UNIONE EUROPEA
Fondo Europeo di Sviluppo Regionale

Ministero dell'Università e della Ricerca

PON RICERCA E INNOVAZIONE 2014 - 2020

Consiglio Nazionale delle Ricerche

Cluster Trasporti

CTN01_00176_163601

TRIM Tecnologia e Ricerca Industriale per la Mobilità Marina

TRIM
Tecnologia e Ricerca Industriale per la Mobilità Marina

Efficient discretisation techniques for numerical simulation of flows in maritime problems

Sotto-Progetto	Efficienza	
Obiettivo Realizzativo	Sviluppo software innovativi	
Descrizione attività	Costruzione interfaccia CAD Sviluppo codice BEM isogeometrico Sviluppo codice bifase isogeometrico Sviluppo metodi a base ridotta	
Tipo di documento	Rapporto Tecnico	
Codice del documento	SP.4-OR.9-D.1	
Data di emissione	30/09/2021	
Redazione	Andrea Mola, Marco Tezzele, Gianluigi Rozza	SISSA Scuola Internazionale Superiore di Studi Avanzati

Consiglio Nazionale delle Ricerche FINCANTIERI CETENA SEASTEMA AZIMUT BENETTI GROUP PERINI NAVI
mare FVG Navtec LIBERTYlines OPUS automazione NEXTWORKS ENGINEERING FORWARD ib SULTAN

Titolo documento

Efficient discretisation techniques for numerical simulation of flows in maritime problems

Codice documento SP.4-OR.9-D.1

Distribuzione Pubblico

Rev.	Data	Pagine	Redazione	Responsabile
0	30/09/2021	2+47	Andrea Mola, Marco Tezzele, Gianluigi Rozza	Gianluigi Rozza

L’attività descritta nella presente pubblicazione è stata finanziata dal Progetto TRIM — Tecnologia e Ricerca Industriale per la Mobilità Marina — coordinato dal Consiglio Nazionale delle Ricerche e finanziato dal Ministero dell’Università e della Ricerca nell’ambito dell’iniziativa dei Cluster Tecnologici Nazionali.

Contents

Summary

In this report we present the state of the art of efficient discretisation techniques suited for fluid flows in maritime problems.

We focus on boundary element method properly fitted into an iso-geometric setting, as well as on a multiphase flow solver fitting the same iso-geometric setting.

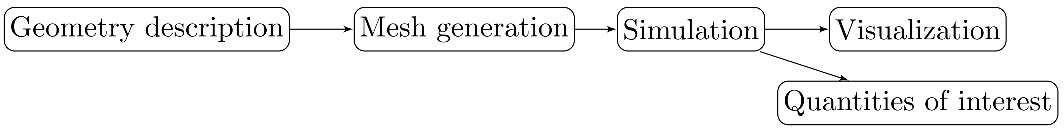
A further effort is devoted to the development of model order reduction methods for efficient simulation and computational cost saving.

Test cases are analysed to highlight the efficiency of the proposed developments in a naval engineering framework. A strategic importance is given to iso-geometric setting to be able to incorporate CAD geometries in an efficient way in the full computational pipeline.

1 CAD Interface

The traditional workflow of finite element, finite volume, and finite difference simulations of physical processes consists of three phases: what is called “preprocessing”; the actual numerical solution of a partial differential equation; and what is called “postprocessing”. In this workflow, preprocessing generally means the generation of a geometric description of the domain on which one wants to solve the problem — either through the use of Computer Aided Design (CAD), or by combining simpler geometries into one via constructive solid geometry (CSG) — and the use of a mesh generator that uses the geometry to create the computational grid on which the simulation is then run. On the other end of the pipeline, postprocessing consists of the visualization of the computed solution and the extraction of quantities of interest.

Overall, this “traditional” workflow can be visualized through the following graph in which information is only propagated from one box to the next:



The fundamental issue with this workflow is that geometry information is only passed on to the mesh generator, but is, in general, not available at the later stages of the pipeline. This approach — which to our knowledge is used in all commercial and open source simulation tools today — is appropriate if the simulator is relatively simple; specifically, if (i) simulation and postprocessing tools only rely on a single, fixed mesh as their sole information on the geometry of the domain on which to solve the problem under consideration, and (ii) if one uses lowest-order finite-element, finite-volume, or finite-difference discretizations for which it is sufficient to use the piecewise linear approximation of the boundary that is generally obtained by replacing the “underlying” geometry of the problem by a fixed mesh characterized solely by its vertices and assuming straight edges. In practice, the limitations of the workflow mentioned above imply that to most finite element analysts, “the mesh is the domain”, even though to the designer the mesh is generally an imperfect approximation of some underlying geometry typically understood to be a CAD or CSG description.

Yet, simulation tools have become vastly more complex since the formulation of the workflow above several decades ago, and as we will show below, we can not make use of their full potential *unless geometry information is propagated to the simulation and analysis tools*, as well as to the postprocessing tools. For example, geometry information is important in the following contexts:

- Modern simulators no longer use only a single mesh, but create hierarchies of meshes. Two typical applications are the generation of a sequence of refined meshes to enable the use of geometric multigrid solvers or preconditioners [1, 2]; and the use of adaptive mesh refinement to obtain a mesh that is better suited to the accurate solution of the underlying equation [3, 4], without a-priori knowledge of how the final mesh will look like. In a similar vein, for large-scale computations with more than a few tens of millions of elements and massively parallel systems, the I/O related to creating and accessing the mesh data structure is often a serious bottleneck. Much better performance can be obtained by reading smaller meshes that get refined as part of the simulation. In all of these cases, refining the mesh involves the computation of new grid points from inside the simulation, and these points need to respect the same geometry used for the original mesh.

- Accurate simulators use curved cells and higher order mappings both at the boundary and in the interior of the mesh. How exactly these curved cells should look requires information about the underlying geometry. To illustrate the importance of this point, let us mention that it has been understood theoretically for a long time that one loses the optimal order of finite element discretizations with polynomial degrees greater than one, if one does not also use higher-order approximations of the boundary [5, 6, 7, 8, 9, 10, 11]. This is also known from practical experience; for example [12] presents experimental evidence, and [13] and the references therein provide an excellent example of the lengths one needs to go to to recover higher-order accuracy if the underlying geometry is not available (Figure 3 below also illustrate the issue). In other words, a finite element solver that does not know about the underlying geometry will compute a solution with an unnecessarily large error or use an unnecessarily large amount of computational work. Alternatively, additional points for a high-order representation need to be computed as a separate workflow step between the meshing and the simulation, as used, e.g., in [14, 15, 16].

- In many contexts — during the simulation, but also during accurate visualization and evaluation of quantities of interest — it is important to know the correct normal vector to faces of cells. An approximation can be obtained by simply taking the location of vertices as provided in the mesh file and their connection into cells, as the ground truth. But the vectors so computed are not consistent with the true, underlying geometry from which the mesh was originally generated. As a consequence, visualizations are often not faithful representations of the actual computations, and quantities of interest are not evaluated to the full accuracy possible. For example, accurate evaluation of mass or energy fluxes across a boundary requires accurate knowledge of the normal vector. For fourth order equations, accurate knowledge of the normal vector may also be required to retain optimal convergence order of finite element schemes if it is necessary to construct C^1 approximations of a curved boundary (see [6] and the references therein).

We will provide more examples below where geometry information is used in finite element simulations.

These considerations point to a need to propagate geometry information not only to the mesh generator, but indeed also to the simulation engine. This raises the question how this can best be done. To the best of our knowledge, no commercial or open source tools do this in a consistent way today. Furthermore, we have to realize that geometries are often described through complex CAD systems that are either not open source, complicated to interact with, or can only be accessed in proprietary ways; as a consequence, it makes sense to ask what kinds of queries a generic geometry engine has to be able to answer to satisfy the needs of simulation software. In order to address all of these points, we have undertaken a comprehensive study with the following goals:

1. Identify a *minimal set of operations* — which we will call "primitives" — that geometry tools need to be able to perform to satisfy the needs of simulation tools.
2. Provide a *comprehensive review of geometry operations* performed in a widely used finite element library and a large application code that is built on the former, with the aim of verifying that the minimal set of operations outlined above is indeed sufficient.
3. Discuss *ways in which geometry tools can implement the minimal set of operations*, given the kinds of geometries and information that is typically available in industrial and research workflows.

We will state the primitives in the form of *oracles*, i.e., as blackboxes that given certain inputs produce appropriate outputs, without specifying how exactly one would need to implement this operation. This reductionist approach is often appropriate when one wants to interface with one of many possible geometry engines, each of which may have its own way of implementing the operations. In such cases, it is often useful to only specify a minimal interface that all engines can relatively easily implement. A common way of representing oracles in object-oriented codes is by providing an abstract base class with unimplemented virtual functions; the base class is then the oracle, whereas derived classes provide actual implementations.

The result of our work is the realization that only two geometry primitives are sufficient. We will discuss these in Section 1.1. Section 1.1.2 is then devoted to the question of how one would implement these two operations in the most common situation in which the geometry is described implicitly through a collection of NURBS patches in typical CAD engines. We demonstrate the practical benefits of the integration of geometry and simulation in Section 1.3.

The practical implication of our work is the identification of a minimal interface that allows the coupling of geometry and simulation engines. We have tested these interfaces via the widely used finite element software DEAL.II [17, 18] and the Advanced Simulator for Problems in Earth ConvecTion ASPECT [19, 20] with geometry descriptions that are either given explicitly, or via the OpenCASCADE library that is widely used for CAD descriptions [21]. All of the results of our work are available in the publicly released versions of DEAL.II and ASPECT. The verification of our approach using these examples, and the fact that the sufficient interface is so small, should provide the certainty necessary to follow a similar path for integrating other simulation and geometry software packages.

We end this introduction by remarking that one may also wish to provide geometry information to the postprocessing stage. For example, this would allow visualization software to produce more faithful graphical representations of the solutions generated by simulators, free of artifacts that result from incomplete knowledge of the domain on which the simulation was performed. We are not experts in visualization and consequently leave an investigation of the geometry needs of visualization software to others. We will, however, comment that the evaluation of quantities of interest — such as stresses at individual points, heat and mass fluxes across boundaries, or average and extremal values for certain solution fields — may be most efficiently done from within the simulator itself, given that geometry information as well as knowledge of shape functions and other details of the discretization are already available there; indeed, this is the approach chosen in the Aspect code mentioned above.

1.1 Fundamental geometric primitives

As we will discuss in detail in the following section, it turns out that the geometric queries needed by finite element codes — such as finding locations for new vertices upon mesh refinement, or computing vectors normal to the surface — can be reduced to only two "primitive" operations: (i) finding a new point given a set of existing points with corresponding weights, and (ii) computing the tangent vector to a line connecting two existing points. This realization of a minimal set of operations allows us great flexibility in choosing software packages that actually provide these primitives, and minimizes the dependency a finite element code incurs when using an external geometry package.

In the literature, implementations that can answer a small number of very specific questions — i.e., provide certain simple operations — are typically referred to as "oracles". The point of postulating the existence of an oracle is that it allows us to separate the *design* of a code from its actual *implementation*. In the current case, all that matters for the purposes of the current section is that an oracle exists that can answer two specific questions, and whose answers can be used throughout a finite element code.

In order to motivate the two geometric primitives that we postulate are sufficient for almost all finite element operations, let us first provide two scenarios of relevance to us. First, consider a d-dimensional surface embedded into a higher dimensional space; one might think of this surface as the boundary of a volume within which we would like to simulate certain physics. The second setting is a d-dimensional volume geometry in d-dimensional space for which we would like to consider interior cells to also deviate from the simplest, d-linear shape; an example would be a volume mesh that extends away from a curved wing around which we would like to simulate air flow. We will use these two scenarios for the examples below.

1.1.1 Statement of primitives

Given this background, the two operations we have found are necessary are the following:

Primitive 1 ("New point") *Given* $N \geq 2$ *existing points* $\mathbf{x}_1, \ldots, \mathbf{x}_N$ *and weights*

$w_1, \ldots, w_N$ that satisfy $\sum_{n=1}^{N} w_n = 1$, return a new point $\mathbf{x}^*(\mathbf{x}_1, \ldots, \mathbf{x}_N, w_1, \ldots, w_N)$ that interpolates the given points, weighting each $\mathbf{x}_n$ with w_n.

Primitive 2 ("Tangent vector") *Given two existing points $\mathbf{x}_1, \mathbf{x}_2$, return the (non-normalized) "tangent" vector $\mathbf{t}$ at point $\mathbf{x}_1$ in direction $\mathbf{x}_2$, defined by*

$$\mathbf{t}(\mathbf{x}_1, \mathbf{x}_2) = \lim_{w \to 0} \frac{\mathbf{x}^*(\mathbf{x}_1, \mathbf{x}_2, (1-w), w) - \mathbf{x}_1}{w}. \tag{1}$$

We will give many examples below of how these two operations are used. As we will show, all other geometric questions a typical finite element code may have can be answered exactly, without approximation, with only these two operations. For the moment, one can think of use cases as follows:

- When adaptively refining a mesh, one needs to introduce new vertex locations on edges, faces, and in the interior of cells. This is easily done using the first of the two primitives above, using the existing vertex locations on an edge, face, or cell as input.
- Computing the normal vector to a face at the boundary of a three-dimensional domain can be done by taking the cross product of two tangent vectors that pass through the point at which we need the normal vector. These tangent vectors are computed via the second primitive.

A concrete implementation that describes a particular geometry will be able to provide answers to the two operations based on its knowledge of the actual geometry. For example, and as discussed in Section 1.1.2, the wing of an airplane would be described using a CAD geometry that can be queried for the primitives above at least on the boundary of the domain. More work — also described in Section 1.2 below — will then be necessary to extend this description into the interior. In other cases, however, the description of the geometry may be available everywhere — for example, if the geometry of interest is an analytically known object such as a sphere.

Remark 1 *As we will see below, the operations that finite element codes require can be implemented by only querying information from objects that describe an entire manifold, without having to know anything about the triangulation that lives on it, or in particular where the triangulation's boundary lies. This greatly simplifies the construction of oracles because they do not need to know anything about meshes, or the domain on which we solve an equation. For example, we will be able to use a CAD geometry of, say, the entire hull of a ship even if we want to solve an equation on only parts of the surface; that there is a boundary to the domain on which we solve, and where on the hull it lies, will be of no importance to the oracle that will answer our queries.*

Information about the domain, its boundary, and the mesh that covers it, will of course be used in constructing the inputs to the queries, but is not necessary in computing their outputs.

Remark 2 *Given the definition of the tangent vector in* (1)*, it is possible to implement this second query approximately through finite differencing with the help of the first, using*

$$\mathbf{t} \approx \frac{\mathbf{x}^*(\mathbf{x}_1, \mathbf{x}_2, (1-\varepsilon), \varepsilon) - \mathbf{x}_1}{\varepsilon}$$

with a small but finite ε. In other words, one could in principle get away with only one primitive if necessary. At the same time, and as discussed in Section 1.1.2, we have found that in many cases, good ways to directly implement the TANGENT VECTOR *primitive exist, and there is no need for approximation.*

Remark 3 *The two primitives mentioned above are the only ones necessary to implement the operations discussed in the next section, exactly and without approximation. As such, they are truly primitive, but that does not mean that one could not come up with a larger set of operations geometry packages could provide, possibly in a more efficient way than when implemented based on the primitives. We did not pursue this idea any further, primarily because (i) we have not found it necessary in practice, and (ii) because these additional operations would have to be implemented for all of the different ways discussed in Section 1.1.2.*

1.1.2 Implementing primitives based on projections onto CAD geometries

For CAD geometries, any two points may fall *across* two different (non-overlapping) NURBS patches U_α and U_β. But there are more difficulties that make it difficult to implement the primitives based on pull back/push forward approaches:

1. NURBS patches do not always satisfy the requirements of a chart in the topological sense, i.e., they may be non-invertible in some points, and they may be non-smooth (i.e., contain corners and edges *within* a single U_α).
2. The metric ϕ_α that results from mapping u-v space to a NURBS patch U_α may be ill-formed: Its derivative may be close to zero near some points, and large at others — equally spaced points in u-v space (or points $\mathbf{x}^*(\mathbf{x}_1, \mathbf{x}_2; w_1, (1-w_1))$ for equally spaced values of w_1) would then be mapped to highly unevenly spaced points.
3. The evaluation of the pull-back $\phi_\alpha^{-1}(\mathbf{x})$ is computationally very expensive for NURBS patches.

As a consequence, *useful* implementations of our primitives will not be based on the concepts of differential geometry, but will rather be *projection-based*. Indeed, a possible implementation of the NEW POINT primitive for (multi-patch) CAD-based geometries is provided by the following algorithm: Given $\mathbf{x}_1, \ldots, \mathbf{x}_N$, $w_1, \ldots, w_N$, $N \geq 2$, define

$$\mathbf{x}^*\left(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N, w_1, w_2, \ldots, w_N\right) = P_{\mathrm{CAD}}\left(\sum_{n=1}^{N} w_n \mathbf{x}_n\right),$$

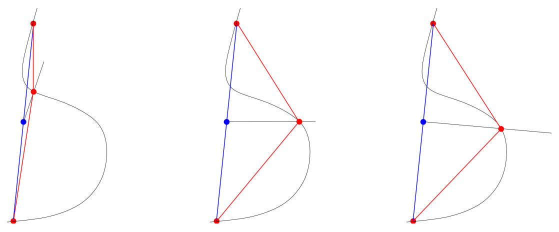


Figure 1: Comparison of three different implementations of the NEW POINT primitive for CAD geometries. The red end points (coarse vertices) of the blue line (coarse cell) form the inputs $\mathbf{x}_1, \mathbf{x}_2$ for which we want to use the NEW POINT primitive to find a new mid-point (i.e., $w_1 = w_2 = \frac{1}{2}$). The blue point is the average of the original vertices to be projected onto the curved geometry. Left: Projection normal to the geometry. Center: Projection in a direction chosen a priori. Right: Projection normal to coarse mesh.

where $P_{\text{CAD}}(\mathbf{x})$ is a projection of the point $\mathbf{x}$ onto the CAD surface (or curve). We will discuss various choices for the projection operator P_{CAD} below, given that the choice will influence the quality of the result. We note that projection-based strategies have been proposed in [22] and are used in a basic variant for a high-order finite element code in [15]. Many CAD programs, and specifically the OpenCASCADE library [21] that we use for the examples shown here, implement all of the operations necessary for the three approaches to implementing a projection discussed below.

Projection in a fixed direction The cheapest way to compute a projection is if the direction of the projection $\mathbf{d} \in \mathbb{R}^d$ is known a priori — for example, because we know that the surface in question has only minor variation from being horizontal. In that case, one might choose

$$\mathbf{x}^* \left(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N, w_1, w_2, \ldots, w_N\right) = I_{\text{CAD}} \left(\sum_{n=1}^{N} w_n \mathbf{x}_n, \mathbf{d} \right),$$

where $I_{\text{CAD}}(\mathbf{x}, \mathbf{d})$ is the intersection of the line $\mathbf{s}(t) = \mathbf{x} + s\mathbf{d}$ and the CAD surface (or curve). In other words, we first average all points $\mathbf{x}_i$ and then move the average back onto the surface along direction $\mathbf{d}$.

Projections taking into account the normal vector In more general situations, however, choosing a projection direction a priori is not possible. Rather, one needs to take into account the geometry of the CAD surface in the vicinity of the point to be projected, for example by considering the normal vectors to either the existing mesh, or to the surface.

The first of these options (shown in the right panel of 1) would use the following implementation:

$$\mathbf{x}^* \left(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N, w_1, w_2, \ldots, w_N\right) = I_{\text{CAD}} \left(\sum_{n=1}^{N} w_n \mathbf{x}_n, \mathbf{n} \right),$$

where the direction $\mathbf{n}$ is now (an approximation) to the normal vector of the area identified by the points $\mathbf{x}_n$. $\mathbf{n}$ is clearly defined if one only has d input points in d dimensional space; if there are more points — e.g., the four vertices of a face of a hexahedron in 3d — then one will want to define some useful approximate vector, e.g., the vector normal to the least squares plane that approximates the point locations. As is clear from the figure, if this implementation is chosen for mesh refinement, one generally ends up with refined meshes with cells of rather uniform sizes.

To use this approach, one needs to have at least d input points in d dimensions in order to define a unique direction normal to the existing points. But we also need to be able to use the NEW POINT primitive when finding a new midpoint for an edge in 3d. Thus, we need an additional condition to identify a unique direction among all of those perpendicular to the line connecting the existing edge end points. We do this by averaging the CAD surface normal at the vertices of the edge (both of which we know are on the surface), and then projecting it onto the edge's axial plane.

An alternative, often implemented in CAD tools but expensive to evaluate, is to use a direction vector $\mathbf{n}$ that is *perpendicular to the actual geometry*, rather than to the current mesh. This is shown in the left panel of Figure 1 and may lead to child cells of different size. Ultimately, however, once the mesh is already a good approximation to a surface, both of the approaches mentioned here will yield very similar results.

Implementing the tangent vector primitive for CAD surfaces In all three of the project-based cases above, the implementation of the TANGENT VECTOR primitive may be constructed using a finite differences approximation as already mentioned in Remark 2. A more accurate approach pushes forward the tangent vector at the pulled-back point.

1.2 Extending boundary representations into volumes

The previous sections only dealt with finding points and tangent vectors on a lower-dimensional surface, given points already on that surface. On the other hand, finite element codes typically use volume meshes for which the CAD geometry only provides information about the *boundary*. Thus, one still needs a way to extend this information into the *interior* of the domain — the importance of this step is apparent by looking at Figure 2.

A general mechanism for this task is based on transfinite interpolation [23]. A transfinite interpolation maps points from some reference space $\hat{\mathbf{x}} \in \hat{K} = [0,1]^d$ to points in real space $\mathbf{x}$ by a weighted sum of information on the geometry of the faces of the image of $\hat{K}$. For example, for a quadrilateral in two dimensions

$$\begin{aligned}\mathbf{x}(\hat{x}_1,\hat{x}_2) = {} & (1-\hat{x}_2)\mathbf{c}_0(\hat{x}_1) + \hat{x}_2\mathbf{c}_1(\hat{x}_1) + (1-\hat{x}_1)\mathbf{c}_2(\hat{x}_2) + \hat{x}_1\mathbf{c}_3(\hat{x}_2) \\ & - \left[(1-\hat{x}_1)(1-\hat{x}_2)\mathbf{x}_0 + \hat{x}_1(1-\hat{x}_2)\mathbf{x}_1 + (1-\hat{x}_1)\hat{x}_2\mathbf{x}_2 + \hat{x}_1\hat{x}_2\mathbf{x}_3\right].\end{aligned}$$

Here, $\mathbf{c_0}(\mathbf{s}), \mathbf{c_1}(\mathbf{s}), \mathbf{c_2}(\mathbf{s}), \mathbf{c_3}(\mathbf{s})$ are the four parameterized curves describing the geometry of the edges of the deformed quadrilateral and $\mathbf{x_0}, \mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}$ are the four vertices. The evaluation on each edge is done via the NEW POINT primitive $\mathbf{x}^*$, i.e., $\mathbf{c}_0(s) = \mathbf{x}^*(\mathbf{x}_0, \mathbf{x}_1, 1-s, s)$ and similarly for the other curves. If an edge is straight, then $\mathbf{c_0}(\mathbf{s}) = (\mathbf{1}-\mathbf{s})\mathbf{x_0} + \mathbf{s}\mathbf{x_1}$. Similar formulas extend to the three-dimensional case. The important point is that transfinite mappings exactly respect the geometry of the boundary, while extending it smoothly into the interior of the domain.

We visualize this approach using higher order mappings. We recall that finite element error estimates on curved cells depend on the product of a Sobolev norm of higher order derivatives of $\mathbf{F}_K$ times a norm of the derivatives of $\mathbf{F}_K^{-1}$ (see, e.g., [24] or [25, Sec. 3.3]), which we visualize by the ratio between the largest and smallest singular value of J_K. Figure 3 compares the singular values for two variants of computing the interior points from the surrounding 11 points per line, i.e., a mapping of polynomial degree 10, on a quarter of an annulus with inner and outer radii 0.5 and 1, respectively. If the weights for the interior points are derived from solving a Laplace equation in the reference coordinates, the representations becomes distored, as is visible from the point distributions. This leads to a ratio of up to 100 between the largest and smallest singular value of the Jacobian $J_K = \hat{\nabla}\mathbf{F}_K(\hat{\mathbf{x}})$, and theory suggests a break-down of convergence; in experiments, L_2 errors of the solution to the Laplacian converge at best at third order. Conversely, using weights from transfinite interpolation results in a minimal singular value of 0.5 throughout the whole domain in this example. The resulting point distribution with transfinite interpolation in this particular example is equivalent to an explicit polar description of the whole domain, but applicable to generic situations with optimal convergence if the coarse cells are valid. We note that these and similar concepts are established in high-order meshing, but with algorithms typically acting on the points of associated polynomial descriptions, see e.g. [26, 14, 27, 28, 29, 15, 13] and references therein, rather than the abstract definition used here.

We associate the transfinite interpolation with the initial ("coarse") mesh of a finite element computation: each coarse mesh cell is used to define the reference coordinate system $\hat{x}$, and this is kept fixed even after many generations of descendants. Interior edges between refined cells are then curved, ensuring high mesh quality, assuming that the initial coarse cells reasonably approximate the geometry. Some of the computations involved in this process can be expensive, and we have therefore implemented caches that mitigate the cost for the case of polynomial mappings.

1.3 Application examples

In the following, let us illustrate the ideas of the previous sections using concrete applications. In particular, we will show how the implementation of the two primitives affects the meshes one obtains for an industrial application (Section 1.3.1) and an example of how one can choose metrics to generate graded meshes. In addition, let us refer to Figure 2 for an illustration of the transfinite interpolation approach for extending surface descriptions to volume interiors.

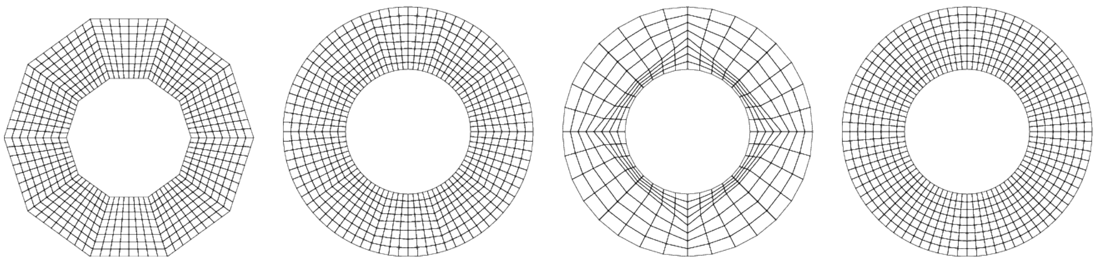


Figure 2: Illustration of the importance of taking geometry into account when refining meshes. Starting from a discretization of an annulus that contains ten coarse cells, we choose new points either ignoring the geometry description, i.e., computing the location of a new vertex by simply averaging the locations of the surrounding points (left), or choosing new points on boundary edges so that they have the correct radius from the origin, and new points of interior edges and cells as the Cartesian mean of the surrounding vertices (second from left). The latter procedure works well when the number of coarse elements is sufficient to resolve the geometry, but leaves some kinks in the grids, where one is still able to identify the original ten coarse cells. However, it may lead to very distorted grids if the coarse mesh is not fine enough, e.g., if the coarse mesh consisted of only four cells (second from right). The ideal case (right) is independent of the number of coarse cells that one may start with, and it exploits full knowledge of the underlying geometry. In this case, this is done by choosing all new points on edges and cells so that they average the *radius and angle* of the adjacent points.

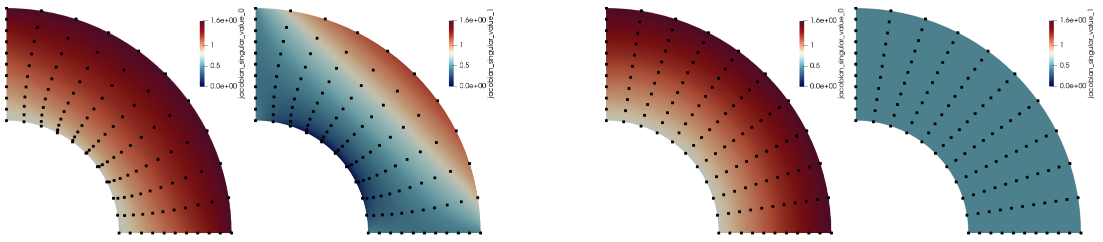


Figure 3: Illustration of the two singular values of $J_K = \hat{\nabla}\mathbf{F}_K(\hat{\mathbf{x}})$ for a quarter of a two-dimensional annulus; here, $\mathbf{F}_K$ is a polynomial mapping of a single element of degree 10. The ratio of the largest over the smallest singular value appears in the interpolation error estimate of the Bramble–Hilbert lemma, and consequently also in all error estimates for partial differential equations. Left two panels: Maximal and minimal singular value for point placement based on Laplace smoothing. Right two panels: Maximal and minimal singular value for point placement using a transfinite interpolation (see the main text for interpretation). The figures also include the positions of 11^2 points equidistantly placed onto the reference cell and then mapped by $\mathbf{F}_K$ to the cell K shown here, to illustrate the distortion.

1.3.1 Surface meshes described by CAD geometries

As discussed in detail in previous sections, CAD surfaces consist of patches that are the images of simpler domains in a two-dimensional u-v space. Let us first consider a case where the geometry is described by a single, albeit rather complex, patch. The issue in even this simplified case is that a patch can be parameterized in many different ways, not all of which imply a more or less constant metric. Thus, it is unwise — although very common — to generate a mesh in u-v space to obtain the corresponding three-dimensional surface grid, even though this of course has the advantage of generating nodes directly on the desired surface. Yet, the resulting mesh will generally have cells of rather unequal sizes and may show other severe deformations.

To illustrate how our approach can be used to generate better meshes, we will use an industrial application that involves meshing a single parametric patch describing the bow portion of one side of the DTMB 5415 ship hull, containing also a sonar dome. The presence of several convex and concave high curvature regions makes such a geometry a particularly meaningful example.

Figures 4–6 show results for this model geometry with the three projection strategies.

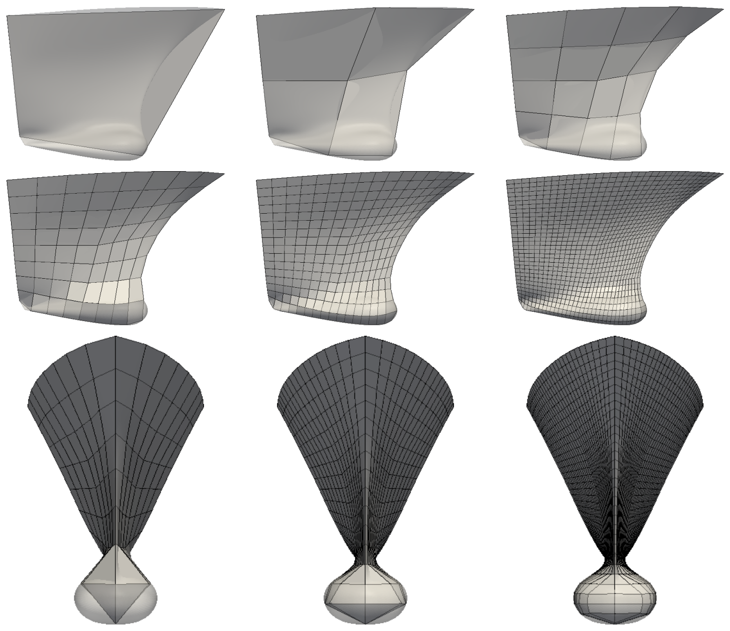


Figure 4: Directional projection strategy with a horizontal direction of projection perpendicular to the axis of symmetry. The first two rows show side views of the coarse grid and grids obtained from five successive refinements. The last row shows a front view of the same grids shown in the second row. This strategy produces uniformly distributed cells away from areas where the projection direction is close to the tangent to the shape (namely, at the bottom of the shape as well as the front of the bulb).

The directional projection strategy with a horizontal direction of projection (Figure 4) generally produces high quality meshes except in those places where the geometry is tangent to the projection direction — i.e., in particular at the front of the bulb as well

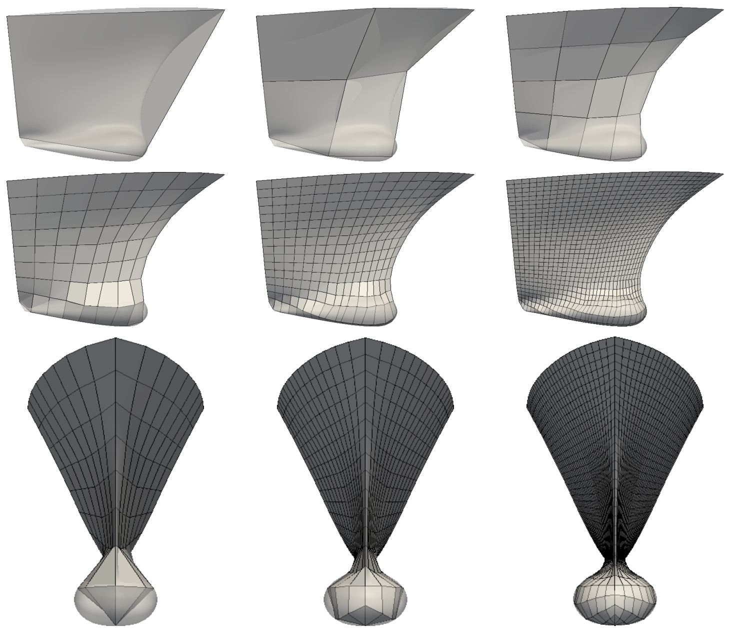


Figure 5: Normal to mesh projection strategy. If the geometry does not intersect the direction normal to the existing points, then the closest point on the shape to the original point — typically lying on the shape boundary — is selected. Panels as in Figure 4. This strategy produces uniformly distributed cells in all cases.

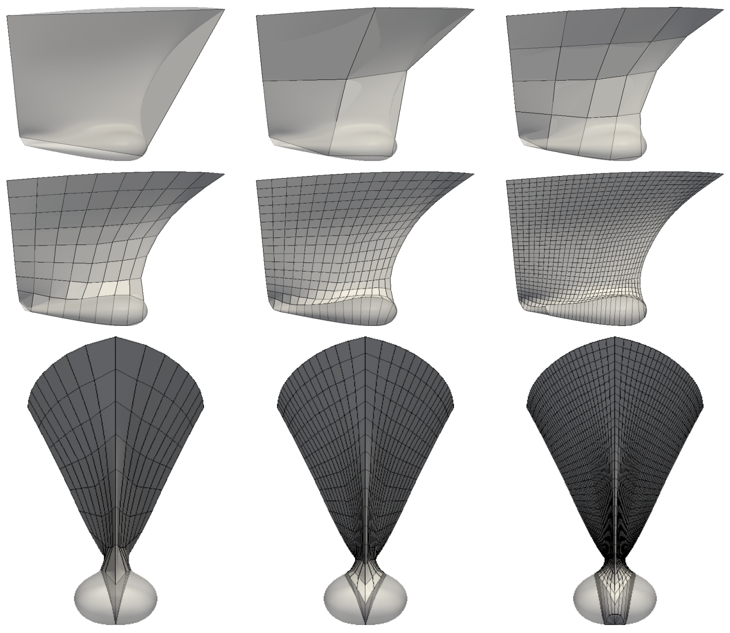


Figure 6: Normal to surface projection strategy. In cases where more than one surface normal projection is available, the closest of them is selected. If the shape is composed by several sub-shapes, the projection is carried out onto every sub-shape and the closest projection point is selected. Panels as in Figure 4. This strategy is unable to produce well-shaped cells in areas of large curvature.

as the bottom. In contrast, using the direction normal to the existing points (Figure 5) generates high quality meshes everywhere. Finally, the option to use a surface normal instead of a mesh normal vector (Figure 6) is not only expensive to compute, but here yields meshes that are grossly distorted wherever the geometry has large curvature (i.e., around the bulb); this might also have been expected from the left panel of Figure 1 that shows a similar effect.

1.3.2 Refinement strategy based on local maximum curvature

The case of multi-patch geometries is more complicated as small gaps or superimpositions are typically present between neighboring patches. Meshes directly generated from this parameterization will therefore often not be "water-tight". On the other hand, one can generate a coarse mesh by hand or by software that simply starts with a few points on the surface that are then connected to cells without taking into account the subdivision into patches; such a mesh can then be refined hierarchically to obtain a mesh of sufficient density.

In the following, let us show how we can use the results of the previous section towards building meshes for an entire ship hull; we will also show how additional information can be extracted from CAD tools to drive the refinement strategy on CAD based geometries. To this end, Figure 7 depicts a CAD model of a Kriso KCS ship hull — a common benchmark for CFD applications of naval architecture [30]. In this production-like CAD model the patches are not connected in a water-tight fashion and the surface parametrization is not continuous at patch junctions. These defects prevent most mesh generators from obtaining a closed grid.

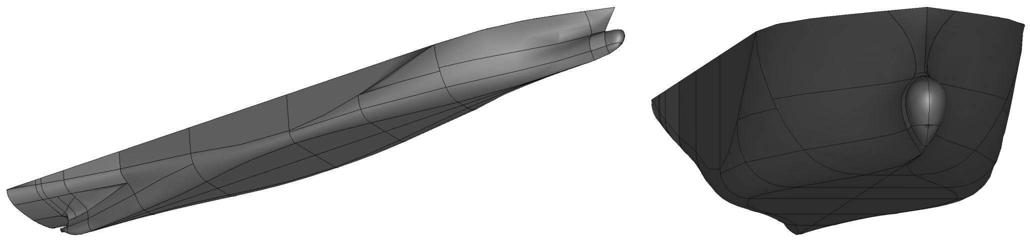


Figure 7: Overall (left) and bow (right) view of the CAD model of a ship hull. The model is composed of approximately 120 parametric patches, delimited by black lines.

We start from a minimal initial surface grid composed of about 40 cells (top panel of Figure 8) and refine it a number of times using the strategy where we project in a direction normal to the existing points. Refinement of the initial grid starts with an anisotropic refinement step in which cells with an aspect ratio larger then a threshold $\lambda_{\max}$ are cut along their most elongated direction. We then refine the resulting quadrilateral mesh adaptively, using an estimate of the local curvature of the CAD surface as a criterion. This estimate is obtained exploiting a technique essentially identical to the one used in the "Kelly" error estimator [31, 32]. Elements in areas with higher curvature will have larger jumps of the (cell) normal vectors across cell boundaries. The bottom panel of Figure 8 shows the final grid generated.

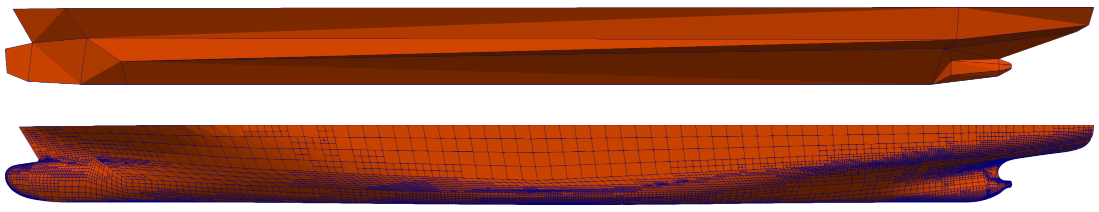


Figure 8: Initial (top) and final (bottom) surface grids on the Kriso KCS hull, with 40 and ˜ 11,500 quadrilateral cells, respectively.

Despite the fact that the original CAD surface is composed of several unconnected parametric patches, the projection procedure is able to find, for each of the grid nodes, the best projection among those obtained onto individual patches. As a result, the final mesh is water-tight, and independent of the CAD surface parameterization and patch distribution. In addition, the adopted refinement strategy distributes a larger number of new nodes in high curvature regions, ensuring a uniform quality of approximation of the geometry. We show this in more detail in Figure 9, illustrating the bow and stern portions of the final grid. Finally, it is worth pointing out that for the generation of the grids portrayed in Figure 8 and Figure 9, no smoothing stage was carried out in between refinements to enhance mesh quality. Yet, the projection in a direction normal to the existing points allows for retaining the quality of the original coarse grid across more than 10 levels of refinement without any additional adjustment.

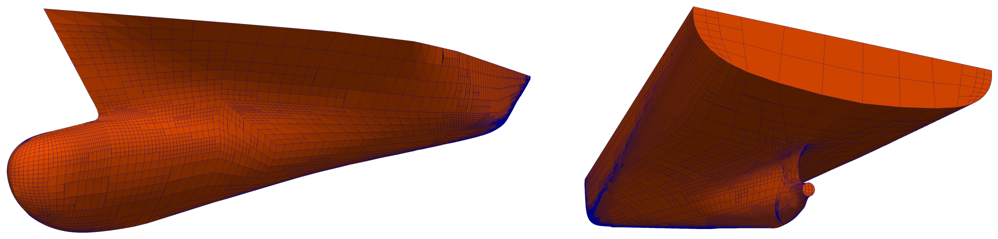


Figure 9: Bow (left) and stern (right) details of the adaptively refined surface grid on the Kriso KCS hull from Figures 7 and 8. The adaptive refinement strategy results in finer cells in high curvature regions, ensuring uniform approximation of the geometry. In addition, the grid is independent of the non-sharp edges separating the 120 parametric patches composing the underlying CAD model. In the final mesh, hanging nodes are placed on the underlying geometry, leading to a non-watertight mesh. However, these artifacts are easily removed by enforcing continuity of the geometry.

2 CAD geometry aware BEM solver for free surface flows

2.1 Fully nonlinear potential model

In this work, we are only considering the motion of a ship advancing at constant speed in calm water. Thus, we choose to solve the problem in a *global*, steadily translating reference frame $\widehat{XYZ}$, which is moving according to the constant horizontal velocity of the boat $\boldsymbol{V}_\infty = (V_\infty, 0, 0)$. Thus, the X axis of the reference frame is aligned with $\boldsymbol{V}_\infty$, the Z axis is directed vertically (positive upwards), while the Y axis is directed laterally (positive port side).

Under the assumptions of irrotational flow, inviscid fluid, and simply connected domain Ω, the velocity field $\boldsymbol{v}(\boldsymbol{X}, t)$ admits a representation through a scalar potential function $\Phi(\boldsymbol{X}, t)$, namely

$$\boldsymbol{v} = \boldsymbol{\nabla}\Phi = \boldsymbol{V}_\infty \cdot \boldsymbol{X} + \phi \qquad \forall\ \boldsymbol{X} \in \Omega(t), \tag{2}$$

in which $\phi(\boldsymbol{X}, t)$ is the so called *perturbation potential.* In such case, the equations of motion simplify to the unsteady Bernoulli equation and to the Laplace equation for the perturbation potential:

$$\frac{\partial \phi}{\partial t} + \frac{1}{2}|\boldsymbol{\nabla}\phi + \boldsymbol{V}_\infty|^2 + \frac{p - p_a}{\rho} - \boldsymbol{g} \cdot \boldsymbol{X} = C(t) \qquad \text{in } \Omega(t) \tag{3a}$$

$$\Delta\phi = 0 \qquad \text{in } \Omega(t) \tag{3b}$$

where $C(t)$ is an arbitrary function of time, and $\boldsymbol{g} = (0, 0, -g)$, is the gravity acceleration vector, directed along the z coordinate axis. In this framework, the unknowns of the problem ϕ and p are uncoupled, and it's possible to recover the pressure by postprocessing the solution of the Poisson problem (3b) via Bernoulli's equation (3a). Thus, the governing equation of our model is the Laplace equation. Such equation is complemented by non penetration boundary conditions on the hull surface $\Gamma^b(t)$ and on the bottom of the basin $\Gamma^{bot}(t)$, and by homogeneous Neumann boundary conditions on the truncation boundaries $\Gamma^{far}(t)$ of the numerical domain. On the water free surface $\Gamma^w(t)$, we employ the kinematic and dynamic fully nonlinear boundary conditions expressed in semi-Lagrangian form, which respectively read

$$\frac{\delta\eta}{\delta t} = \frac{\partial\phi}{\partial z} + \boldsymbol{\nabla}\eta \cdot (\boldsymbol{w} - \boldsymbol{\nabla}\phi - \boldsymbol{V}_\infty) \qquad \text{in } \Gamma^w(t) \tag{4}$$

$$\frac{\delta\phi}{\delta t} = -g\eta + \frac{1}{2}|\boldsymbol{\nabla}\phi|^2 + \boldsymbol{\nabla}\phi \cdot (\boldsymbol{w} - \boldsymbol{\nabla}\phi - \boldsymbol{V}_\infty) \qquad \text{in } \Gamma^w(t). \tag{5}$$

The former equation expresses the fact that a material point moving on the free surface will stay on the free surface — here assumed to be the graph of a single valued function $\eta(X, Y, t)$ of the horizontal components X and Y of the position vector $\boldsymbol{x}$. The latter condition represents a manipulation of Bernoulli's equation (3a), under the assumption of constant atmospheric pressure on the water surface. This peculiar form of the fully nonlinear boundary conditions was proposed by Beck [33]. Equation (4) allows for the computation of the vertical velocity of markers which move on the water free surface with a prescribed horizontal speed (w_X, w_Y). Equation (5) is used to obtain the velocity potential values in correspondence with such markers. The resulting vector $\boldsymbol{w} = (w_X, w_Y, \frac{\delta\eta}{\delta t}) = \dot{\boldsymbol{X}}$ is the time derivative of the position of the free surface markers. In this work, such free surface markers are chosen as the free surface nodes of the computational grid. To avoid an undesirable mesh nodes drift along the water stream, the markers arbitrary horizontal velocity is set to 0 along the X direction. The Y component of the water nodes in contact with the ship — which is moved according with the computed linear and angular displacements — is chosen so as to keep such nodes on the hull surface. As for the remaining water nodes, the lateral velocity value is set to preserve mesh quality.

2.2 Three dimensional hull rigid motions

In this work, the ship hull is assumed to be a rigid body. To study its motions, we employ a second, *hull-attached* reference frame $\widehat{xyz}$, which follows the hull in its translations and rotations. The center of such reference frame will be located in correspondence with the ship center of gravity, which in the global reference frame reads $\boldsymbol{X}^G(t) = X^G(t)\boldsymbol{e}_X + Y^G(t)\boldsymbol{e}_Y + Z^G(t)\boldsymbol{e}_Z$, where $\boldsymbol{e}_X$, $\boldsymbol{e}_Y$, $\boldsymbol{e}_Z$ are the unit vectors along the global system axes.

The rotation matrix $R(t)$ is used to pass from the coordinates of a point $\boldsymbol{x}$ written in the hull-attached reference frame, to those in the global frame $\boldsymbol{X}$, namely

$$\boldsymbol{X}(t) = R(t)\boldsymbol{x} + \boldsymbol{X}^G(t). \tag{6}$$

The global frame velocity of a point having coordinates $\boldsymbol{x}$ in the hull-attached frame is obtained as

$$\boldsymbol{V}^{\text{hp}}(t) = \boldsymbol{\omega}(t) \times \boldsymbol{x} + \dot{\boldsymbol{X}}^G(t), \tag{7}$$

where $\boldsymbol{\omega}$ is the angular velocity vector, the components of which specify the rotational speed of the hull about the global frame axes X, Y and Z respectively.

Equations (6) and (7) imply that once $\boldsymbol{X}^G(t)$, $R(t)$, and $\boldsymbol{\omega}(t)$ are known at time t, the position and velocity of each point of the hull can be computed. For this reason, writing the time evolution equations for each of such quantities will be sufficient in order to determine the hull dynamics. In the next sections, we will present the evolution equations used in this work.

2.2.1 Linear momentum conservation

The evolution equation for $\boldsymbol{X}^G(t)$ is obtained via the linear momentum conservation equation, which in the case of our hydrodynamics simulation framework reads

$$m_s\ddot{\boldsymbol{X}}^G(t) = m_s\boldsymbol{g} + \boldsymbol{F}^w(t). \tag{8}$$

In Equation (8), m_s is the mass of the ship, while the hydrodynamic force vector $\boldsymbol{F}^w(t)$ is in principle obtained as the sum of the pressure and viscous forces on the hull, propeller and appendages.

2.2.2 Angular momentum conservation

The evolution equation for $\boldsymbol{\omega}$ is obtained writing the angular momentum conservation, namely

$$R(t)I^G R(t)^T\dot{\boldsymbol{\omega}}(t) + \boldsymbol{\omega}(t) \times R(t)I^G R(t)^T\boldsymbol{\omega}(t) = \boldsymbol{M}^w(t), \tag{9}$$

where I^G is the matrix of inertia of the ship in the hull-attached reference frame, and hydrodynamic moment vector $\boldsymbol{M}^w(t)$ is the sum of the moment about the ship center of gravity of the pressure and viscous forces on hull, propeller and appendages.

2.2.3 Rotation matrix and hull quaternions

To write an evolution equation for R, we first the introduce the angular velocity tensor associated to $\boldsymbol{\omega}$, namely

$$\omega(t) = \begin{bmatrix} 0 & -\omega_Z(t) & \omega_Y(t) \\ \omega_Z(t) & 0 & -\omega_X(t) \\ -\omega_Y(t) & \omega_X(t) & 0 \end{bmatrix}. \tag{10}$$

Note that tensor $\omega(t)$ will act on a vector $\boldsymbol{u} \in \mathbb{R}^3$ as if the $\boldsymbol{\omega}\times$ operator were applied to $\boldsymbol{v}$:

$$\boldsymbol{\omega} \times \boldsymbol{u} = \omega\boldsymbol{u}. \tag{11}$$

Making use of such tensor, an evolution equation for the rotation matrix R reads

$$\dot{R} = \omega R, \tag{12}$$

which can be advanced in time to obtain the components of R and close the equations of motions of a rigid body in three dimension. Yet, in common the practice of rigid body simulations, direct numerical integration of Equation (12) is avoided. The most important reason for this, is related to numerical drift. If we in fact keep track of the orientation of a rigid body integrating Equation (12), numerical error will build up in the entries of $R(t)$, so that it will no longer be a rotation matrix, losing its properties of orthogonality and of having determinant equal to 1. Physically, the effect would be that applying $R(t)$ to a body would cause a skewing effect.

A better way to represent the orientation of a rigid body in three dimensions (even with large rotations) is represented by the use of *unit quaternions* (see Shoemake [34]). For our purposes, quaternions can be considered as a particular type of four element vector, normalized to unit length. If we indicate the quaternion $\boldsymbol{q} = s + v_X \boldsymbol{e}_X + v_Y \boldsymbol{e}_Y + v_Z \boldsymbol{e}_Z$ as $[s, \boldsymbol{v}]$, the internal product of two quaternions $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$ is defined as

$$\boldsymbol{q}_1\boldsymbol{q}_2 = [s_1, \boldsymbol{v}_1]\,[s_2, \boldsymbol{v}_2] = [s_1 s_2 - \boldsymbol{v}_1 \cdot \boldsymbol{v}_2\,,\, s_1 \boldsymbol{v}2 + s_2 \boldsymbol{v}1 + \boldsymbol{v}_1 \times \boldsymbol{v}_2]\,. \tag{13}$$

The norm of a quaternion $\boldsymbol{q}$ is defined as $||\boldsymbol{q}|| = \sqrt{s^2 + v_X^2 + v_Y^2 + v_Z^2}$. Unit quaternions can be used to represent rotations in a three dimensional space. In fact, given a quaternion $\boldsymbol{q} : ||\boldsymbol{q}|| = 1$, we can obtain the corresponding rotation matrix as

$$R = \begin{bmatrix} 1 - 2v_Y^2 - 2v_Z^2 & 2v_X v_Y - 2sv_Z & 2v_X v_Z + 2sv_Y \\ 2v_X v_Y + 2sv_Z & 1 - 2v_Y^2 - 2v_Z^2 & 2v_Y v_Z - 2sv_X \\ 2v_X v_Z - 2sv_Y & 2v_Y v_Z + 2sv_X & 1 - 2v_Y^2 - 2v_Z^2 \end{bmatrix}. \tag{14}$$

Finally, the equation needed to describe the time evolution for the hull quaternion $\boldsymbol{q}(t)$ is

$$\dot{\boldsymbol{q}}(t) = \frac{1}{2}\boldsymbol{\Omega}(t)\boldsymbol{q}(t), \tag{15}$$

where $\boldsymbol{\Omega}(t) = [0, \boldsymbol{\omega}(t)]$ is the quaternion associated with the angular velocity vector $\boldsymbol{\omega}(t)$. As quaternions only have four entries, there only is one extra variable used to describe the three degrees freedoms of a three dimensional rotation. A rotation matrix instead employs nine parameters for the same three degrees of freedom; thus, the quaternions present far less redundancy than rotation matrices. Consequently, quaternions experience far less numerical drift than rotation matrices. The only possible source of drift in a quaternion occurs when the quaternion has lost its unit magnitude. This can be easily corrected by periodically renormalizing the quaternion to unit length.

2.3 Discretization and numerical solution

In this section, we will briefly describe the spatial discretization method based on the boundary integral formulation of the Laplace equation. In addition, we will present the Differential Algebraic Equation system coupling the fluid dynamics and hull rigid motions equations, and discuss its solution strategy.

2.3.1 Boundary integral formulation

While Equation (3) is time dependent and defined in the entire domain $\Omega(t)$, we are really only interested in its solution on its boundary $\Gamma(t)$, in particular on the unknown free surface part of the boundary, and on the ship hull $\Gamma^b(t)$, where we would like to recover the pressure distribution by postprocessing Bernoulli's equation (3a).

At any time instant $\bar{t}$ we want to compute ϕ satisfying

$$
\begin{aligned}
-\Delta\phi &= 0 && \text{in } \Omega(\bar{t}) && \text{(16a)}\\
\phi &= \overline{\phi} && \text{on } \Gamma^w(\bar{t}) && \text{(16b)}\\
\phi_n &= \left(\boldsymbol{V}^{\text{hp}} - \boldsymbol{V}_\infty\right)\cdot \boldsymbol{n} && \text{on } \Gamma^b(\bar{t}) && \text{(16c)}\\
\phi_n &= 0 && \text{on } \Gamma^{bot}(\bar{t}) && \text{(16d)}\\
\phi_n &= 0 && \text{on } \Gamma^{far}(\bar{t}) && \text{(16e)}\\
& && && \text{(16f)}
\end{aligned}
$$

where $\overline{\phi}$ is the potential on the free surface at the time $\bar{t}$. This is a purely spatial boundary value problem, in which time appears only through boundary conditions and through the shape of the time dependent domain.The solution of problem (16) is then used to evaluate the right hand side of Equation (4) and Equation (5), which are integrated over time to obtain the new position of the free surface markers and the corresponding potential field values.

We call G the *free space Green's function*, i.e., the function

$$G(\boldsymbol{r}) = \frac{1}{4\pi|\boldsymbol{r}|},$$

which is the distributional solution of

$$
\begin{aligned}
&-\Delta G(\boldsymbol{X}-\boldsymbol{X}_0) = \delta(\boldsymbol{X}_0) && \text{in } \mathbb{R}^3\\
&\lim_{|\boldsymbol{X}|\to\infty} G(\boldsymbol{X}-\boldsymbol{X}_0) = 0,
\end{aligned}
\tag{17}
$$

where $\delta(\boldsymbol{X}_0)$ is the Dirac distribution centered in $\boldsymbol{X}_0$.

If we select $\boldsymbol{X}_0$ to be in $\Omega(t)$, multiply the Laplace equation by G, and use the defining property of the Dirac delta and the second Green identity, we obtain

$$
\begin{aligned}
\phi(\boldsymbol{X}_0,t) = \int_{\Omega(t)} &\left[-\left(\Delta G(\boldsymbol{X}-\boldsymbol{X}_0)\right)\phi(\boldsymbol{X},t)\right] \mathrm{d}\Omega = \\
&\int_{\Gamma(t)} \left[(\boldsymbol{\nabla}\phi(\boldsymbol{X},t)\cdot\boldsymbol{n})G(\boldsymbol{X}-\boldsymbol{X}_0) - (\boldsymbol{\nabla}G(\boldsymbol{X}-\boldsymbol{X}_0)\cdot\boldsymbol{n})\phi(\boldsymbol{X},t)\right] \mathrm{d}\Gamma.
\end{aligned}
$$

In the limit for $\boldsymbol{X}_0$ touching the boundary $\Gamma(t)$, the integral on the right hand side will have a singular argument, and should be evaluated according to the Cauchy principal value. This process yields the so called *Boundary Integral Equation* (BIE)

$$\alpha\phi = \int_{\Gamma(t)} \left[\phi_n G - \frac{\partial G}{\partial n}\phi\right] \mathrm{d}\Gamma \qquad \text{on } \Gamma(t), \tag{18}$$

where $\alpha(\boldsymbol{X}, t)$ is the fraction of solid angle 4π with which the domain $\Omega(t)$ is seen from $\boldsymbol{X}$.

With Equation (18), the continuity equation has been reformulated as a boundary integral equation of mixed type defined on the moving boundary $\Gamma(t)$, where the main ingredients are the perturbation potential $\phi(\boldsymbol{X}, t)$ and its normal derivative $\phi_n(\boldsymbol{X}, t)$.

2.3.2 Iso-parametric spatial discretization

The spatial discretization of the Laplace problem, has been carried out making use of the classes of the C++ open source library for finite elements implementation DEAL.II (Bangerth et al. [35, 36]). A detailed description of how such classes are employed for the implementation of the present Boundary Element Method can be found in Mola et al. [37]. In such framework, we approximate the geometry of the domain boundary by means of bilinear quadrilateral panels. We define the Lagrangian shape functions $N_l(u, v) \quad l = 1, \dots, 4$ on the reference panel, which allow us to introduce a local parametrization of the k-th panel as

$$\boldsymbol{X}_k(u, v, t) := \sum_{l=1}^{4} \boldsymbol{X}^{k_l}(t) N_l(u, v) \qquad u, v \in [0, 1]^2, \tag{19}$$

where the weights are the positions of the nodes in the current domain $\Gamma_h(t)$, and k_l is the *local to global* numbering index which identifies the 4 basis functions φ^{k_l} which are different from zero on the k-th panel.

The global basis functions $\varphi^i(\boldsymbol{X})$ can be identified and evaluated on each panel K via their local parametrization as

$$\varphi^i_k(u, v) := \varphi^i(\boldsymbol{X}_k(u, v)) = \sum_{l=1}^{4} \delta_{i\,k_l} N_l(u, v), \qquad \delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases} \tag{20}$$

At this point, a local representation of $\phi(\boldsymbol{X}^k(u, v, t), t)$ and of its normal derivative on the k-th panel are available as

$$\phi_k(u, v, t) = \sum_{l=1}^{4} \phi^{k_l}(t) N_l(u, v) \qquad \phi_{n\,k}(u, v, t) = \sum_{l=1}^{4} {\phi_n}^{k_l}(t) N_l(u, v),$$

where $\phi^{k_l}, {\phi_n}^{k_l}$ $l = 1, \dots, 4$ are the nodal values of the potential and of its normal derivative in panel k.

2.3.3 Collocation boundary element method

With the iso-parametric representation, we write the boundary integral equation for each support point $\boldsymbol{X}^i$, $i = 1, \dots, N_V$, and finally recast the discrete version of the boundary integral equation as

$$[\alpha]\{\phi\} + [N]\{\phi\} = [D]\{\phi_n\} \tag{21}$$

where we have used the following notation

- $\{\phi\}$ and $\{\phi_n\}$ are the vectors containing the potential and its normal derivative node values, respectively;
- $[\alpha]$ is a diagonal matrix composed by the $\alpha(\boldsymbol{X}_i(t))$ coefficients;
- $[D]$ and $[N]$ are the Dirichlet and Neumann matrices respectively whose elements are

$$D_{ij} = \sum_{k=1}^{M} \int_{\hat{K}} G(\boldsymbol{X}_i(t) - \boldsymbol{X}^k(u,v,t))\varphi_k^j(u,v)J^k(u,v,t)\,\mathrm{d}u\,\mathrm{d}v \tag{22}$$

$$N_{ij} = \sum_{k=1}^{M} \int_{\hat{K}} \frac{\partial G}{\partial n}(\boldsymbol{X}_i(t) - \boldsymbol{X}(u,v,t))\varphi_k^j(u,v)J^k(u,v,t)\,\mathrm{d}u\,\mathrm{d}v, \tag{23}$$

$J^k(u,v,t)$ being the determinant of the first fundamental form on the k-th of the M panels composing the computational grid.

The numerical evaluation of the panel integrals appearing in Equations (22) and (23) needs some special treatment, due to the presence of the singular kernels $G(\boldsymbol{Y} - \boldsymbol{X})$ and $\frac{\partial G}{\partial n}(\boldsymbol{Y} - \boldsymbol{X})$. Whenever $\boldsymbol{y}$ is not a node of the integration panel, the integral argument is not singular, and standard Gauss quadrature formulas can be used. If $\boldsymbol{y}$ is a node of the integration panel, the integral kernel is singular and special quadrature rules are used, which remove the singularity by performing an additional change of variables (see, for example, Lachat and Watson [38]).

2.3.4 Time discretization

The time dependent boundary value problem composed by Laplace equation and by the kinematic and dynamic boundary condition, along with the hull rigid motion equations can be recast in the following form

$$F(t, y, y') = 0, \tag{24}$$

where we grouped the variables of the system in the vector y:

$$y = \left\{ \begin{array}{c} \{\boldsymbol{X}\} \\ \{\phi\} \\ \{\phi_n\} \\ \left\{\dot{\boldsymbol{X}}^G\right\} \\ \left\{\boldsymbol{X}^G\right\} \\ \{\boldsymbol{\omega}\} \\ \{\boldsymbol{q}\} \end{array} \right\}. \tag{25}$$

Here, $\{\boldsymbol{X}\}$, $\{\phi\}$ and $\{\phi_n\}$ represent the vectors containing nodal coordinates, potential and potential normal gradient values respectively; $\left\{\dot{\boldsymbol{X}}^G\right\}$ and $\left\{\boldsymbol{X}^G\right\}$ are the hull baricenter velocity and position vector respectively; finally $\{\boldsymbol{\omega}\}$ and $\{\boldsymbol{q}\}$ are the hull angular velocity and quaternion vector. Note that the three second order differential equations for the hull baricenter coordinates $\left\{\boldsymbol{X}^G\right\}$ have been here recast into a set of six first order differential equations.

Equation (24) represents a system of nonlinear differential algebraic equations (DAE), which we solve using the IDA package of the SUNDIALS OpenSource library (Hindmarsh et al. [39]). The integration of such DAE system is performed through a variable-order, variable-coefficient BDF (backward difference formula), which reads

$$\sum_{i=0}^{q} \alpha_{n,i} y_{n-i} = h_n \dot{y}_n, \tag{26}$$

where y_n and $\dot{y}_n$ are the computed approximations to $y(t_n)$ and $y'(t_n)$, respectively, and the step size is $h_n = t_n - t_{n-1}$. The coefficients $\alpha_{n,i}$ are uniquely determined by the order q, and the history of the step sizes. The application of the BDF scheme to the DAE system results, at each time step, in a nonlinear algebraic system, solved by means of Newton iteration.

2.3.5 Treatment of CAD surfaces and fully automated mesh generation

In a typical Computer-Aided Design (CAD) model, several parametric surfaces are patched together to compose the whole ship surface. At the industrial level, there is no requirement that such patches are logically connected one to each other, nor that the full hull surface is or even continuous. Most of CAD shapes indeed present several small gaps or overlaps between each surface composing them. A small tolerance can in fact be set during the CAD model generation, to control the dimension of such imperfections in a way that makes them negligible once the actual hull is crafted. Yet, the presence of disconnected and overlapping patches impairs in most cases the generation of computational grids for fluid mechanics simulations purposes. Most solvers for fluid dynamics equations require in fact the generation of a three dimensional grid in the volume of fluid surrounding the hull surface. Such domain is in most cases obtained via boolean subtraction of the hull volume from a large box of fluid surrounding it. Clearly, in presence of gaps and overlaps between the hull surface patches, both delimiting the hull volume and using it to perform boolean operations is extremely difficult. One of the most important advantages of the Boundary Integral Formulation adopted, is that only the boundary of the three dimensional domain needs to be discretized. Thus, the reduced complexity of the superficial computational grids needed has been exploited in this work to achieve full automation of the mesh generation process, regardless of the number of number of surfaces composing the hull model, and the continuity of their connections. In this work, we made use of the Open CASCADE Community Edition (OCE) open source library to import and interrogate CAD models in the mesh generation module of the BEM solver developed. While building the grid, a certain number of fundamental geometric functions are in fact needed to properly place the newly created nodes on the hull surface and on its edges. To this end, we implemented a series of wrappers that use the OCE functions to provide the mesh handler with the geometrical tools needed. A grid composed of a very small number of quadrilaterals is initially generated. The points are placed at strategic locations on the hull intersections with its symmetry plane and with the undisturbed free surface. At this stage, the hull is discretized only in two cells per each side.

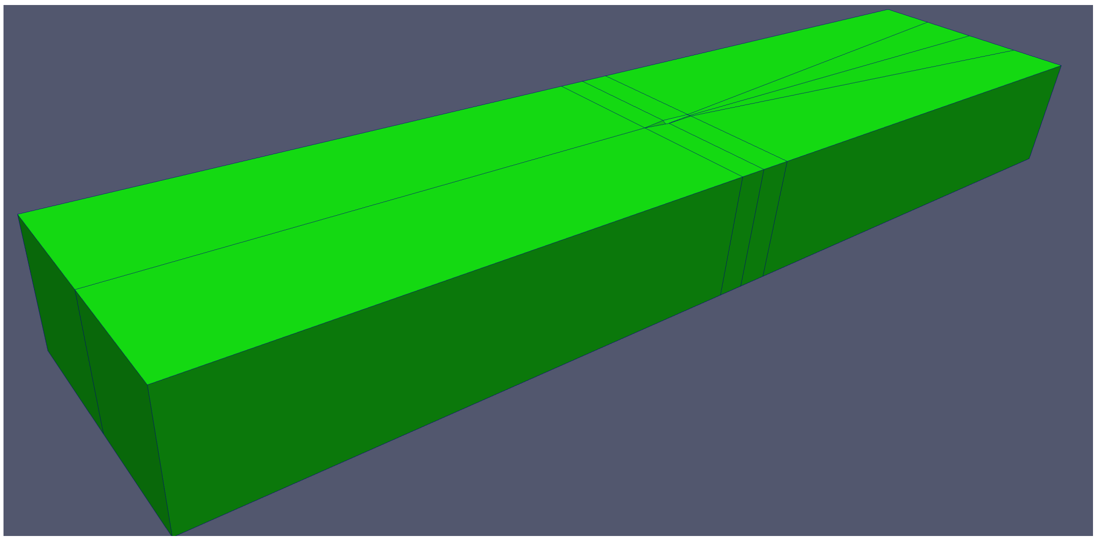


Figure 10: The very first grid created using the points of intersection between the undisturbed free surface and the CAD model of the hull. The domain extends for $18 \times L_{\mathrm{pp}}$ in longitudinal direction, $4 \times L_{\mathrm{pp}}$ in lateral direction and $2 \times L_{\mathrm{pp}}$ in vertical direction.

Figure 10 shows a view of the initial mesh. Starting from such initial mesh, the cells are hierarchically refined to improve the level of approximation of the original CAD geometry. The new nodes are generated onto the quadrilateral cells composing the grid, and then projected on the hull surface making use of a set of surface projectors implemented through OCE (see Mola et al. [40] and Dassi et al. [41] for a more detailed description). To generate the grid, initial cycles of uniform refinement are followed by a final number of local adaptive refinement cycles based on an error estimator represented by the distance between the center cell and its projection on the boat surface. Finally, during the simulation at fixed intervals in time, the simulation is stopped and the surface gradient of the finite element approximation of η is post-processed. This provides a quantitative estimate of the cells in which the approximation error may be higher. In particular, for each cell K of our triangulation we compute the quantity

$$\tau_K^2 := \frac{h}{24} \int_{\partial K} \left[\boldsymbol{\nabla}_s \eta \cdot \boldsymbol{n}_{\partial K}\right]^2 \, \mathrm{d}\gamma, \tag{27}$$

where $[\boldsymbol{\nabla}_s \eta \cdot \boldsymbol{n}_{\partial K}]$ denotes denotes the jump of the surface gradient of η across the edges of the grid element K. The vector $\boldsymbol{n}_{\partial K}$ is perpendicular to both the cell normal $\boldsymbol{n}$ and to the boundary of the element K, and h is the diameter of the cell itself.

By a practical standpoint, τ_K represents an estimate of the jumps of gradients across the edges of each quadrilateral cell. The higher these values are, the smaller the cells should become. The a posteriori error estimator described is a modification of the gradient recovery error estimator by Kelly et al. [31] and Gago et al. [32]. Its choice was mostly motivated by its simplicity.

The vector containing the cell error estimates τ_K is ordered, and a fixed fraction of the cells with the highest and lowest errors are flagged for refinement and coarsening. The computational grid is then refined, ensuring that any two neighboring cells differ for at most one refinement level. Standard interpolation is used to transfer all finite dimensional solutions from one grid to another. The resulting computational grid is non conformal. At each hanging node, all the dimensional fields computed are constrained to be continuous. This results in a set of algebraic constraints which are introduced into the DAE system in correspondence with all the degrees of freedom associated with the hanging nodes.

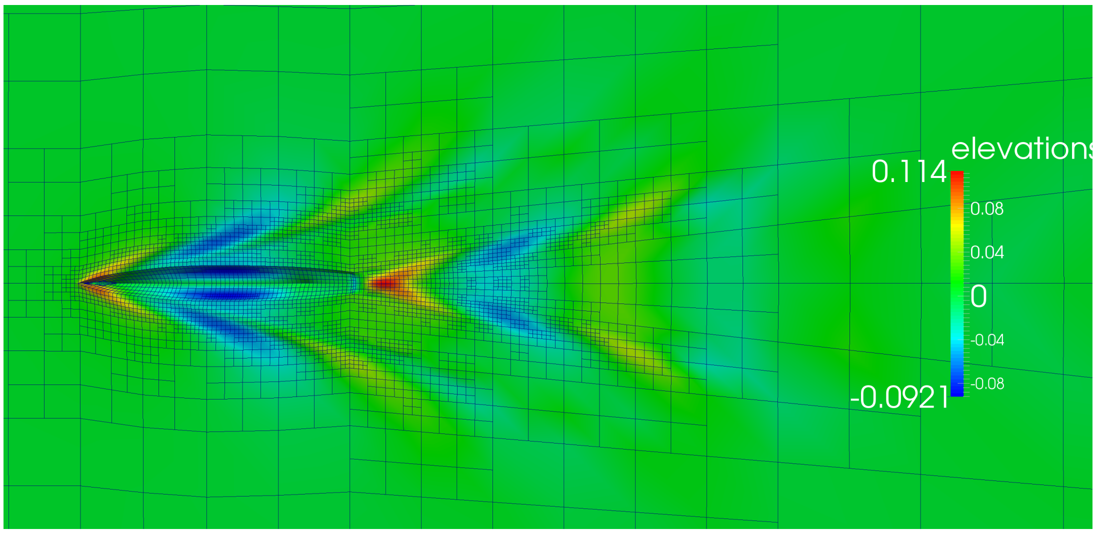


Figure 11: The final, adapted mesh at the end of the simulation of the flow past the DTMB 5415 hull.

Figure 11 shows the final mesh for the simulation of the flow past the DTMB 5415 hull. As can be appreciated, the adaptation algorithm is able to place the vast majority of the free surface cells in correspondence with the featherlets of the Kelvin wake detaching from the hull. Thus, despite being composed by only 4700 nodes, the grid is able to cover the most physically meaningful portions of the flow domain with sufficiently small elements.

Finally, at each time step of the simulation, the current angular and linear displacements are applied to the CAD hull geometry to place it in the proper position. After this, all the water nodes in contact with the ship are projected onto the displaced surface, so that the lateral velocity required to keep such nodes on the water surface can be computed from the distance between the original and projected points.

2.4 Numerical experiments

The first test case selected for the validation of the FSI solver described, is that of a DTMB 5415 hull advancing at constant speed in calm water. For such naval combatant geometry, Olivieri et al. [42] carried out an extensive experimental campaign at Istituto Nazionale per Studi ed Esperienze di Architettura Navale (INSEAN), the biggest Italian ship model basin. In the tests, a DTMB 5415 hull model with displacement $\Delta = 0.549\,\mathrm{t}$ and length between perpendiculars $L_{\mathrm{pp}} = 5.72\,\mathrm{m}$, free to sink and trim, has been towed at speeds ranging from $0.3\,\mathrm{m/s}$ to $3.3\,\mathrm{m/s}$. For each towing velocity V_∞, the hydrodynamic equilibrium values of sink, trim angle and total drag were recorded. Additional wave pattern and wake velocity measurements were carried out at the velocity corresponding to the Froude number $Fr = V_\infty/\sqrt{gL_{\mathrm{pp}}} = 0.28$. Given such quantity of experimental data available, the DTMB 5415 hull is a popular benchmark for the validation of CFD models and software. For this reason, the CAD file describing its geometry is available in the literature. In this work, we assumed that the orientation of the hull in such CAD description is the one corresponding with the hydrostatic equilibrium. Thus, the horizontal position of the hull center of mass has been assumed to coincide with that of the hydrostatic pressure center. The vertical position of the hull center of mass — for which no information has been found in Olivieri et al. [42] — is assumed to coincide with the undisturbed water level.

To reproduce the experimental setup, we carried out a series of FSI simulations in which the DTMB 5415 model hull was impulsively set in motion at speed V_∞. To this end, the only component of the translational motion considered was the vertical one. For this reason, the hydrodynamic force vector $\boldsymbol{F}^w$ appearing in Equation (8) has been computed as

$$\boldsymbol{F}^w = \left(\int_{\Gamma^b} p\boldsymbol{n}\, d\Gamma \cdot \boldsymbol{e}_Z\right)\boldsymbol{e}_Z, \tag{28}$$

so that only the vertical component of the pressure forces will affect the hull motion. Since we only focused on the pitch motion, only the Y component of the pressure forces moment about the ship center of gravity was considered. Thus, the hydrodynamic moment in Equation (9) reads

$$\boldsymbol{M}^w = \left(\int_{\Gamma^b} (\boldsymbol{x} \times p\boldsymbol{n})\,, d\Gamma \cdot \boldsymbol{e}_Y\right)\boldsymbol{e}_Y, \tag{29}$$

For each speed tested, the simulation continued until the hydrodynamic equilibrium position was reached, and the equilibrium values of sink, trim angle and total resistance were recorded. Figure 12 displays the time history of the trim angle θ for the simulation corresponding to $Re = 0.28$. For reference, the green line in the plot represents the equilibrium value of θ, computed as the trim angle average value in the last 20 s of the simulations.

Several FSI simulations were executed, imposing cruise velocities V_∞ which correspond to Froude numbers ranging between 0.15 and 0.4. At the current time, the parallel

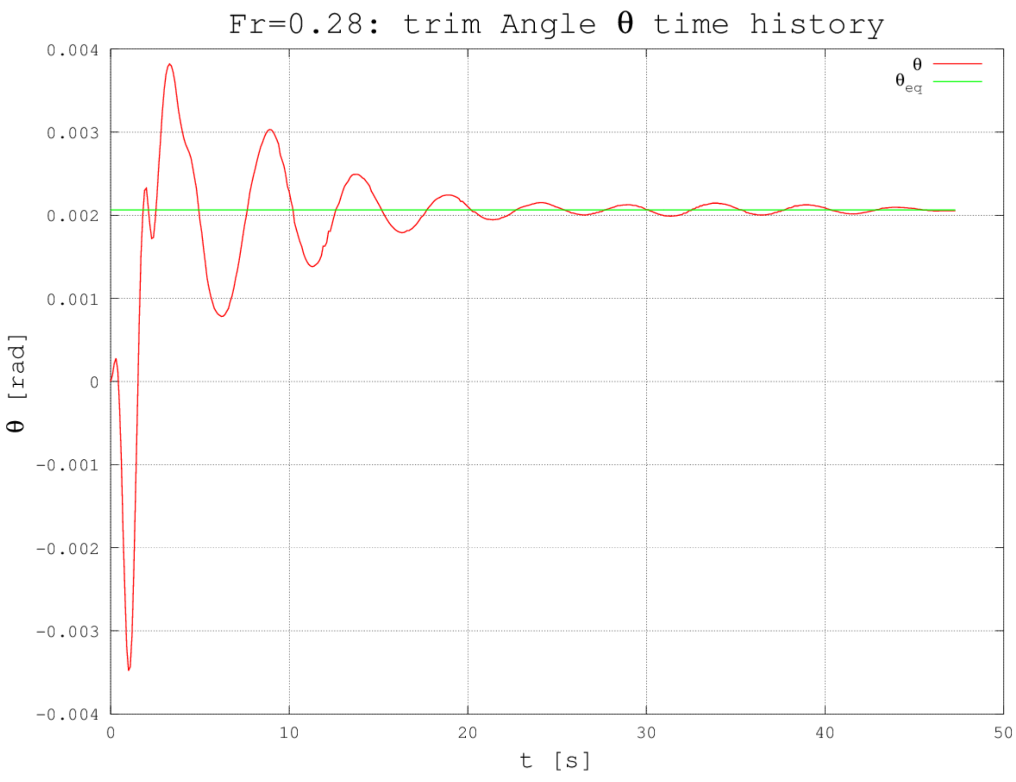


Figure 12: Trim angle θ of the DTMB 5415 hull impulsively started at $Fr = 0.28$, as a function of the simulation time t. The red line represents the time history of the trim angle. The green line displays the final, steady state value, computed as the average angle in the last 20 s of the simulation.

version of the solver is still under development. Thus, the simulations presented in this work were executed on a serial Intel Xeon E5530, 2.40 GHz processor taking about 20 hours to reach convergence.

2.5 Results and Discussion

A quality assessment for the water elevation predictions of the unsteady fully non-linear potential model developed is presented in Figure 13. The plot represents the nondimensional elevation η/L_{pp} of the points of the water surface in contact with the DTMB 5415 hull advancing at $Fr = 0.28$, as a function of the nondimensional longitudinal coordinate $x/x/L_{\text{pp}}$. The colored continuous curves in the plot indicate experimental results obtained by different research groups and presented in Olivieri et al. [42]. The blue asterisks refer instead to the values computed in the present work. By a qualitative standpoint, the numerical results appear in good agreement with the measurements, as the shape of the computed water elevation curve is close to the experimental ones. By a quantitative standpoint, it can be noticed that the longitudinal position of the bow and stern wave crests is reproduced accurately in the simulations. The same can be said for the position of the steep trough following the bow wave, located at $x/L_{\text{pp}} = 0.17$ and for the position of the milder second trough, located in the region around $x/L_{\text{pp}} = 0.65$. As for the wave amplitudes, the predicted wave elevation curve is falling among the experimental ones for most of the hull length. Only in the bow wave region, the water height prediction appears slightly lower then the measured value. This could be related to some numerical dissipation depending on an under refined grid. Further investigations will be carried out to confirm this deduction.

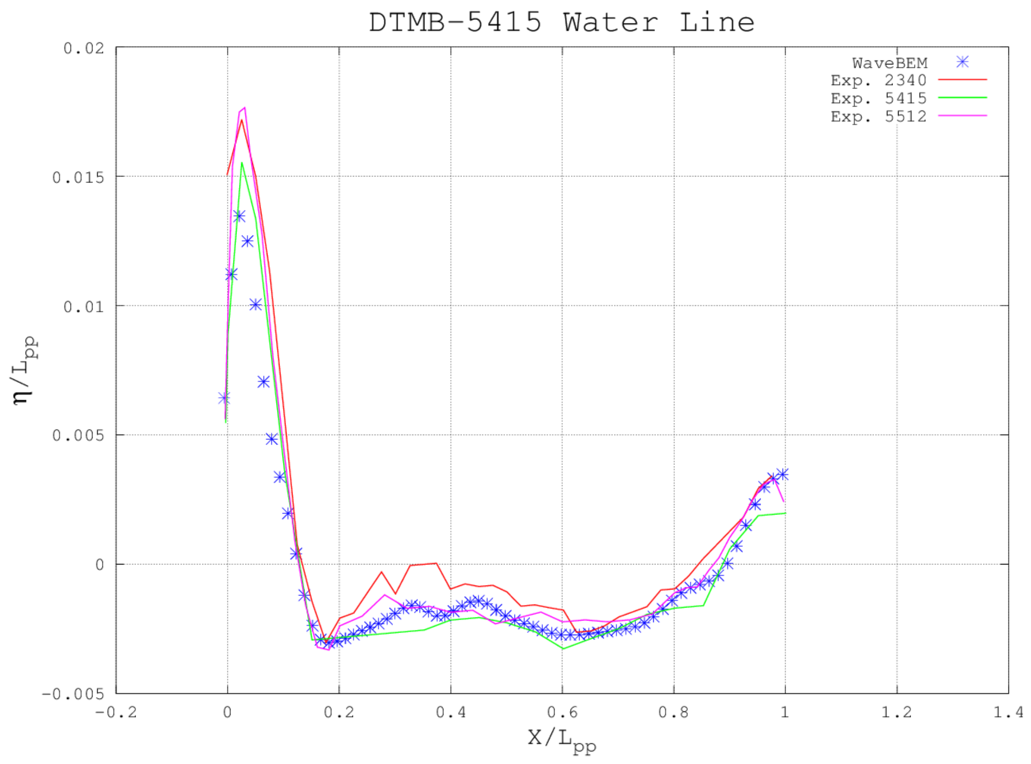


Figure 13: Nondimensionalized wave elevation profile on the surface of the DTMB 5415 hul at $Fr = 0.28$, as a function of the nondimensional longitudinal coordinate. The colored continuous lines represent experimental values obtained by three different groups and reported by Olivieri et al. [42]. The blue asterisks refer to the predictions of the model presented in this work.

Figure 14 shows the comparison between experimental and predicted values for the DTMB 5415 hull hydrodynamic equilibrium sink s_z/L_{pp}, as a function of the Froude number $Fr = V_\infty/\sqrt{gL_{\text{pp}}}$. The blue continuous curve refers to experimental data presented in Olivieri et al. [42], while the red dashed line represents the predictions of the FSI model presented in this work. Again, the qualitative behavior of the numerical results is similar to that of the experimental data. Yet, it can be noticed that the predicted values, especially at low Froude numbers, are consistently lower then the corresponding experimental results. This is likely related to the hydrostatic component of the pressure forces. In fact, even in hydrostatic conditions, the ship surface approximated through the surface mesh is associated to a lower immersed volume with respect to the actual hull. Hence, the discretized ship in the simulation will settle at lower sink values to compensate for the missing buoyancy force. With the grids used for the present simulations, this effect accounts for a 2.5 mm sink value at $Fr = 0$, which is compatible with the offset observed in the left portion of Figure 14. At higher Froude numbers, the hydrodynamic component of the pressure forces becomes higher, and the predicted curve approaches the experimental one. A possible way to reduce this source of error is resorting to iso-geometric discretization techniques, either for the whole BEM solver, or more simply to carry out the pressure integrals on the actual hull surface. This possible improvement to the solver will be investigated in the near future.

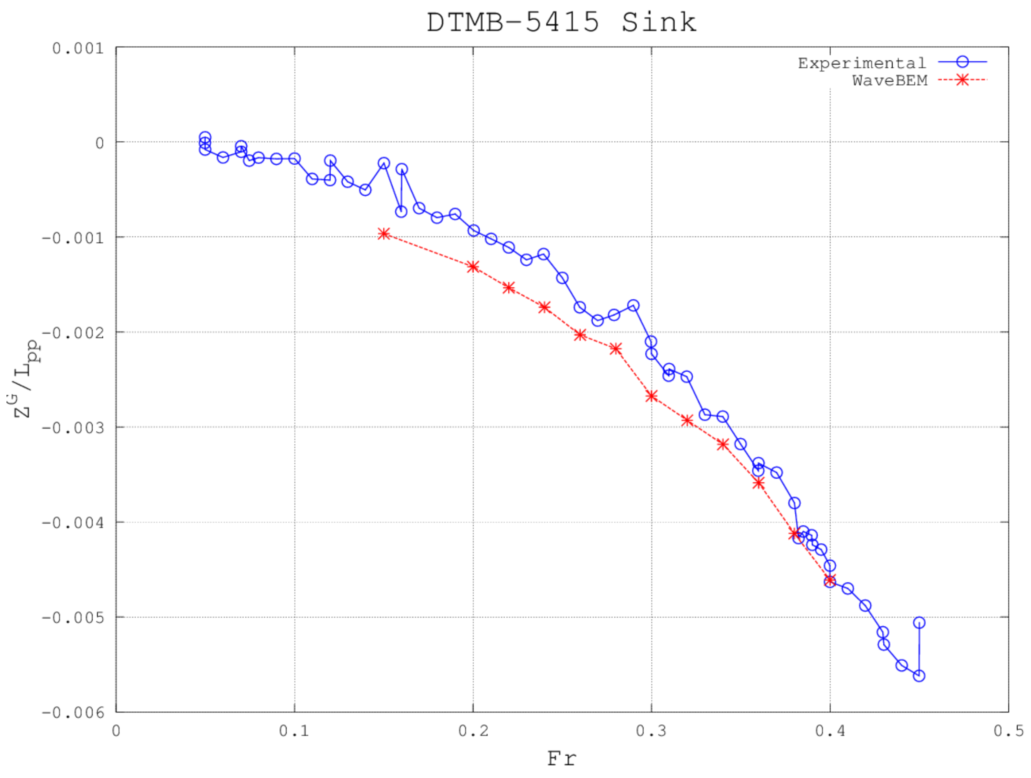


Figure 14: Nondimensional sink as a function of the Froude number for the DTMB 5415 hull. The blue continuous line represents the experimental values presented in Olivieri et al. [42]. The values obtained in this work are represented by the red dashed line.

The plot presented in Fig 15 represents the hydrodynamic equilibrium pitch angle value for the DTMB 5415 hull, as a function of the Froude number. Also in this case, the blue continuous curve refers to the experimental data presented in Olivieri et al. [42], while the red dashed line represents the results obtained in this work. The plot suggests that the model proposed is able to reproduce the qualitative behavior of the DTMB 5415 experimental trim angle curve. For low Froude numbers, the hull presents a positive pitch, while at higher speeds the trim angle progressively becomes negative. The values predicted with the FSI model proposed reproduce with sufficient accuracy both the location ($Fr = 0.28$) and the value ($\theta = 0.0019$ rad) of the maximum of the trim angle curve. Also the predicted location ($Fr = 0.377$) of the curve zero falls rather close to the experimental value of ($Fr = 0.367$). On the right portion of the plot, corresponding to highest speed range tested, the computed trim angles appear to over predict the experimental values. This could have several explanations. On one hand, it might be related to a different vertical location of the hull baricenter, with respect to the experiments. At high speeds, when the pressure at the bulbous bow stagnation point becomes significant, it is in fact possible that the horizontal pressure forces in that region might affect differently the pitch angle, for different vertical distances with the hull baricenter. Even more importantly, the presence of wave breaking in the bow region is not represented in the present model, which might explain for the more trimmed by the stern experimental equilibrium.

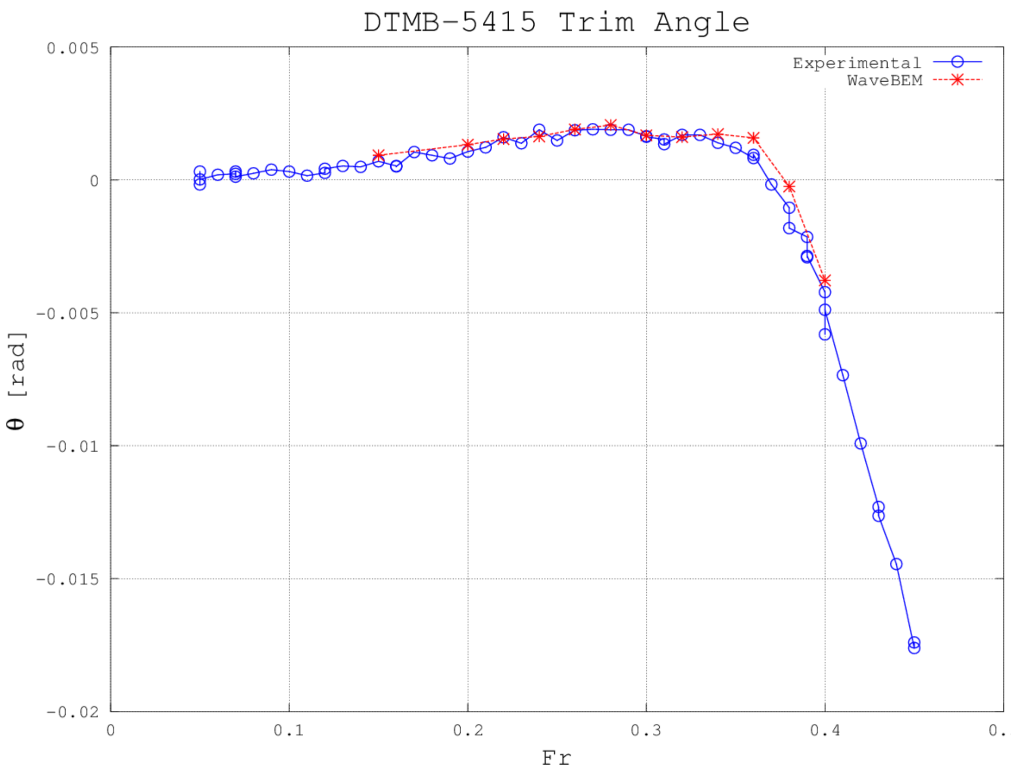


Figure 15: Trim angle (in radiants) as a function of the Froude number for the DTMB 5415 hull. The blue continuous line represents the experimental values presented in Olivieri et al [42]. The values obtained in this work are represented by the red dashed line.

Finally, Figure 16 depicts a comparison between experimental and numerical values of total drag for the DTMB 5415 hull, as a function of the Froude number. Again, the blue continuous curve refers to the experimental data presented in Olivieri et al. [42], while the red dashed line represents the numerical results. In this work, the viscous drag component has been computed by means of the ITTC-57 formula. The plot shows that the behaviour of the total drag curve predicted with the proposed FSI model is in close agreement with that of the experimental one. In all the range of velocities tested, the relative error between experimental and numerical values of total drag is below 9%, while the maximum absolute error is $\sim 7\,\mathrm{N}$. The graph also shows that, as the vessel speed is increased, the FSI model total drag predictions seem to be underestimating the experimental values. This could be again associated to the growing extent of the bow breaking wave, which is not reproduced with the fluid dynamic model adopted.

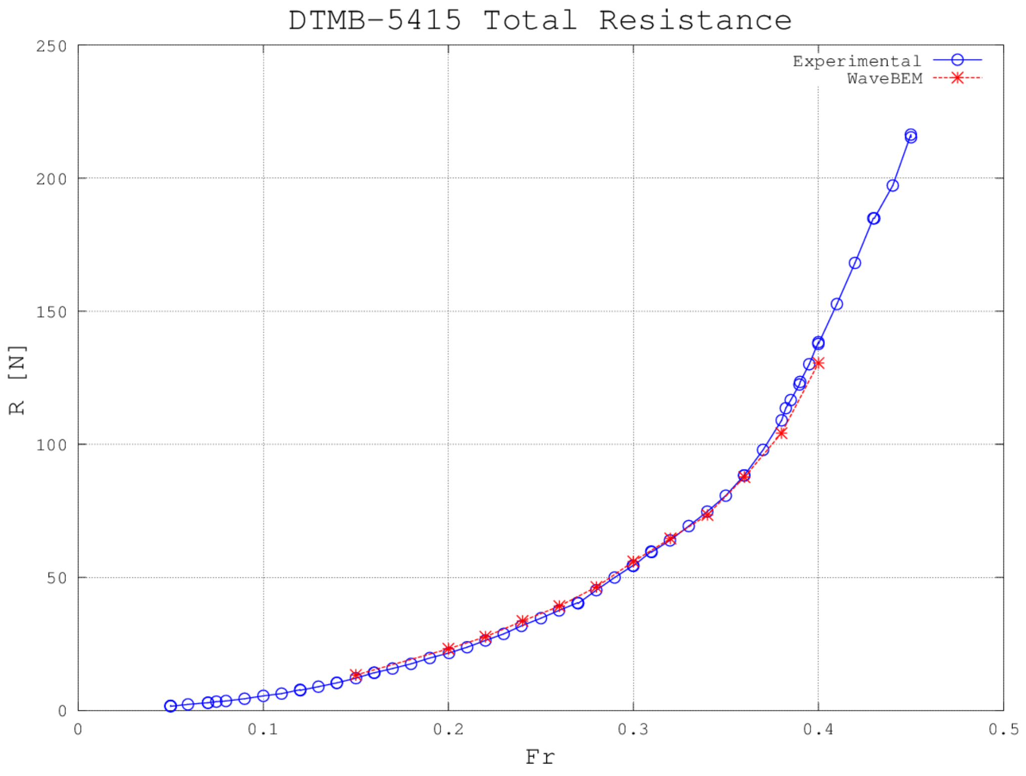


Figure 16: Total resistance of the DTMB 5415 hull as a function of the Froude number. The blue continuous line represents the experimental values presented in Olivieri et al. [42]. The values obtained in this work are represented by the red dashed line.

3 Model order reduction application

3.1 Estimation of the resistance of a hull advancing in calm water

In this section we introduce the problem of the estimation of the resistance of a ship advancing in calm water. The hull shape considered in this work is that of the DTMB 5415, which was conceived for the preliminary design of a US Navy Combatant ship and includes a sonar dome and a transom stern (see Figure 17). Given the abundance of experimental data available in the literature (among others, we cite Olivieri et al. [42]) has become a common benchmark for naval hydrodynamics simulation tools.

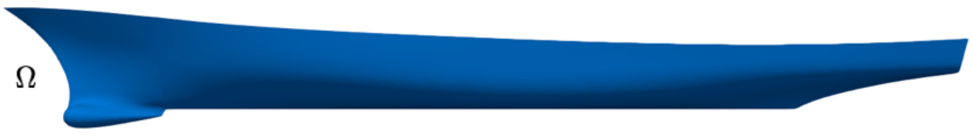


Figure 17: Reference domain Ω, that is the DTMB 5415 hull.

We denote with $\Omega \subset \mathbb{R}^3$ the domain (see Figure 17) associated with our model hull. More specifically, Ω is our *reference* domain, and corresponds with the undeformed DTMB 5415 hull — the latter assumption is not fundamental for the remainder of the paper, but is convenient for practical reasons. We must here remark that the domain considered in the fluid dynamic simulations is in principle the volume of water Ω_w surrounding the hull. Further details about the fluid dynamic domain will be provided in the following sections.

We define the shape morphing $\mathcal{M}(\boldsymbol{x};\boldsymbol{\mu}) : \mathbb{R}^3 \to \mathbb{R}^3$ that maps the reference domain Ω into the deformed domain $\Omega(\boldsymbol{\mu})$, namely $\Omega(\boldsymbol{\mu}) = \mathcal{M}(\Omega;\boldsymbol{\mu})$. It is quite natural to infer that the flow field, and thus the result of the fluid dynamic simulations, will depend on the specific hull shape considered. In turn, such shape is associated to the parameters defining the morphing $\mathcal{M}$ — which will be extensively defined in the next sections. One of the main purposes of this contribution is then to investigate the effect of the morphing parameters on the total resistance, the main fluid dynamic performance parameter. We must here remark that one of the geometrical quantities having the most effect on the resistance is the immersed volume of a hull shape, as higher volumes will generate higher drag values. If designers only consider the absolute resistance value, they might then disregard hull shapes that have relatively good drag performances despite having higher immersed volume. This is of course undesired and could be avoided adopting different strategies. A first possibility is considering a resistance value which is nondimensionalized using a measure of the displaced volume. In alternative, the morphing parameters could be constrained so as to impose a fixed hull immersed volume. Unfortunately, the former solution poses some problem in the identification of the most suitable nondimensionalization strategy resulting in resistance output indices truly independent of the hull immersed volume. As for the latter possibility, it would lead to an undesired complication of the shape parametrization methodology. In this work the effect of immersed volume variations on the hull resistance is taken into account in a more natural fashion. In the fluid dynamic simulations, the rigid motions of the hull are also considered. In this way, once the ship displacement is imposed, the hull shape considered in the simulation reaches the equilibrium position under the action of gravity and hydrodynamic forces. When a shape morphing is characterized by a higher volume, the computed sink will reduce to obtain the same vertical component of the hydrodynamic force. This ensures that each shape is compared on a level ground.

By a practical standpoint, once a point in the parameter domain $\mathbb{D}$ is identified, the specific hull geometry is provided to the fluid dynamic solver, which carries out a flow simulation to come up with a resistance estimate. In this framework free form deformation has been employed for the generation of a very large number of hull geometries based on the DTMB 5415 naval combatant hull shape morphing. Each geometry generated has been used to set up a high-fidelity hydrodynamic simulation with the desired ship displacement and hull speed. Since the serial time dependent high-fidelity simulations are quite time consuming, taking approximatively 24 hours to reach a steady state solution, we adopted a reduction strategy based on dynamic mode decomposition (DMD) to cut the overall computational cost of each simulation to roughly 10 hours. The output resistances for all the configurations tested have been finally analyzed by means of active subspaces (AS) in order to verify if a further reduction in the parameter space is feasible, which could significantly speed up the work of designers.

We refer the interested reader to Mola et al. [37, 40, 43] for further information on the fully nonlinear potential free surface model, on its application to complex hull geometries, and on the treatment of the hull rigid motions respectively. In the next sections, we will provide a brief description of the free form deformation application to the problem at hand, and describe the dynamic mode decomposition strategy used to cut the computational time of the simulations. Finally, the active subspaces used to analyze the resistance dependence on the morphing parameters will be presented.

3.2 Shape parametrization through free form deformation

In this section we will briefly describe how the free form deformation (FFD) shape morphing strategy has been used to produce a parametrised set of modified DTMB 5415 hulls to be used in the fluid dynamic simulations. FFD is a versatile parametrization technique used for shape optimization in a variety of fields such as aerospace engineering, structural mechanics, and biomedical engineering among others. For further insight on the original formulation of FFD we suggest to see Sederberg et al [44], while for a recent work we suggest Salmoiraghi et al [45]. One of the main features of FFD, is that it does not directly manipulate the geometrical object at hand. Instead it deforms a lattice of points built around the object itself, manipulating the whole space in which the geometry is embedded. This lattice has the topology of a hypercube of dimension equal to the dimension of the geometry we want to morph (3D in this work). The lattice is deformed using a trivariate tensor-product of B-spline functions. This produces a continuous and smooth deformation of the geometry. Such methodology has been implemented into a stand alone Python package PyGeM, which has been used to obtain the morphed geometries considered in the present work.

Figure 18 displays a detail of the DTMB 5415 sonar dome, surrounded by the points composing the FFD lattice. FFD is devised to only deform objects which fall within the FFD lattice, so the picture indicates that in this work we are only modifying the shape of the DTMB 5415 sonar dome. The green lattice points in the picture are the ones that have been moved to produce the deformations of domain Ω used in this work.

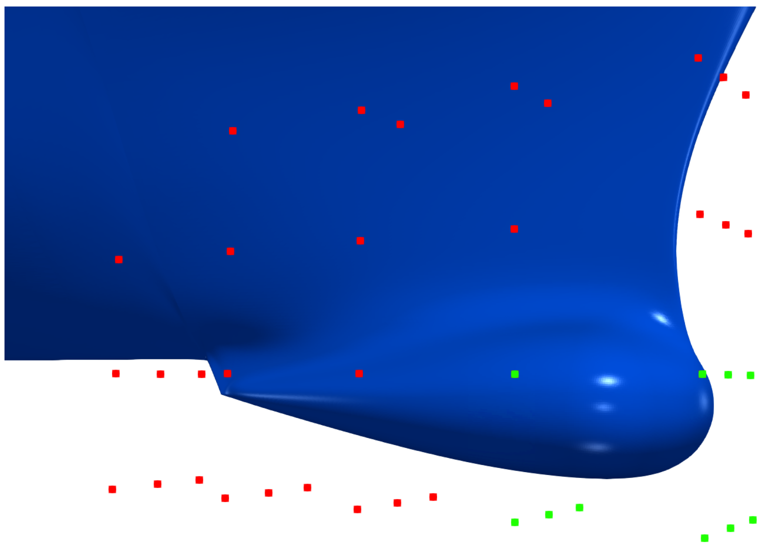


Figure 18: Control points of the FFD lattice around the bulbous bow. The points which are displaced to generate the hull deformations are indicated with the green color.

The FFD procedure can be in fact subdivided into three steps. First, the physical domain Ω is mapped to the reference domain $\widehat{\Omega}$ through a map ψ. Then, some control points (the green ones in the picture, in our case) $\boldsymbol{P}$ of the lattice are moved. Note that the displacement of such points are the parameters $\boldsymbol{\mu}$ that identify a particular deformed geometry. Once the lattice has been deformed, trivariate tensor-products of B-spline functions are used to compute the map $\widehat{T}$ that associates the original lattice to the deformed one. Finally, the back mapping from the deformed reference domain is applied to the deformed physical domain $\Omega(\boldsymbol{\mu})$ using the map ψ^{-1}. So it possible to express the FFD map $\mathcal{M}(\boldsymbol{x};\boldsymbol{\mu}) : \Omega \subset \mathbb{R}^3 \rightarrow \Omega(\boldsymbol{\mu}) \subset \mathbb{R}^3$ by the composition of these three maps, i.e.

$$\mathcal{M}(\cdot\,;\boldsymbol{\mu}) = (\psi^{-1} \circ \widehat{T} \circ \psi)(\cdot\,;\boldsymbol{\mu}).$$

In our case we have chosen $\boldsymbol{\mu} \in \mathbb{D} := [-0.3, 0.3]^8$, that results in a wide range of different bulbous bow configurations (see Figure 19 for an idea of different possible deformations). For sake of clarity we underline that the undeformed original domain is obtained setting all the geometrical parameters to 0. The main purpose of this work is to carry out a rather wide exploration of the shape parameter space, to assess the dependence of the output parameters on the shape morphing parameters. That is why the parameters bounds are chosen so as to be able to obtain somewhat significant — and yet physically meaningful — deformations of the bow bulb.

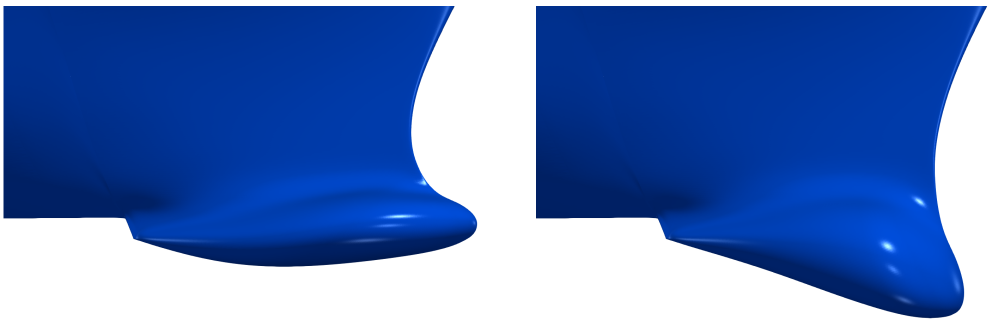


Figure 19: Two different examples of deformations.

3.3 System evolution reconstruction with dynamic mode decomposition

As mentioned, the unsteady and nonlinear fluid dynamic model adopted results in rather expensive simulations. Of course this could in principle reduced through full parallelization of the software developed which, however, has yet to be carried out. In this work, the computational cost of each simulation carried out has been reduced through the first application application of the dynamic mode decomposition (DMD) to our fully nonlinear potential solver output.

Dynamic mode decomposition (DMD) is a data-driven algorithm that provides a finite approximation of the infinite dimensional Koopman operator (see Koopman [46]). Proposed in Schmid [47] for fluid dynamics analysis, this technique has become popular in the last years mainly because (i) it allows to approximate nonlinear dynamics through low-rank structures that evolve in time and (ii) it relies only on the data, avoiding assumptions on the underlying system. We have implemented this algorithm, as well as many of its variants, in an open source Python package on GitHub, called PyDMD (see Demo et al. [48]). In this section we will introduce the DMD algorithm and we will discuss the fluid dynamic problem at hand as an example of its application. For an application of DMD to snapshots obtained from the solution of RANS equations see Demo et al. [49].

Let the variable x_k represent the state of the evolving system at time $t_k = k\Delta t$. Basically, we want to find a linear finite dimensional Koopman operator $\mathbf{A}$ such that $x_{k+1} = \mathbf{A}x_k$. In order to build this operator, we collect a series of data vectors $\{x_i\}_{i=1}^{l}$, which we will refer to as snapshots from now on, and which represent the time-equispaced system states. We assume all the snapshots have the same dimension, that is $x_k \in \mathbb{R}^n$ for all $k = 1, \ldots, l$, and we assume the dimension n of a snapshot is larger that the number of snapshots l, i.e. $n > l$. We arrange the snapshots in two matrices, $\mathbf{S}$ and $\dot{\mathbf{S}}$, as

$$\mathbf{S} = \begin{bmatrix} x_1^1 & x_2^1 & \cdots & x_{l-1}^1 \\ x_1^2 & x_2^2 & \cdots & x_{l-1}^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^n & x_2^n & \cdots & x_{l-1}^n \end{bmatrix}, \qquad \dot{\mathbf{S}} = \begin{bmatrix} x_2^1 & x_3^1 & \cdots & x_l^1 \\ x_2^2 & x_3^2 & \cdots & x_l^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_2^n & x_3^n & \cdots & x_l^n \end{bmatrix} \tag{30}$$

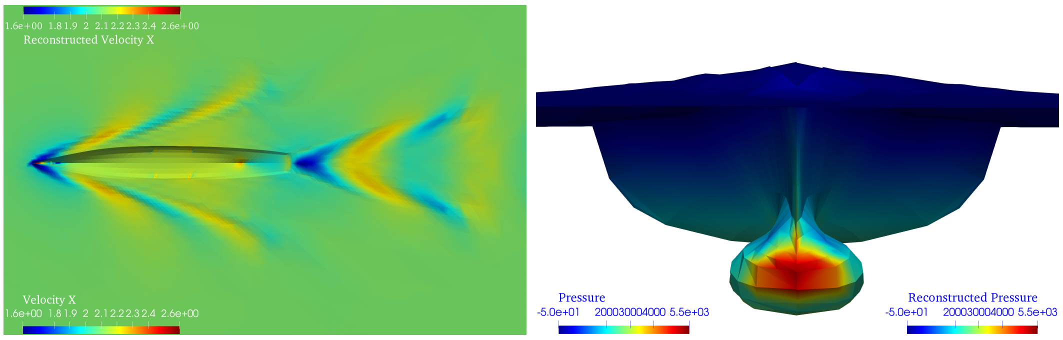

1.6e+00 1.8 1.9 2 2.1 2.2 2.3 2.4 2.6e+00
Reconstructed Velocity X
Velocity X
1.6e+00 1.8 1.9 2 2.1 2.2 2.3 2.4 2.6e+00
Pressure
-5.0e+01 2000 3000 4000 5.5e+03
Reconstructed Pressure
-5.0e+01 2000 3000 4000 5.5e+03

Figure 20: A comparison between the original output of the fluid dynamic simulations and the DMD reconstructed one. The top image represents a contour plot of water X velocity around a morphing of the DTMB 5415 hull advancing at $Fr = 0.28$ in calm water. The top half of the plot depicts the reconstructed fluid velocity, while the bottom half represents the one resulting from the high-fidelity computation. The bottom image s a front view of a contour plot of the pressure field on the hull and water surface. In this case, the left part of the plot refers to the high-fidelity solution, while the right half refers to the reconstructed pressure.

in order to build the linear operator by minimizing $\|\dot{\mathbf{S}} - \mathbf{AS}\|_2$. We underline that each column of $\dot{\mathbf{S}}$ contains the state vector at the next timestep of the one in the corresponding $\mathbf{S}$ column. Hence, the best-fit matrix $\mathbf{A}$ is given by $\mathbf{A} = \dot{\mathbf{S}}\mathbf{S}^\dagger$, where the symbol † denotes the Moore-Penrose pseudo-inverse. Since the snapshots usually have high dimension for complex systems, the matrix $\mathbf{A}$ becomes very large and it is difficult to manipulate. The DMD algorithm projects the data onto a low-rank subspace defined by the Proper Orthogonal Decomposition (POD) modes, then computes the low-dimensional operator $\tilde{\mathbf{A}}$. This operator is used to reconstruct the leading nonzero eigenvalues and eigenvectors of the full-dimensional operator $\mathbf{A}$ without ever explicitly computing $\mathbf{A}$. Using the truncated singular value decomposition of matrix $\mathbf{S} \approx \mathbf{U}_r\mathbf{\Sigma}_r\mathbf{V}_r^*$ with rank r, we can build the low-rank linear operator as:

$$\tilde{\mathbf{A}} = \mathbf{U}_r^*\mathbf{A}\mathbf{U}_r = \mathbf{U}_r^*\dot{\mathbf{S}}\mathbf{S}^\dagger\mathbf{U}_r = \mathbf{U}_r^*\dot{\mathbf{S}}\mathbf{V}_r\mathbf{\Sigma}_r^{-1}\mathbf{U}_r^*\mathbf{U}_r = \mathbf{U}_r^*\dot{\mathbf{S}}\mathbf{V}_r\mathbf{\Sigma}_r^{-1}. \tag{31}$$

We can now reconstruct the eigenvectors and eigenvalues of the matrix $\mathbf{A}$ using the eigendecomposition $\tilde{\mathbf{A}}\mathbf{W} = \mathbf{W}\mathbf{\Lambda}$. In detail (see Tu et al. (2014)), the DMD modes $\mathbf{\Theta}$ can be computed by projecting the low-rank eigenvectors on the high-dimensional space $\mathbf{\Phi} = \mathbf{U}_r\mathbf{W}$ (*projected modes*) or computing the eigenvectors of $\mathbf{A}$ as $\mathbf{\Theta} = \dot{\mathbf{S}}\mathbf{V}_r\mathbf{\Sigma}_r^{-1}\mathbf{W}$ (*exact modes*). Moreover, the eigenvalues of $\tilde{\mathbf{A}}$ correspond to the nonzero eigenvalues of $\mathbf{A}$, and they contain the growth/decay rate and the frequencies of the corresponding modes. Recalling the equations above, we underline that $\mathbf{A} = \mathbf{\Theta}\mathbf{\Lambda}\mathbf{\Theta}^\dagger$. The generic snapshot x_{k+1} can be reconstructed by premultiplying the first snapshot k times by the linear operator, such that $x_{k+1} = \mathbf{A}^k x_1 = (\mathbf{\Theta}\mathbf{\Lambda}\mathbf{\Theta}^\dagger \ldots \mathbf{\Theta}\mathbf{\Lambda}\mathbf{\Theta}^\dagger)x_1$. Hence the state of the system can be approximated, for any time t_{k+1} as $x_{k+1} = \mathbf{\Theta}\mathbf{\Lambda}^k\mathbf{\Theta}^\dagger x_1$, where the vector $\mathbf{\Theta}^\dagger x_1$ is usually called *amplitudes*.

The application of DMD to the computational fluid dynamics simulations in this work is carried out collecting the snapshots within the temporal window $t = [7\,\mathrm{s}, 15\,\mathrm{s}]$, with $\Delta t = 0.1\,\mathrm{s}$. As both the motion of the hull and of the free surface nodes are computed in

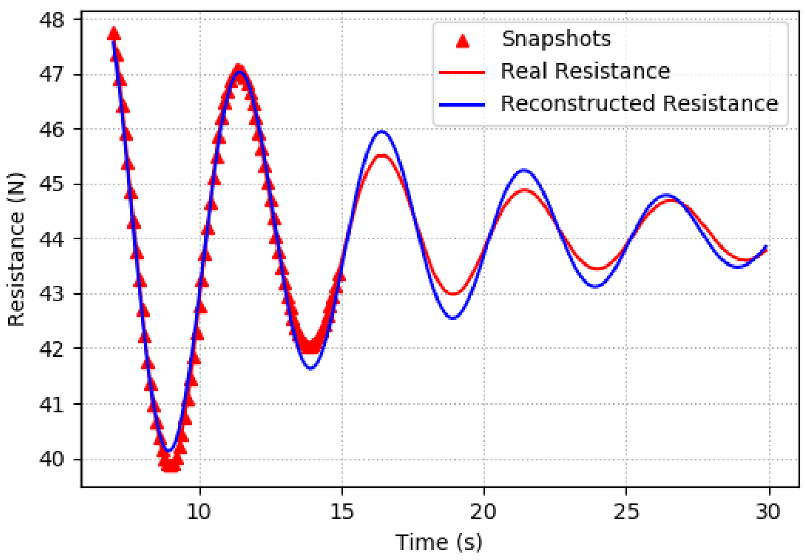


Figure 21: Comparison of high-fidelity (red continuous curve) and DMD reconstructed (blue continuous curve) total resistance time history. The triangular markers in the plot denote the times at which the snapshots for the DMD have been stored.

the simulations, the DMD algorithm is educated using snapshot vectors which include the grid nodes coordinates, along with the flow velocity and pressure. The DMD algorithm has been used to compute a rather accurate approximation of the whole fluid dynamic problem solution up to $t = 30\,\mathrm{s}$, at the mere cost of a post processing phase requiring only few seconds. This led to a significant reduction of the computational cost, dropping the time required for each 30 s simulation from 24 h to 10 h.

Figure 20 compares the high-fidelity fluid dynamic solution at $t = 30\,\mathrm{s}$ with the one which has been reconstructed through the described DMD strategy. The figure includes both pressure and velocity field plots, which have been split in half to give a parallel view of both the high-fidelity and reconstructed field. Despite the DMD has been extrapolated from snapshots that are alder than $t = 15\,\mathrm{s}$, no difference between original and reconstructed flow and water elevation fields is appreciable in the contour plots. Figure 21 represents a comparison of time history of the hull resistance computed from the high-fidelity and DMD reconstructed hull pressure field. Again, the DMD approximated time history seems able to reconstruct with good approximation the high-fidelity values. This is especially true for the last instants of the simulation, which are used to compute the final value of the total hull resistance, one of the output parameters considered in this work.

3.4 Parameter space verification and reduction by active subspaces

The active subspaces (AS) property has been brought to attention recently through the work of P. Constantine [50]. The AS property is a characteristic of the scalar function relating the scalar output to the parameters $\boldsymbol{\mu}$, and of a probability density function associated to such function. By a qualitative standpoint, AS is typically exploited to assess whether the parameter space allows for a significant — and of course useful — dimension reduction. By a quantitative standpoint, it can also be used to assess the sensitivity of the output with respect to each parameter considered. The main idea of AS is to operate in the parameters space, rescaling the inputs $\boldsymbol{\mu}$ and then rotating them with respect to the origin. In some cases (see Tezzele et al. [51, 52]), such procedure reveals in lower dimension behavior of the output function $f(\boldsymbol{\mu})$ (the total resistance or the hull sink and trim, in our case). We underline that AS does not identify a subset of the inputs as important, instead it identifies a set of important directions in the space of all inputs. These directions (which are linear combinations of the input variables) are the ones along which the output function varies the most on average. When an active subspace is identified for the problem of interest, it is possible to perform different parameter studies.

Now we review how it is possible to find active subspaces. Let us assume[1] $f : \mathbb{R}^m \to \mathbb{R}$ is a scalar function and $\rho : \mathbb{R}^m \to \mathbb{R}^+$ a probability density function, where m is the dimension of the parameters. Since all the geometrical configurations can be drawn with equal probability, a uniform probability density will suffice in our case. In particular, we assume f continuous and differentiable in the support of ρ, with continuous and square-integrable (with respect to the measure induced by ρ) derivatives. The active subspaces of the pair (f, ρ) are the eigenspaces of the covariance matrix associated to the gradients $\nabla_{\boldsymbol{\mu}} f$. This matrix, denoted by $\boldsymbol{\Sigma}$, is the so-called uncentered covariance matrix of the gradients of f (among others see Devore (2015) for a more deep understanding of these operators). Its elements are the average products of partial derivatives of f, that is:

$$\boldsymbol{\Sigma} = \mathbb{E}\,[\nabla_{\boldsymbol{\mu}} f \, \nabla_{\boldsymbol{\mu}} f^T] = \int_{\mathbb{D}} (\nabla_{\boldsymbol{\mu}} f)(\nabla_{\boldsymbol{\mu}} f)^T \rho \, d\boldsymbol{\mu}, \tag{32}$$

where $\mathbb{E}[\cdot]$ is the expected value. We use a Monte Carlo method to approximate the eigenpairs of $\boldsymbol{\Sigma}$ as in Constantine et al. (2015):

$$\boldsymbol{\Sigma} \approx \frac{1}{N_{\text{train}}^{\text{AS}}} \sum_{i=1}^{N_{\text{train}}^{\text{AS}}} \nabla_{\boldsymbol{\mu}} f_i \, \nabla_{\boldsymbol{\mu}} f_i^T, \tag{33}$$

where we draw $N_{\text{train}}^{\text{AS}}$ independent samples $\boldsymbol{\mu}^{(i)}$ from the measure ρ, and where we have $\nabla_{\boldsymbol{\mu}} f_i = \nabla_{\boldsymbol{\mu}} f(\boldsymbol{\mu}^{(i)})$. The matrix $\boldsymbol{\Sigma}$ admits a real eigenvalue decomposition because it is symmetric positive semidefinite, so we have $\boldsymbol{\Sigma} = \mathbf{W}\boldsymbol{\Lambda}\mathbf{W}^T$, where $\mathbf{W}$ contains the eigenvectors and is in $O(m)$, the orthogonal group, while $\boldsymbol{\Lambda}$ is the diagonal matrix of non-negative eigenvalues arranged in descending order.

The lower dimensional parameter subspace is formed by selecting the first $M < m$ eigenvectors. We underline that perturbations in the first set of coordinates change f, on average, more than perturbations in the second set of coordinates. We can discard

[1]In this section we will omit the dependence on $\boldsymbol{\mu}$. It should be understood that $f = f(\boldsymbol{\mu})$, $\rho = \rho(\boldsymbol{\mu})$, etc.

the vectors corresponding to the low eigenvalues since they are in the nullspace of the covariance matrix. Doing so, we are able to construct an approximation of f. To be more clear, let us partition $\mathbf{\Lambda}$ and $\mathbf{W}$ as follows:

$$\mathbf{\Lambda} = \begin{bmatrix} \mathbf{\Lambda}_1 & \\ & \mathbf{\Lambda}_2 \end{bmatrix}, \qquad \mathbf{W} = [\mathbf{W}_1 \quad \mathbf{W}_2],$$

where $\mathbf{\Lambda}_1 = \text{diag}(\lambda_1, \dots, \lambda_M)$, and $\mathbf{W}_1$ contains the first M eigenvectors. The active subspace is the the range of $\mathbf{W}_1$. We call inactive subspace the range of the remaining eigenvectors in $\mathbf{W}_2$. The linear combinations of the input parameters with weights from the important eigenvectors are the active variables. By projecting the full parameter space onto the active subspace we can approximate the behaviour of f. In particular we have the following formulas for the active variable $\boldsymbol{\mu}_M$ and the inactive variable $\boldsymbol{\eta}$:

$$\boldsymbol{\mu}_M = \mathbf{W}_1^T \boldsymbol{\mu} \in \mathbb{R}^M, \qquad \boldsymbol{\eta} = \mathbf{W}_2^T \boldsymbol{\mu} \in \mathbb{R}^{m-M}. \tag{34}$$

Using Equation (34) and the fact that $\mathbf{W} \in O(m)$ we can express any point in the parameter space $\boldsymbol{\mu} \in \mathbb{R}^m$ in terms of $\boldsymbol{\mu}_M$ and $\boldsymbol{\eta}$ as follows:

$$\boldsymbol{\mu} = \mathbf{W}\mathbf{W}^T \boldsymbol{\mu} = \mathbf{W}_1\mathbf{W}_1^T \boldsymbol{\mu} + \mathbf{W}_2\mathbf{W}_2^T \boldsymbol{\mu} = \mathbf{W}_1 \boldsymbol{\mu}_M + \mathbf{W}_2 \boldsymbol{\eta}.$$

So it is possible to rewrite f as $f(\boldsymbol{\mu}) = f(\mathbf{W}_1 \boldsymbol{\mu}_M + \mathbf{W}_2 \boldsymbol{\eta})$, and, using only the active variables, we can construct a surrogate quantity of interest g, that is $f(\boldsymbol{\mu}) \approx g(\mathbf{W}_1^T \boldsymbol{\mu}) = g(\boldsymbol{\mu}_M)$. In our pipeline, the surrogate quantity of interest g will be obtained by a response surface method.

3.5 Numerical Results

In this section we present the numerical results obtained by applying all the methodologies presented above to the DTMB 5415 model hull. The FFD morphing methodology illustrated has been used to generate 130 different deformations of the original hull. The parametrised shapes correspond to uniform sampling points in the parameter space box $\mathbb{D} = [-0.3, 0.3]^8$. Each IGES geometry produced has been then used as the input of a high fidelity simulation in which the hull has been set to advance in calm water at a constant speed corresponding to $Fr = 0.28$. Each high fidelity computation has been carried out to simulate 15 s of the flow past the hull after it has been impulsively started from rest. Between the 7th and 15th second of the high-fidelity simulations, the solver saved the full flow field at sampling intervals $\Delta t = 0.1$ s. Such flow field snapshots have been used to feed the DMD algorithm implemented, and complete the fluid dynamic simulations until convergence to the regime solution was reached at $t = 30$ s. The reconstructed flow fields have been finally used to evaluate the hull total resistance and the hydrodynamic trim position, which are the output performance parameters considered in this work. The dataset composed by the output of the simulations has been divided in a train dataset (75% of the outputs) used to train the AS algorithm and a test dataset (25% of the outputs) used to validate the methodology.

Figure 22 depicts the eigenvalues estimates of the matrix $\boldsymbol{\Sigma}$ (black dots and line) and also displays the bootstrap intervals (corresponding to the grey area surrounding the eigenvalues lines). In the figure, the left plot is referred to the total resistance, while the right one is displays the hydrodynamic trim angle.

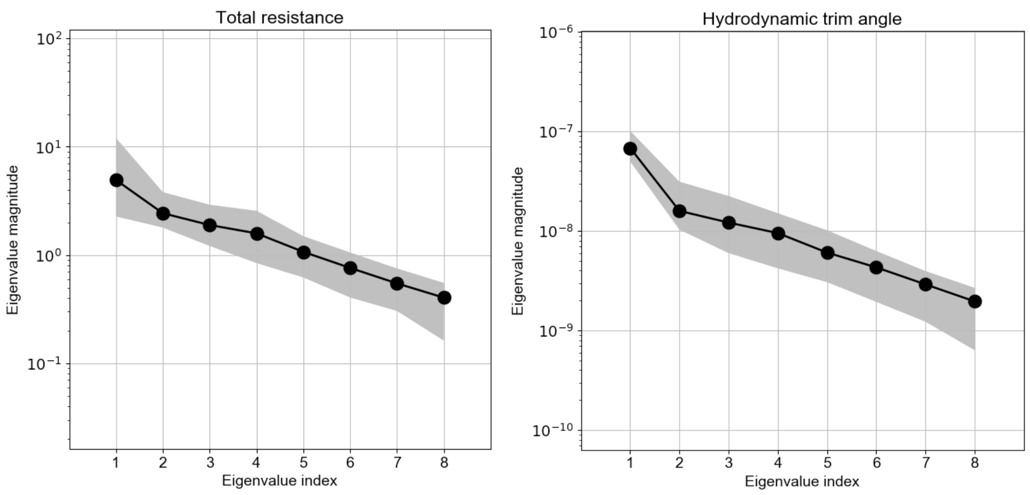


Figure 22: Eigenvalues estimates of the matrix $\boldsymbol{\Sigma}$ for the total resistance (on the left) and the hydrodynamic trim angle (on the right). The black dots in the plot indicate the eigenvalues, which the grey area is defined by the bootstrap intervals.

The plots indicate that a factor of at least 10 exists between the highest and lowest $\boldsymbol{\Sigma}$ eigenvalues. Such difference is clearly more pronounced when the hydrodynamic trim is the output considered. Yet, the plots also show that the eigenvalues magnitude is rather evenly distributed across the range they span. More precisely, the absence of a major gap between the higher module eigenvalues and the lower module ones, is suggesting that the active subspace is most revealing a clear cut low dimensional behaviour of the target functions with respect to the active variables, as is the case for different applications (Tezzele et al. [52]). Yet, especially in the case of the hydrodynamic trim angle output, the first eigenvalue module is considerably higher than that of the remaining ones. That is why, for the hydrodynamic trim it was possible to compute a bivariate surface response using the first two active variables, obtained as linear combinations of the original parameters with coefficients obtained by the eigenvectors corresponding to the two highest module eigenvalues. Figure 23 shows the quartic surface that best approximates the training dataset in the sense of least squares, along with the points in the test dataset (which are indicated by the dots). Each point represents the value of the target function $f(\boldsymbol{\mu})$ against the active variables $\boldsymbol{\mu}_M = \boldsymbol{W}_1^T\boldsymbol{\mu} \in \mathbb{R}^2$. As can be appreciated, the points corresponding to the true output are not distributed randomly in the space, but tend to be somewhat clustered around the surface. This is particularly true when the output parameter considered is the hydrodynamic trim angle (right plot) for which, as we have seen, the $\boldsymbol{\Sigma}$ eigenvalues corresponding to the first active variable was significantly higher then the remaining ones. Thus, whenever the gap between the leading $\boldsymbol{\Sigma}$ eigenvalues allows for it, the AS algorithm is able to successfully identify a set active variables upon which the output is — with reasonable approximation — exclusively depending.

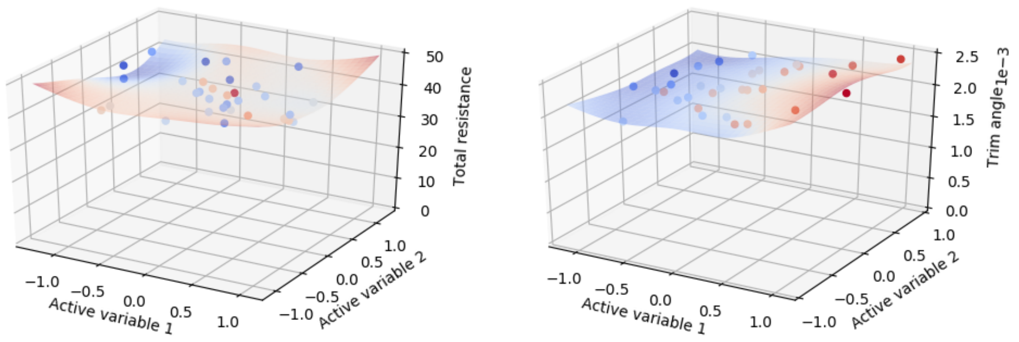


Figure 23: Comparison between the quartic surface response obtained with two active variables and the true output for the test data set, indicated by the dots. The left plot refers to the total resistance output, while the right one to the hydrodynamic trim angle.

To provide a more quantitative assessment of how much such approximation is in fact reasonable, and depends on the gap existing between the first $\mathbf{\Sigma}$ eigenvalues and the following ones, we computed the average error as the average, among 10 different eigenvalues estimates, of the root mean square error divided by the maximum range of variation of the target function. As expected, the accuracy of the two dimensional surface response predictions for the hull total resistance is rather low, and only a 20% average error is obtained, that is about 1.8 N. The average error obtained for the hydrodynamic trim angle output is approximately 11%, that corresponds to an absolute rmse of $8 \cdot 10^{-5}$, confirming that such output can be better represented through AS. Thus, for all those applications for which errors of the reported magnitudes are acceptable — as might be the case for early design stages — AS provides a recipe to reduce the parameter space from 8 to 2 variables. In addition, the error analysis provides a further confirmation of the fact that once the eigenvalue analysis is carried out, the detection of a possible cliff in the eigenvalue curve (see Figure 22) is a measure of how well AS will perform. So, since the computational cost of the post processing operations required for the AS algorithm proposed is marginal with respect to that of the high-fidelity simulations, it should be always worth checking if AS could be used to obtain a significant drop in the parameter space dimension.

References

[1] Briggs W. L., Henson V. E., and McCormick S. F. *Multigrid Tutorial*. Society of Industrial and Applied Mathematics, second edition, 2000.

[2] Clevenger T. C., Heister T., Kanschat G., and Kronbichler M. A flexible, parallel, adaptive geometric multigrid method for FEM. *ACM Transactions on Mathematical Software*, 47(1):7:1–27, 2021.

[3] Carey G. F. *Computational Grids: Generation, Adaptation and Solution Strategies*. Taylor & Francis, 1997.

[4] Bangerth W. and Rannacher R. *Adaptive Finite Element Methods for Differential Equations*. Birkhäuser Verlag, 2003.

[5] Ciarlet P. G. and Raviart P.-A. The combined effect of curved boundaries and numerical integration in isoparametric finite element methods. In Aziz A. K., editor, *The Mathematical Foundations of the Finite Element Method with Applications to Partial Differential Equations*, pages 409–474. Academic Press, 1972.

[6] Mansfield L. Approximation of the boundary in the finite element solution of fourth order problems. *SIAM Journal on Numerical Analysis*, 15(3):568–579, 1978.

[7] Bassi F. and Rebay S. High-order accurate discontinuous finite element solution of the 2D Euler equations. *Journal of Computational Physics*, 138(2):251–285, 1997.

[8] Braess D. *Finite Elements*. Cambridge University Press, 2007.

[9] Bartels S., Carstensen C., and Dolzmann G. Inhomogeneous Dirichlet conditions in a priori and a posteriori finite element error analysis. *Numerische Mathematik*, 99(1):1–24, 2004.

[10] Dörfler W. and Rumpf M. An adaptive strategy for elliptic problems including a posteriori controlled boundary approximation. *Mathematics of Computation*, 67(224):1361–1382, 1998.

[11] Bhattacharyya P. K. and Nataraj N. On the combined effect of boundary approximation and numerical integration on mixed finite element solution of 4th order elliptic problems with variable coefficients. *ESAIM: Mathematical Modelling and Numerical Analysis*, 33(4):807–836, 1999.

[12] Hartmann R. *Adaptive Finite Element Methods for the Compressible Euler Equations*. PhD thesis, University of Heidelberg, 2002.

[13] Mengaldo G., Moxey D., Turner M., Moura R. C., Jassim A., Taylor M., Peiró J., and Sherwin S. J. Industry-relevant implicit large-eddy simulation of a high-performance road car via spectral/hp element methods, 2021. last revised version.

[14] Hindenlang F., Bolemann T., and Munz C.-D. Mesh curving techniques for high order discontinuous Galerkin simulations. In *Notes on Numerical Fluid Mechanics and Multidisciplinary Design*, pages 133–152. Springer International Publishing, 2015.

[15] Krais N., Beck A., Bolemann T., Frank H., Flad D., Gassner G., Hindenlang F., Hoffmann M., Kuhn T., Sonntag M., and Munz C. FLEXI: A high order discontinuous galerkin framework for hyperbolic-parabolic conservation laws. *Computers & Mathematics with Applications*, 81:186–219, 2021. Development and Application of Open-source Software for Problems with Numerical PDEs.

[16] Moxey D., Cantwell C. D., Bao Y., Cassinelli A., Castiglioni G., Chun S., Juda E., Kazemi E., Lackhove K., Marcon J., Mengaldo G., Serson D., Turner M., Xu H., Peiró J., Kirby R. M., and Sherwin S. J. Nektar++: Enhancing the capability and application of high-fidelity spectral/hp element methods. *Computer Physics Communications*, 249:107110, 2020.

[17] Arndt D., Bangerth W., Davydov D., Heister T., Heltai L., Kronbichler M., Maier M., Pelteret J.-P., Turcksin B., and Wells D. The DEAL.II finite element library: Design, features, and insights. *Computers & Mathematics with Applications*, 81:407–422, 2021.

[18] Arndt D., Bangerth W., Clevenger T. C., Davydov D., Fehling M., Garcia-Sanchez D., Harper G., Heister T., Heltai L., Kronbichler M., Kynch R. M., Maier M., Pelteret J.-P., Turcksin B., and Wells D. The DEAL.II library, Version 9.1. *Journal of Numerical Mathematics*, 27(4):203–213, 2019.

[19] Kronbichler M., Heister T., and Bangerth W. High accuracy mantle convection simulation through modern numerical methods. *Geophysical Journal International*, 191(1):12–29, 2012.

[20] Heister T., Dannberg J., Gassmöller R., and Bangerth W. High accuracy mantle convection simulation through modern numerical methods – ii: realistic models and problems. *Geophysical Journal International*, 210(2):833–851, 2017.

[21] Open Cascade S.A.S. Opencascade Technology, 2010. http://www.opencascade.org.

[22] Farin G. *Curves and surfaces for CAGD: A practical guide*. Morgan Kaufmann, San Francisco, CA, USA, 5th edition, 2002.

[23] Gordon W. J. and Thiel L. C. Transfinite mappings and their application to grid generation. *Applied Mathematics and Computation*, 10:171–233, 1982.

[24] Ciarlet P. G. and Raviart P.-A. Interpolation theory over curved elements, with applications to finite element methods. *Computer Methods in Applied Mechanics and Engineering*, 1(2):217–249, 1972.

[25] Strang G. and Fix G. F. *An analysis of the finite element method*. Wellesley-Cambridge Press, Wellesley, MA, USA, 1988.

[26] Šolín P., Segeth K., and Doležel I. *High-order finite element methods.* Chaptman & Hall/CRC, Boca Raton, FL, USA, 2004.

[27] Moxey D., Ekelschot D., Keskin Ü., Sherwin S. J., and Peiró J. High-order curvilinear meshing using a thermo-elastic analogy. *Computer-Aided Design*, 72:130–139, 2016.

[28] Mittal K. and Fischer P. F. Mesh smoothing for the spectral element method. *Journal of Scientific Computing*, 78(2):1152–1173, 2019.

[29] Dobrev V., Knupp P., Kolev T., Mittal K., and Tomov V. The target-matrix optimization paradigm for high-order meshes. *SIAM Journal on Scientific Computing*, 41(1):B50–B68, 2019.

[30] Kim W. J., Van S. H., and Kim D. H. Measurement of flows around modern commercial ship models. *Experiments in Fluids*, 31:567–578, 2001.

[31] Kelly D. W., Gago J. P. de S. R., Zienkiewicz O. C., and Babuška I. A posteriori error analysis and adaptive processes in the finite element method: Part I–Error analysis. *International Journal for Numerical Methods in Engineering*, 19(11):1593–1619, 1983.

[32] Gago J. P. de S. R., Kelly D. W., Zienkiewicz O. C., and Babuška I. A posteriori error analysis and adaptive processes in the finite element method: Part II–Adaptive mesh refinement. *International Journal for Numerical Methods in Engineering*, 19(11):1621–1656, 1983.

[33] Beck R. F. Time-domain computations for floating bodies. *Applied Ocean Research*, 16:267–282, 1994.

[34] Shoemake K. Animating rotation with quaternion curves. *ACM SIGGRAPH Computer Graphics*, 19(3):245–254, 1985.

[35] Bangerth W., Hartmann R., and Kanschat G. deal.II – A general-purpose object-oriented finite element library. *ACM Transactions on Mathematical Software*, 33(4):24, 2007.

[36] Bangerth W., Davydov D., Heister T., Heltai L., Kanschat G., Kronbichler M., Maier M., Turcksin B., and Wells D. The deal.II library, version 8.4. *Journal of Numerical Mathematics*, 24(3):135–141, 2016.

[37] Mola A., Heltai L., and De Simone A. A stable and adaptive semi-lagrangian potential model for unsteady and nonlinear ship-wave interactions. *Engineering Analysis with Boundary Elements*, 37(1), 2013.

[38] Lachat J. C. and Watson J. O. Effective numerical treatment of boundary integral equations: a formulation for three-dimensional elastostatics. *International Journal for Numerical Methods in Engineering*, 10(5):991–1005, 1976.

[39] Hindmarsh A. C., Brown P. N., Grant K. E., Lee S. L., Serban R., Shumaker D. E., and Woodward C. S. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software*, 31(3):363–396, 2005.

[40] Mola A., Heltai L., and De Simone A. Potential model for ship hydrodynamics simulations directly interfaced with CAD data structures. In *The 24th International Ocean and Polar Engineering Conference*, volume 4, pages 815–822. International Society of Offshore and Polar Engineers, International Society of Offshore and Polar Engineers, 2014.

[41] Dassi F., Mola A., and Si H. Curvature-adapted remeshing of CAD surfaces. *Engineering with Computers*, 34(3):565–576, 2018.

[42] Olivieri A., Pistani F., Avanzini A., Stern F., and Penna R. Towing tank experiments of resistance, sinkage and trim, boundary layer, wake, and free surface flow around a naval combatant INSEAN 2340 model. Technical Report 421, Iowa Institute of Hydraulic Research (IIHR), 2001.

[43] Mola A., Heltai L., and De Simone A. Wet and dry transom stern treatment for unsteady and nonlinear potential flow model for naval hydrodynamics simulations. *Journal of Ship Research*, 61(1):1–14, 2017.

[44] Sederberg T. W. and Parry S. R. Free-form deformation of solid geometric models. *ACM SIGGRAPH Computer Graphics*, 20(4):151–160, 1986.

[45] Salmoiraghi F., Scardigli A., Telib H., and Rozza G. Free-form deformation, mesh morphing and reduced-order methods: enablers for efficient aerodynamic shape optimisation. *International Journal of Computational Fluid Dynamics*, 32(4–5):233–247, 2018.

[46] Koopman B. O. Hamiltonian systems and transformation in hilbert space. *Proceedings of the National Academy of Sciences*, 17(5):315–318, 1931.

[47] Schmid P. J. Dynamic mode decomposition of numerical and experimental data. *Journal of Fluid Mechanics*, 656:5–28, 2010.

[48] Demo N., Tezzele M., and Rozza G. PyDMD: Python dynamic mode decomposition. *Journal of Open Source Software*, 3(22):530, 2018.

[49] Demo N., Tezzele M., Gustin G., Lavini G., and Rozza G. Shape optimization by means of proper orthogonal decomposition and dynamic mode decomposition. In *Technology and Science for the Ships of the Future: Proceedings of NAV 2018: 19th International Conference on Ship & Maritime Research*, pages 212–219. IOS Press, 2018.

[50] Constantine P. G. *Active subspaces: Emerging ideas for dimension reduction in parameter studies*, volume 2 of *SIAM Spotlights*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2015.

[51] Tezzele M., Ballarin F., and Rozza G. Combined parameter and model reduction of cardiovascular problems by means of active subspaces and POD-Galerkin methods. In Boffi D., Pavarino L. F., Rozza G., Scacchi S., and Vergara C., editors, *Mathematical and Numerical Modeling of the Cardiovascular System and Applications*, volume 16 of *SEMA-SIMAI Series*, pages 185–207. Springer International Publishing, 2018.

[52] Tezzele M., Salmoiraghi F., Mola A., and Rozza G. Dimension reduction in heterogeneous parametric spaces with application to naval engineering shape design problems. *Advanced Modeling and Simulation in Engineering Sciences*, 5(1):25, 2018.