A Preconditioner for Free-Surface Hydrodynamics BEM

Gabriele Ciaramella, Marco Gambarini, and Edie Miglio

1 Introduction

The computation of hydrodynamic loads from sea surface waves on large arrays of objects is of physical and engineering interest. Typical applications are the simulation of arrays of wave energy converters [3] and the modeling of ice floes in the marginal ice zone [6]. The interest is in array sizes of the order of tens (for wave energy converter arrays) to hundreds (for ice floes) of objects. In these scenarios, the relatively small distances between the floating objects make the correct simulation of mutual hydrodynamic interactions essential. Under the assumptions of incompressible, irrotational, inviscid flow and small displacements, one can derive a linear potential model, which is widely used for the considered range of applications. This model is discretized using the boundary element method [2], resulting in a linear system characterized by a dense and complex matrix. The dimension of the discrete problem grows proportionally to the number of simulated objects. In general, iterative solvers are not scalable for the corresponding numerical solution: the number of iterations needed to achieve a given tolerance grows with the number of objects [5]. To tackle this problem, we propose a preconditioner for the efficient simulation of large arrays of objects and present its implementation using hierarchical matrices.

Consider an array of n floating objects. To compute all its hydrodynamic properties, a number of problems equal to the number of its degrees of freedom needs to be solved. Each problem corresponds to imposing a unit oscillation in one of the degrees of freedom, while keeping all others fixed. Exploiting linearity, the solution of the dynamic problem with loads from incident waves and possibly other external forces can then be written as a linear combination of such unit oscillations. Considering only vertical oscillations, system (1) needs to be solved for $i = 1, \dots, n$

Gabriele Ciaramella, Marco Gambarini, Edie Miglio
MOX, Dipartimento di Matematica Politecnico di Milano, Italy, e-mail:
gabriele.ciaramella@polimi.it, marco.gambarini@polimi.it, edie.miglio@polimi.it

$$\begin{cases} \Delta\phi = 0 & \text{in } \Omega \subset \mathbb{R}^3, \\ \dfrac{\partial\phi}{\partial n} = 0 & \text{on } \Gamma_b, \\ \dfrac{\partial\phi}{\partial z} - \dfrac{\omega^2}{g}\phi = 0 & \text{on } \Gamma_s, \\ \dfrac{\partial\phi}{\partial n} = n_z & \text{on } \Gamma_{o,i}, \\ \dfrac{\partial\phi}{\partial n} = 0 & \text{on } \Gamma_{o,j}, \quad j = 1, \dots, n \wedge j \neq i, \end{cases} \tag{1}$$

where ϕ is the velocity potential, Ω is the (3D) domain, bounded by the sea bottom Γ_b, the mean free surface Γ_s, and the immersed surfaces of the objects $\Gamma_{o,i}$, $i = 1, \dots, n$. Further, ω is the angular frequency of oscillations, g is the gravitational field, and n_z is the vertical component of the normal vector to the surface of objects. The numerical solution using a source-distribution boundary element method (BEM) is based on recasting (1) in integral form:

$$\frac{1}{2}\sigma(\boldsymbol{x}) + \int_{\cup_k \Gamma_{o,k}} \sigma(\boldsymbol{x}') \frac{\partial \mathcal{G}}{\partial n}(\boldsymbol{x}; \boldsymbol{x}')\, \mathrm{d}\boldsymbol{x}' = \begin{cases} n_z & \text{if } \boldsymbol{x} \in \Gamma_{o,i}, \\ 0 & \text{if } \boldsymbol{x} \in \Gamma_{o,j}, \quad j \neq i, \end{cases} \tag{2}$$

$$\phi(\boldsymbol{x}) = \int_{\cup_k \Gamma_{o,k}} \sigma(\boldsymbol{x}') \mathcal{G}(\boldsymbol{x}; \boldsymbol{x}')\, \mathrm{d}\boldsymbol{x}', \quad \forall \boldsymbol{x} \in \Omega. \tag{3}$$

Here, the unknown is the source distribution σ defined on body surfaces. The kernel is the Green function $\mathcal{G}$, a complex elementary solution of the Laplace equation satisfying the boundary conditions on the bottom and free surface [7, Sect. 16]. By discretizing the surfaces of objects into elements, Eq. (2) can be represented as the linear algebraic system $A\boldsymbol{\sigma} = \boldsymbol{b}$. Once this system has been solved, Eq. (3), in the discretized form $\boldsymbol{\phi} = B\boldsymbol{\sigma}$, can be used to compute the potential in any point of the domain.

2 The coarse-corrected block-Jacobi algorithm

The matrix A resulting from the discretization of Eq. (2) is full, because each element interacts with all others. Moreover, even though the Green function is symmetric with respect to an exchange of its arguments, matrix A is non-symmetric because interacting elements have in general different areas and orientations. The problem has a natural block structure

$$A = \begin{bmatrix} A_{11} & \cdots & A_{1n} \\ \vdots & \ddots & \vdots \\ A_{n1} & \cdots & A_{nn} \end{bmatrix}, \quad \boldsymbol{\sigma} = \begin{bmatrix} \boldsymbol{\sigma}_1 \\ \vdots \\ \boldsymbol{\sigma}_n \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} \boldsymbol{b}_1 \\ \vdots \\ \boldsymbol{b}_n \end{bmatrix}, \tag{4}$$

where $\boldsymbol{\sigma}_j$ is a vector containing the unknowns corresponding to the j-th object. The diagonal block A_{ii} represents the interaction of body i with itself. The off-diagonal block A_{ij} represents the effect on body i of waves radiated from body j. The structure of (4) suggests the use of a block-Jacobi algorithm, equivalent to the parallel method of reflections [5]. This method, together with a coarse correction, has been presented in [5] for the real Laplace equation in perforated domains. Block-Jacobi is based on the splitting $A = D - N$, where D is the block-diagonal part of A. At each iteration, starting from $\boldsymbol{\sigma}^k$, it requires solving for $\boldsymbol{\sigma}^{k+1/2}$ in

$$D\boldsymbol{\sigma}^{k+1/2} = N\boldsymbol{\sigma}^k + \boldsymbol{b}. \tag{5}$$

The solution of (5) can be performed block by block in parallel. After the block-Jacobi step, a coarse correction is performed by solving the correction problem $A\boldsymbol{e} = \boldsymbol{r}^{k+1/2}$ in a low-dimensional (coarse) space C, where $\boldsymbol{r}^{k+1/2} = \boldsymbol{b} - A\boldsymbol{x}^{k+1/2}$ is the residual. Consider, for simplicity, a problem with n identical bodies, each one discretized with p elements, so that the full system has dimension np. Define $C = \text{span}\{\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_m\}$, $m \ll np$. Then we can introduce a restriction operator $\mathcal{R}\colon \mathbb{R}^{np} \longrightarrow C$ represented by matrix $R = \begin{bmatrix}\boldsymbol{c}_1\ \boldsymbol{c}_2\ \ldots\ \boldsymbol{c}_m\end{bmatrix}^T$ and a prolongation operator $\mathcal{P} : \mathbb{R}^{np} \longrightarrow C$ represented by matrix R^T. Let $\boldsymbol{e}_c \in \mathbb{R}^m$ be a vector such that $\hat{\boldsymbol{e}} = R^T\boldsymbol{e}_c$ is an approximation of the error $\boldsymbol{e}$. The coarse problem is

$$RAR^T\boldsymbol{e}_c = R\boldsymbol{r}^{k+1/2}, \tag{6}$$

where $A_c := RAR^T$. Once the coarse problem (6) has been solved, the update

$$\boldsymbol{\sigma}^{k+1} = \boldsymbol{\sigma}^{k+1/2} + R^T\boldsymbol{e}_c$$

is performed. The efficiency of the correction step is strongly related to the choice of the coarse space C. This has to be rich enough to well represent the main error components that block-Jacobi cannot deal with, but its dimension m must be relatively small, so that the cost of a single iteration is not increased significantly. A simple choice for the coarse space is taking a constant value of the source distribution σ on each body. This choice is suggested by the one presented in [5] and corresponds to $\boldsymbol{c}_i := \mathbf{1}_i$, $i = 1, \ldots, n$, $\mathbf{1}_i$ being the discrete indicator function of the i-th object. In this case, the dimension of C is equal to the number of objects n.

Our two-level block-Jacobi method is detailed in Algorithms 1 and 2. The former is a precomputation step, that does not depend on the right-hand side vector. Thus, if multiple systems with the same matrix and different right hand sides need to be solved, Alg. 1 needs to be performed only once. In this algorithm, matrix $\widetilde{R} = RA$ is efficiently (see Section 3) computed, so that the cost for computing the restricted residual at each iteration is reduced. Alg. 2 corresponds to the stationary method

$$\begin{aligned}\boldsymbol{\sigma}^{k+1} &= [I - (P_c + D^{-1} - P_cD^{-1})A]\boldsymbol{\sigma}^k + (P_c + D^{-1} - P_cD^{-1})\boldsymbol{b}\\ &= \boldsymbol{\sigma}^k + (P_c + D^{-1} - P_cD^{-1})\boldsymbol{r}^k,\end{aligned}$$

Algorithm 1 Two-level block-Jacobi algorithm: initialization

```
1: for $i = 1$ to $n$ do
2:     Compute the LU decomposition of $A_{ii}$.
3: end for
4: Compute $\widetilde{R} = RA$, $A_c = \widetilde{R}R^T$.
```

Algorithm 2 Two-level block-Jacobi algorithm: solution

```
Require: Initial guess $\boldsymbol{\sigma}^0$, tolerance $tol$, maximum number of iterations $maxit$.
 1: Set $k = 0$.
 2: while $\|\boldsymbol{b} - A\boldsymbol{\sigma}^k\| > tol$ and $k < maxit$ do
 3:     Compute $\boldsymbol{q} = \boldsymbol{b} - N\boldsymbol{\sigma}^k$.
 4:     for $i = 1$ to $n$ do
 5:         Solve $A_{ii}\boldsymbol{\sigma}_i^{k+1/2} = \boldsymbol{q}_i$ using the LU decomposition of $A_{ii}$.
 6:     end for
 7:     Compute the restricted residual $\boldsymbol{r}_c = R\boldsymbol{b} - \widetilde{R}\boldsymbol{\sigma}^{k+1/2}$.
 8:     Solve for $\boldsymbol{e}_c$ in $A_c\boldsymbol{e}_c = \boldsymbol{r}_c$.
 9:     Update $\boldsymbol{\sigma}^{k+1} = \boldsymbol{\sigma}^{k+1/2} + R^T\boldsymbol{e}_c$.
10:     Update $k = k + 1$.
11: end while
```

with $P_c = R^T A_c^{-1} R$ and where we can recognize the inverse preconditioner $P^{-1} = P_c + D^{-1} - P_c D^{-1}$. Such preconditioner can then be used to accelerate a Krylov method. Using P^{-1}, the system is recast as $P^{-1}A\boldsymbol{\sigma} = P^{-1}\boldsymbol{b}$. Since the new system matrix $P^{-1}A$ is not symmetric, a classical choice is GMRES. In our implementation, the preconditioning matrix P^{-1} is not assembled explicitly; instead, GMRES is provided with a function (based on Alg. 2) computing the action of $P^{-1}A$ on an arbitrary vector.

3 Implementation details and $\mathcal{H}$-matrices

Hierarchical matrices, denoted here as $\mathcal{H}$-matrices, are an efficient tool for reducing the storage and computational cost of BEM problems. The method is based on defining a hierarchical cluster tree from the set of mesh elements. The system matrix is then built with a hierarchical block structure accordingly. Each block describes the interaction between two clusters of elements. If the centers of the two clusters are farther than a threshold, then a low-rank approximation on the block is built; otherwise, the block is built in dense form. If the tree is balanced and if we take as leaves of the tree the single objects, discretized with p elements, then both the costs of storage and of matrix-vector multiplication are $O(\max(r, p)np\log(np))$ [4, Th. 2.6, 2.8], where r is the maximum rank of matrix blocks.

Fig. 1 shows the tree and the hierarchical structure of matrix A for an example with 10 objects on a row, with spacing of 5 m. The ordinate of each node in the tree is the distance between the centers of its sons. In the matrix, blue blocks are dense, while white blocks are low-rank. Notice that dense blocks gather mostly close to

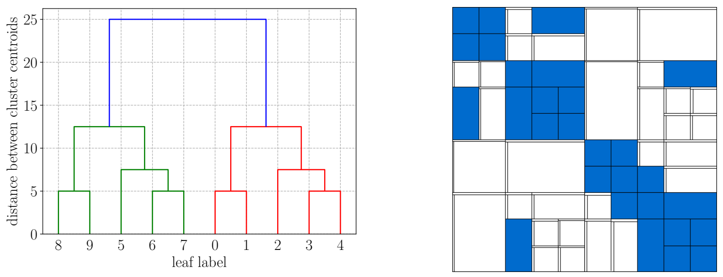


Fig. 1 Clustering of positions (left) and hierarchical structure of matrix A (right) for a test with 10 objects.

the diagonal. Information on the nodes of the tree is stored in the so-called linkage matrix. The leaves constitute the first n nodes of the tree. All other nodes are defined by the rows of the linkage matrix: its i-th row contains the labels of the sons of the $(n+i)$-th node.

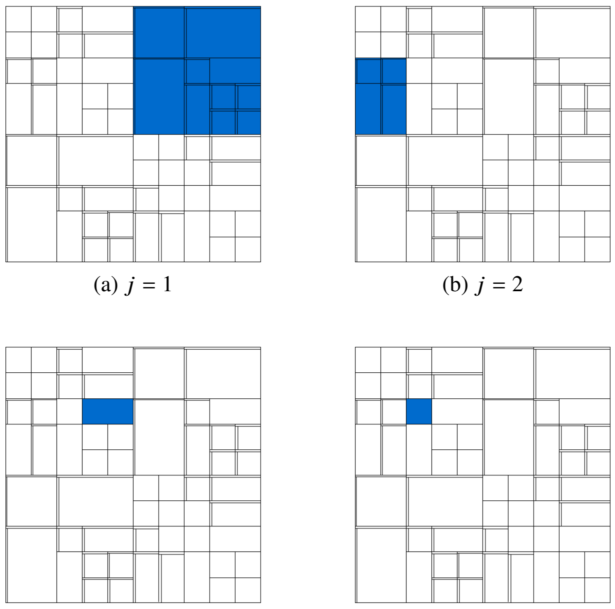
Our implementation of Alg. 1 and 2 is done starting from the BEM code Capytaine [2], which includes an $\mathcal{H}$-matrices engine. Matrix-vector multiplication in line 3 of Alg. 2 is performed with the built-in routine. Handling of diagonal blocks and building the coarse space, instead, require special care. Since the structure of matrix storage is hierarchical, extracting the diagonal blocks of A to build matrices D and N is not immediate. In order to do it, first a list `leaves`, whose i-th element is the list of leaves belonging to node i, is built by sweeping over the rows of the linkage matrix. Then, a list `paths` is constructed. This contains n sublists. The i-th sublist has size equal to the level of the i-th leaf. The ℓ-th element of this sublist is equal to 0 or 1, if at the ℓ-th level one has to turn left or right, respectively, to step down toward the i-th leaf. For example, in Fig. 1 (left) the path to leaf 5 is `paths`$[5] = [0, 1, 0]$. The lists `leaves` and `paths` are exploited to compute RA efficiently in Alg. 2. Because of the sparsity of the rows of R, that are vectors $\mathbf{c}_i^T$, for dense matrices this operation can be made very efficient by multiplying each of the $\mathbf{c}_i$ only by the rows of A corresponding to its non-zero elements. Slicing a hierarchical matrix, however, is not as trivial. For this reason, we propose the recursive procedure detailed in Alg. 3 and described graphically in Fig. 2. At the beginning, $\widetilde{A} = A$ and $\boldsymbol{v} = \boldsymbol{c}_i$ are set. The algorithm then descends from the root to the level above the i-th leaf following list `path` $=$ `paths`$[i]$. In doing this, because of the structure of the tree, 2×2 blocks are encountered at each level. At level j, the nonzero contributions $\boldsymbol{c}_i A$ come only from the k-th block-row, with $k =$ `path`$[j]$. The off-diagonal part of the k-th block row is directly multiplied by the appropriate slice of $\boldsymbol{v}$; then, the algorithm is applied again to the diagonal block $\widetilde{A}_{kk}$. At the end, only the (dense) diagonal block corresponding to the interaction of the i-th object with itself is left, and this last multiplication is performed. The main advantage of this strategy is that, at each level of the hierarchy except the last, off-diagonal blocks, that are expected to be mostly low-rank, are multiplied.

Algorithm 3 Computation of $\widetilde{R} = RA$ for $\mathcal{H}$-matrices

```
Require: $A$, $R = [\boldsymbol{c}_1, \cdots, \boldsymbol{c}_n]$ and paths.
 1: for $i = 1$ to $n$ do
 2:     Select the path to the $i$-th leaf: path = paths[$i$].
 3:     Set $\widetilde{A} = A$, $\boldsymbol{v} = \boldsymbol{c}_i$, $a = 0$, $b = np$, and initialize a zero array $\boldsymbol{d}$ of size $np$.
 4:     for $j = 1$ to length(path) do
 5:         Set $k$ = path[$j$] and $N_{row}$ as the number of rows of $\widetilde{A}_{kk}$.
 6:         if $k = 0$ then
 7:             Select the first $N_{row}$ rows of $\boldsymbol{v}$: $\boldsymbol{v} = \boldsymbol{v}[0 : N_{row}]$.
 8:             Multiply $\boldsymbol{w} = \boldsymbol{v}\widetilde{A}_{01}$ and set $\boldsymbol{d}[b - \text{length}(\boldsymbol{w}) : b] = \boldsymbol{w}$.
 9:             Update: $b = b - \text{length}(\boldsymbol{w})$.
10:         else
11:             Select the last $N_{row}$ rows of $\boldsymbol{v}$: $\boldsymbol{v} = \boldsymbol{v}[\text{end} - N_{row} : \text{end}]$.
12:             Multiply $\boldsymbol{w} = \boldsymbol{v}\widetilde{A}_{10}$ and set $\boldsymbol{d}[a : a + \text{length}(\boldsymbol{w})] = \boldsymbol{w}$.
13:             Update: $a = a + \text{length}(\boldsymbol{w})$.
14:         end if
15:         Set $\widetilde{A} = \widetilde{A}_{kk}$.
16:     end for
17:     Diagonal block multiplication: $\boldsymbol{d}[a : b] = \boldsymbol{v}\tilde{A}$.
18:     Set $\widetilde{R}[i, :] = \boldsymbol{d}$.
19: end for
```

4 Numerical experiments

The method is implemented by integration with the BEM code Capytaine [2]. Hierarchical clustering on the positions of the objects is performed using SciPy. We simulate two geometries: line arrays and grid arrays. In both cases the objects are half-spheres of radius 2 m and the minimum distance between two bodies is 5 m. The results are reported in Table 1. Times for GMRES and preconditioned GMRES refer to the solution of the n systems required to build the radiation dataset; thus the number of systems needing to be solved increases with the number of objects. The loops described in Algorithms 2 and 3 are performed serially. We build the radiation dataset only for vertical motion; in the general case of a rigid body, $6n$ systems would need to be solved. In some cases, the number of iteration varies depending on the right hand side (i.e., depending on the radiating object).



(a) $j = 1$ (b) $j = 2$

(c) $j = 3$ (d) Diagonal block

Fig. 2 Blocks selected for multiplication in Algorithm 3 for leaf $i = 5$.

Table 1 Results of the numerical experiment. Top table: line geometry. Bottom table: grid geometry. In both tables, init is the time for initializing the coarse solver (coarse space definition).

		GMRES			Preconditioned GMRES			
n	storage (%)	niter	t (s)	t/n (s)	init (s)	niter	t (s)	t/n (s)
80	6.19	12	18	0.23	0.47	7	15	0.19
160	3.32	13	107	0.67	1.05	7	68	0.85
240	2.33	14	449	1.87	1.64	7	294	1.23
320	1.75	14-15	831	2.60	2.51	7	487	1.52
400	1.43	15	1182	2.95	3.46	7	698	1.74
480	1.23	15-16	1715	3.57	4.14	7	993	2.07

		GMRES			Preconditioned GMRES			
n	storage (%)	niter	t (s)	t/n (s)	init (s)	niter	t (s)	t/n (s)
16	48.42	10	0.51	0.03	0.07	7	1.03	0.06
64	20.00	13	35	0.55	0.87	8	25	0.40
144	10.82	17	281	1.95	3.40	8-9	157	1.09
256	6.72	24-26	2216	8.66	15.5	9	826	3.23
400	4.82	38-43	7215	18.03	23.5	9-10	2030	5.08

5 Discussion and conclusions

The presented results indicate that the preconditioned GMRES method has a lower cost than the standard GMRES method for large arrays of floating objects. The advantage becomes larger as the number of bodies increases: speedups of up to a factor of 3.5 are obtained. For the line geometry, the number of iterations of GMRES tends to become constant with respect to the number of objects, while the iterations of preconditioned GMRES remain exactly constant and equal to 7. On the other hand, for the grid geometry the number of iterations of GMRES increases as n grows, while preconditioned GMRES scales well. The use of Alg. 3 for the construction of the coarse space, which needs to be performed only once, keeps the cost of such operation low. Thus, a substantial speedup can be obtained with respect to standard GMRES even when a small subset of the entire radiation dataset needs to be computed. In the grid test case the percentage of dense blocks is larger, resulting in a larger time for the initialization of Alg. 3.

Possible improvements include the parallelization of the loops in Alg. 2 and the use of a preconditioner also for the solution of the coarse problem, whose cost can become relevant for very large arrays. In the case of a single row of bodies, the coarse matrix A_c has a Toeplitz structure, and the natural choice in this case is to use a circulant preconditioner. This strategy has been explored at block level in [1], while some choices of circulant preconditioners are presented in [8].

References

1. Ancellin, M. and Dias, F. Using the floating body symmetries to speed up the numerical computation of hydrodynamics coefficients with Nemoh. Proceedings of the 37th International Conference on Ocean, Offshore and Artic Engineering (2018).
2. Ancellin, M. and Dias, F. Capytaine: a Python-based linear potential flow solver. *J. Open Source Softw.* **4**(36), 1341 (2019).
3. Babarit, A. On the park effect in arrays of oscillating wave energy converters. *Renewable Energy* **58**, 68–78 (2013).
4. Bebendorf, M. *Hierarchical Matrices*. Lecture Notes in Computational Science and Engineering. Springer Berlin, Heidelberg (2008).
5. Ciaramella, G., Gander, M. J., Halpern, L., and Salomon, J. Methods of Reflections: relations with Schwarz methods and classical stationary iterations, scalability and preconditioning. *SMAI J. Comput. Math* **5**, 161–193 (2019).
6. Squire, V. A. Ocean wave interactions with sea ice: A reappraisal. *Annual Review of Fluid Mechanics* **52**(1), 37–60 (2020).
7. Wehausen, J. V. and Laitone, E. V. Surface waves. In: Truesdell, C. (ed.), *Fluid Dynamics / Strömungsmechanik*, 446–778. Springer Berlin Heidelberg (1960).
8. Zhu, Z. and Wakin, M. B. On the asymptotic equivalence of circulant and Toeplitz matrices. *IEEE Transactions on Information Theory* **63**(5), 2975–2992 (2017).