality of the low-rank, approximated Jacobian based on input-output pairs. Again, the surrogate solution of the 1D and 1DA surrogate models are far less accurate than the one obtained with the coarser version of the solver. However, the comparable required number of iterations in all three cases shows the importance of a fine discretization in the dominant flow direction, which is identical for the 3DF, 1D and 1DA solvers, but three times coarser for the 3DC solvers.

6.3. Tube case: steady 3D

The third example is a steady case, for which there are no previous time steps available to use as reuse surrogate model. However, the results for the previous cases in the first step show that the use of a coarse or less accurate surrogate model may still greatly improve convergence.

The total pressure at the inlet and outlet of the tube are respectively 10 kPa and 9 kPa. As a result, flow is created and the tube inflates as illustrated by Figure 11. Again, the 3DF solvers are used, aided by a 1D, 1DA or 3DC surrogate model. The comparison factors and tolerance are the same as before. The results are summarized in Table 7.

Remark that in a steady case the inertia of the fluid does not contribute and hence there is no added-mass effect. Similar to the previous two examples, the first case uses a relaxation factor \( \omega \) of 0.05 for the first iteration. In the presence of the added-mass effect this is required in order to achieve a feasible value for the second iteration, as at that point no secant Jacobian is available yet: \( x^1 = x^0 + \omega r^0 \). However, the need disappears when a surrogate model with
**Figure 11:** Illustration of the inflated tube. The pressure field is shown on the deformed fluid grid.

**Table 7:** Required number of iterations for the solution of the steady tube case with 3DF solvers, for the use of a 1D, 1DA and 3DC surrogate model. The comparison factors for the 1D, 1DA and 3DC surrogate models are approximately 0.01%, 0.01% and 3.77%, respectively. Note that no synchronization needs to occur, as the case is steady.

<table>
<thead>
<tr>
<th>Jacobian approximation</th>
<th>Solver</th>
<th>Surrogate</th>
<th>Sync.</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>IQN-ILS $\omega = 0.05$</td>
<td>8.00</td>
<td>-</td>
<td>-</td>
<td>8.00</td>
</tr>
<tr>
<td>IQN-ILS no relaxation</td>
<td>6.00</td>
<td>-</td>
<td>-</td>
<td>6.00</td>
</tr>
<tr>
<td>GAUSS-SEIDEL no relaxation</td>
<td>7.00</td>
<td>-</td>
<td>-</td>
<td>7.00</td>
</tr>
<tr>
<td>IQN-ILS 1D no SJ</td>
<td>6.00</td>
<td>6.00</td>
<td>-</td>
<td>6.00</td>
</tr>
<tr>
<td>IQN-ILS 1DA no SJ</td>
<td>6.00</td>
<td>4.00</td>
<td>-</td>
<td>6.00</td>
</tr>
<tr>
<td>IQN-ILS 3DC no SJ</td>
<td>4.00</td>
<td>6.00</td>
<td>-</td>
<td>4.23</td>
</tr>
</tbody>
</table>

A surrogate Jacobian is used. Here, there is no added-mass and a relaxation factor is not needed. The second case shows the result without relaxation. In other words, a pure Gauss-Seidel update is applied after the first iteration. The third case illustrates that, in the absence of added-mass, even pure Gauss-Seidel coupling suffices to achieve convergence.

Nonetheless, a surrogate model may still reduce the required number of equivalent iterations, if used as predictor. In these cases, the surrogate Jacobian is not included, as the coupling does not require stabilization and its inclusion does not accelerate convergence.

Further, Figure 12 provides the convergence history. Note, that the surrogate solution of the 3DC solver is again much closer to the actual solution than the 1D or 1DA surrogate approximation. Moreover, the figure clearly shows that without the use of a surrogate Jacobian, the 1D and 1DA surrogate models have the same effect, as expected.

### 6.4. Breaking dam case: transient 2D

The breaking dam example simulates the bending of a flexible gate due to the changing pressure exerted by the water volume, initially located to the right of the gate. The original height of the water level is $H$. The total simulation time is 0.1 s in 100 time steps of 0.001 s. At the top and left-most boundaries, ambient pressure is imposed. These calculations are performed with the 2DF solvers, using different combinations of two types of surrogate models: a coarser version of the same solver (2DC) and a reuse model (R), using at most 50 previous time steps. The tolerance $\epsilon$
Table 8: Averaged number of iterations over all time steps required for the transient breaking dam case with 2DF solvers, for the use of a coarse version of the solver as surrogate model (2DC), in different combinations with reuse (R), with and without surrogate prediction. The comparison factor is approximately 11.0%.

<table>
<thead>
<tr>
<th>Jacobian approximation</th>
<th>Solver</th>
<th>Surrogate</th>
<th>Sync.</th>
<th>Equivalent</th>
<th>Solver</th>
<th>Surrogate</th>
<th>Sync.</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>7.49</td>
<td>7.14</td>
<td>0</td>
<td>8.27</td>
<td>7.19</td>
<td>-</td>
<td>-</td>
<td>7.19</td>
</tr>
<tr>
<td>2DC</td>
<td>5.67</td>
<td>7.14</td>
<td>0</td>
<td>6.45</td>
<td>5.53</td>
<td>7.14</td>
<td>0</td>
<td>6.31</td>
</tr>
<tr>
<td>2DC + R</td>
<td>5.64</td>
<td>4.69</td>
<td>0</td>
<td>6.15</td>
<td>5.45</td>
<td>4.69</td>
<td>0</td>
<td>5.96</td>
</tr>
<tr>
<td>R</td>
<td>5.08</td>
<td>7.14</td>
<td>0</td>
<td>5.86</td>
<td>5.30</td>
<td>-</td>
<td>-</td>
<td>5.30</td>
</tr>
<tr>
<td>R + 2DC</td>
<td>4.44</td>
<td>7.14</td>
<td>0</td>
<td>5.22</td>
<td>4.59</td>
<td>7.14</td>
<td>0</td>
<td>5.37</td>
</tr>
<tr>
<td>R + 2DC + R</td>
<td>4.43</td>
<td>4.69</td>
<td>0</td>
<td>4.94</td>
<td>4.20</td>
<td>4.69</td>
<td>0</td>
<td>4.71</td>
</tr>
</tbody>
</table>

of the 2DF residual operator is $10^{-9}$ m. Because the ratio of fluid and structure density is close to one, the added-mass effect plays an important role. An illustration of the deflection of the gate is provided by Figure 13.

Table 8 summarizes the results, using the same notations as before. Here, no synchronization is performed, because it was observed that synchronization did not help the performance. Again the overall performance is summarized by the column with the equivalent number of iterations as explained in Section 5.1. The comparison factor in this case is approximately 11.0%, meaning that the cost of each surrogate iteration corresponds to about one-ninth of the cost of an iteration with the actual solvers.

First, note that, because no synchronization is performed, the required number of iterations of all surrogate residual operators without reuse is identical. The same is true for all surrogate residual operators with reuse.

Second, in most of the cases, the 2DC surrogate prediction has a negative effect on the number of actual solver iterations and therefore also on the equivalent number of iterations, IQN-ILS M R and IQN-ILS M 2DC R being the exception. The predictive capability of this 2DC surrogate is therefore less than what was observed in the tube case.

Third, when more surrogate models are added, there is a decreasing trend in the required number of actual solver iterations both with and without 2DC surrogate prediction. This again demonstrates the benefit of adding more surrogate models. This trend is also visible in the equivalent number of iterations, which take the cost of the surrogate

---

4For this case the dynamic mesh tolerances were changed slightly.
model into account. Only for IQN-ILSM R 2DC without prediction, the additional cost of having to evaluate the 2DC surrogate model, compared to IQN-ILSM R, is greater than the achieved benefit.

7. Conclusion

This work presents a new generalized framework, called IQN-ILSM, to include surrogate models in quasi-Newton methods. Whereas quasi-Newton methods are excellent in stabilizing the coupling of two solvers by building an approximate Jacobian based on secant information from input-output pairs, they are a complete black-box approach. Additional information in the form of a surrogate model allows to enhance the performance even further. The surrogate contribution is twofold: it provides an approximate solution to be used as initial value, as well as a supplementary Jacobian, which is used throughout the coupling iterations of a time step with diminishing importance as more secant information becomes available. The IQN-ILSM method achieves the inclusion of additional information while keeping the computational complexity linear. Different types of surrogate models have been discussed, including a coarser version of the solvers, reduced-physics solvers, an analytical surrogate model and reuse of interface data from previous time steps. Furthermore, it has been shown how existing methods such as the IQN-IMVLS method from Spenke et al. and the multi-level technique of Degroote and Vierendeels also fit inside this framework.

The performance of IQN-ILSM has been evaluated on a flexible tube case and a breaking dam case, for the mentioned surrogate models alone as well as different combinations. Particularly, in a transient case, the reuse model is very efficient, as the previous time steps and the current one share a strong resemblance. Moreover, this surrogate model is essentially free. In the examined tube case, the reuse of previous time steps led to a reduction of 69% in the average number of iterations per time step. For the breaking dam case this was 26%. Nonetheless, the other types of surrogate models also provide an important improvement, especially in a steady case or at the start of a transient calculation. In the 2D transient tube case examined here, the coarse and 1D surrogate model led to a reduction of 36% and 39%, respectively. For the coarse surrogate model, performing a prediction with the surrogate model was generally beneficial, in contrast to the 1D surrogate model. The best overall performance was obtained with a combination of 1D and reuse surrogate models leading to a reduction of 70% in the required number of iterations. In the breaking dam case, the introduction of a coarse surrogate model led to a reduction of 12% in the average number of iterations per time step. The highest overall reduction was achieved with IQN-ILSM R 2DC R without surrogate
prediction and amounted to 34%. Finally, in the steady tube case, the coarse surrogate model performed best and led to a reduction of 30%.

**Acknowledgment**

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

**Appendix A. Jacobian of the analytical 1D solvers**

Eq. (28) formulates the inverse Jacobian of the residual operator in function of the Jacobians of the flow and structure solvers. For brevity the index $s$ will be dropped. The required Jacobian is thus retrieved by determining the Jacobian of the structure solver, a backward Euler discretization has been used. Instead of a Newmark time discretization for the structure solver, a backward Euler discretization has been used. However, this has no effect on the methodology to determine the Jacobians. For the sake of completeness, the set of discretized equations for the flow and structure solver are reproduced here.

For the flow solver, the discretized continuity and momentum equations for a cell with center $i$ are given by

$$
\frac{\Delta x}{\Delta t} \left( v_i^{k+1} - v_i^n a_t^n \right) + \tilde{r}_i^{k+1,1/2} - \tilde{r}_{i-1/2}^{k+1,1/2} - \frac{\pi \alpha}{\rho_f} \left( \tilde{p}_{i+1}^{k+1} - 2\tilde{p}_i^{k+1} + \tilde{p}_{i-1}^{k+1} \right) = 0 \quad (A.1a)
$$

$$
\frac{\Delta x}{\Delta t} \left( \tilde{v}_i^{k+1,1/2} - v_i^n a_t^n \right) + \tilde{v}_i^{k+1,1/2} - \tilde{v}_{i-1/2}^{k+1,1/2} - \frac{1}{2\rho_f} \left( a_{i+1/2}^{k+1} \left( \tilde{p}_{i+1}^{k+1} - \tilde{p}_i^{k+1} \right) + a_{i-1/2}^{k+1} \left( \tilde{p}_i^{k+1} - \tilde{p}_{i-1}^{k+1} \right) \right) = 0 \quad (A.1b)
$$

for $v_i \geq 0$, where $\Delta x$ is the length of a cell, $\Delta t$ the time step, $v$ the axial velocity and $a$ the cross-sectional area of the tube. The values with a tilde are calculated, while the ones without are given. As before, the superscripts $n$ and $k$ denote the time step and iteration respectively. The superscript $n+1$, indicating the current time step, has been dropped for brevity. The subscript $i$ refers to the current cell, whereas the subscripts $i+1$ and $i-1$ indicate the adjacent cell centers. The subscripts $i \pm 1/2$ denote the values at the cell interfaces calculated with

$$
\begin{align*}
v_{i-1/2} &= (v_i - v_{i+1})/2 \\
v_{i+1/2} &= (v_i + v_{i+1})/2.
\end{align*}
$$

Finally, $\alpha$ is a pressure stabilization term, which is introduced to prevent pressure wiggles due to central discretization of the pressure.

The equation of the structure solver has been discretized as

$$
\frac{\rho_s h}{\Delta t^2} \ddot{r}_i^{k+1} + \frac{b_1}{\Delta x^4} \left( \dot{r}_i^{k+1,2} - 4\dot{r}_i^{k+1,1} + 6\dot{r}_i^{k+1,0} - 4\dot{r}_i^{k+1,-1} + \dot{r}_i^{k+1,-2} \right) - \frac{b_2}{\Delta x^2} \left( \ddot{r}_i^{k+1,1} - 2\dot{r}_i^{k+1,0} + \dot{r}_i^{k+1,-1} \right) + b_3 \left( r_i^{k+1} - r_o \right) = (p_i^{k+1} - p_o) + p_o h \left( \frac{1}{\Delta t^2} \dot{r}_i^n + \frac{1}{\Delta t} \ddot{r}_i^n \right) \quad (A.3a)
$$

for cell $i$. The overdot indicates a time derivative. The parameters $b_1$, $b_2$, and $b_3$ are related to the inner action of the bending, axial tension and circumferential stress. For a thin-walled tube, clamped in the axial direction they are given by

$$
\begin{align*}
b_1 &= \frac{hE}{1 - \nu^2} \frac{h^2}{12} \\
b_2 &= \frac{hE}{1 - \nu^2} \frac{h^2 2\nu}{12 r_o^2} \\
b_3 &= \frac{hE}{1 - \nu^2} \frac{1}{12 r_o^2} + \frac{hE}{1 - \nu^2} \frac{h^2 1}{12 r_o^2}.
\end{align*}
$$

(A.3b)
where the second term of $b_3$ is neglected with respect to the first one. Once the coupling iterations within time step $n + 1$ have converged, the corresponding velocity is calculated as

$$v_{n+1}^i = v_n^i - \frac{r_{n+1}^i - r_n^i}{\Delta t}. \quad (A.3c)$$

Returning to the determination of the flow Jacobian, the flow solver $\mathcal{F}$ has an input vector $v$, which contains the radial component of the displacement $d_y$ and the output vector $\tilde{y}$ contains the pressure $p$. Therefore, the column vectors $d_y$ and $p$ will be used to denote the in- and output of $\mathcal{F}$ in the following. The corresponding system of equations can be written as

$$\mathcal{F}(v, p, a) = 0. \quad (A.4)$$

where the column vector $v$ contains the axial velocity, the column vector $p$ the pressure and the column vector $a$ the cross-sectional area of the tube. While taking the differential, the velocity and pressure vectors are grouped resulting in

$$d\mathcal{F} = \partial \mathcal{F} / \partial (v, p) d(v, p) + \partial \mathcal{F} / \partial a da = 0, \quad (A.5)$$

where the symbolic notation $(v, p)$ refers to the concatenation of the two vectors. Consequently, it is found that

$$\frac{d(v, p)}{da} = - \left( \frac{\partial \mathcal{F}}{\partial (v, p)} \right)^{-1} \frac{\partial \mathcal{F}}{\partial a}. \quad (A.6)$$

From this matrix, only the rows corresponding to $p$ are retained to obtain $\frac{dp}{da}$. Note that $da = 2\pi \text{diag}(r) dr = 2\pi \text{diag}(r) d d_y$, where $\text{diag}(r)$ is the diagonal matrix whose entries are the elements of the vector $r$. Therefore, the Jacobian $\mathcal{F}'$ equals

$$\mathcal{F}' = \frac{dp}{dd_y} = 2\pi \frac{dp}{da} \text{diag}(r) \quad (A.7)$$

The resulting flow Jacobian depends on the values of $v$, $p$, $a$ and the corresponding values in the previous time steps.

The 1D structure solver $\mathcal{S}$ has an input vector $y$, which contains the pressure $p$. The output vector $\tilde{x}$ contains the radial component of the displacement $d_y$. Again, the column vectors $p$ and $d_y$ are used as in- and output. The corresponding system of equations can be written as

$$\mathcal{S}(r, p) = 0, \quad (A.8)$$

where the column vector $r$ contains the radial coordinate of the tube wall interface and should not be confused with the residual. After taking the differential,

$$d\mathcal{S} = \frac{\partial \mathcal{S}}{\partial r} dr + \frac{\partial \mathcal{S}}{\partial p} dp = 0, \quad (A.9)$$

it is found that

$$\frac{dr}{dp} = - \left( \frac{\partial \mathcal{S}}{\partial r} \right)^{-1} \frac{\partial \mathcal{S}}{\partial p}. \quad (A.10)$$

By noticing $\frac{\partial S}{dp} = -I$ and the fact that a change in radius of the tube wall is identical to a change in radial displacement, the Jacobian $\mathcal{S}'$ equals

$$\mathcal{S}' = \frac{dd_y}{dp} = \left( \frac{\partial \mathcal{S}}{\partial r} \right)^{-1} \quad (A.11)$$

This Jacobian is constant, as it does not depend on the radius or pressure and therefore should only be calculated once.

Additional equations to enforce the boundary conditions are important for the solver’s evaluations, but should not be considered in the Jacobian determinations as those boundaries are not included in the fluid-structure interface. As a final note, it is clear that the calculation cost of the analytical Jacobian increases drastically with the number of degrees of freedom of the solvers, due to the explicit matrix construction, multiplication and inversion of dense matrices. However, as the number of degrees of freedom in this 1D solver is typically limited, this cost remains minor.
References


