Fast solution of boundary integral equations for elasticity around a crack network: a comparative study

Ibtihel Ben Gharbia, Marcella Bonazzoli, Xavier Claeys, Pierre Marchand, Pierre-Henri Tournier

Fast solution of boundary integral equations for elasticity around a crack network: a comparative study*

Ibtihel Ben Gharbia[1], Marcella Bonazzoli[2], Xavier Claeys[3], Pierre Marchand[3], and Pierre-Henri Tournier[3]

[1] *IFP Energies nouvelles, 1 et 4 avenue de Bois-Préau, 92852 Rueil-Malmaison. E-mail: ibtihel.ben-gharbia@ifpen.fr*
[2] *Université Côte d'Azur, CNRS, LJAD, France. E-mail: bonazzoli@ljll.math.upmc.fr*
[3] *Sorbonne Universités, UPMC Univ Paris 06, CNRS, INRIA, UMR 7598, Labo. Jacques-Louis Lions, équipe Alpines, 75005, Paris. E-mail: claeys@ann.jussieu.fr, pierre.marchand@inria.fr, tournier@ljll.math.upmc.fr*

Abstract

Because of the *non-local* nature of the integral kernels at play, the discretization of boundary integral equations leads to dense matrices, which would imply high computational complexity. Acceleration techniques, such as hierarchical matrix strategies combined with Adaptive Cross Approximation (ACA), are available in literature. Here we apply such a technique to the solution of an elastostatic problem, arising from industrial applications, posed at the surface of highly irregular cracks networks.

Introduction

Many applications involve the solution to an elliptic boundary value problem in a background medium perturbed by the presence of cracks that take the form of one or many pieces of surface (with boundary). Crack (also called "screens" in electrical engineering) problems arise in all classical fields of applied physics: acoustics [14, 9], electromagnetics [15, 5] and elasticity [8, 4]. Such problems are of particular interest in industrial applications related to geophysics that often involve fractures and dislocations.

When the background medium can be considered as homogeneous, which is a valid approximation in many cases, boundary integral equations appear as a method of choice for the numerical solution to crack problems. With such an approach, the problem is reformulated as a fully non-local equation posed at the surface of cracks. This is the strategy adopted by IFP Energies Nouvelles (IFPEN) for the evolution of the deformation and perturbed stress field associated with the solution of an elastostatic problem around a network composed of multiple cracks. The latter problem has motivated the present contribution. Most of the existing literature considers either the case of cracks that remain well separated, or the case of a few faults located at regular pieces of manifolds. A salient feature of the present contribution, concerns the geometry under consideration where cracks intersect each other forming a geometrically highly irregular structure, see Figure 1.

Discretization of boundary integral equations by means of a Galerkin procedure, resulting in the so-called Boundary Element Method (BEM), leads to densely populated matrices due to the full non-locality of the operators under consideration. Then, if the matrix of the problem is of size N, any matrix-vector product (the most elementary operation in any iterative linear solver) requires at least $\mathcal{O}(N^2)$ operations. This is clearly not acceptable as it requires unreasonable computational effort, especially in an industrial context where large size problems usually arise.

*This work received support from ANR research grant ANR-15-CE23-0017-01 and from IFP Energies Nouvelles.

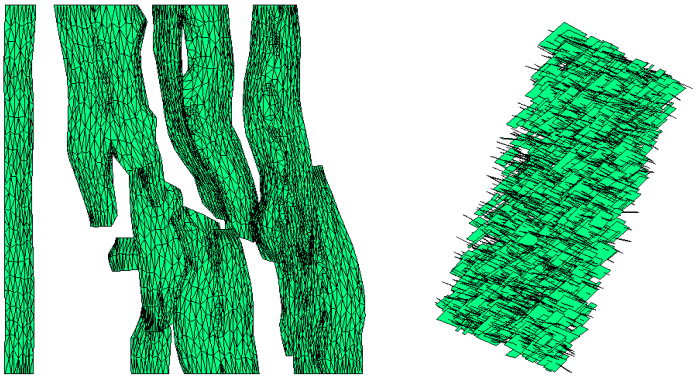


Figure 1: Examples of crack networks provided by IFPEN: fault network (left) and discrete fracture network (right).

To circumvent the computational complexity issue, current literature offers a panel of refined acceleration techniques: fast multipole methods, hierarchical matrix strategies, and the like. These techniques, that have been developed during the last two decades, have been introduced to accelerate computations in a wide variety of problems ranging from molecular dynamics [3] to astrophysics [1]. For a general overview see [6]. Acceleration of boundary integral equations on smooth surfaces has also been historically a key challenge for stimulating the development of such methods [12].

To reduce computational complexity, IFPEN did not adopt one of the currently available acceleration techniques mentioned above, but rather developed its own approach, which shall be referred to as "sparsification", consisting in forcing coefficients of the BEM matrix to zero whenever the corresponding interaction involves sufficiently distant points of the computational domain. This sparsification procedure is implementation friendly, and approximates the originally fully populated matrix with a sparse counterpart that allows fast matrix-vector products. On the other hand, this strategy also induces substantial consistency error: measured in relative Frobenius norm, the perturbation on the matrix is typically 30% large.

The main objective of the present contribution is to compare the performance of sparsification with another well established method of the current literature: the Hierarchical Matrix [7] format combined with the Adaptive Cross Approximation (ACA) compression method [2]. We chose to consider this alternative method, later referred to as HM-ACA, because this is one of the only existing approaches that treats generation of the matrix of the problem in a fully black-box manner.

Although HM-ACA has already been analyzed in detail, and was proved to perform well on classical boundary integral equation problems, even in industrial contexts (see e.g. [10]), the present geometry with possibly many intersecting fractures is highly non regular, and thus cannot be considered a classical test case. We shall indeed see that HM-ACA not always achieves both accuracy and high compression. We will show that whether this strategy is preferable in terms of compression rate to sparsification depends on certain geometrical parameters of the problem related to the density of cracks.

The outline of this contribution is as follows. We first present in Section 1 the problem under consideration, its discretization and the way IFPEN sparsifies the obtained matrix. Then in Section 2.1 we introduce the adaptive cross approximation(ACA) method and show its efficiency for the compression of dense matrices admitting fast decreasing singular values (such matrices shall be referred to in the sequel as admissible). However, BEM matrices are not admissible because of the singularity of the integral kernel. Thus, we introduce in Section 2.2 a recursive splitting algorithm which decomposes the matrix into admissible sub-blocks in which the ACA compression method can be applied. This algorithm produces so-called Hierarchical Matrices and is referred to as the HM-ACA method. Then, in Section 3, we give an overview of the code we developed, and present a series of test cases to compare HM-ACA to the sparsification procedure. The numerical results obtained will lead us to

conclude and provide an outlook for our work.

1 Initial problem, exact and sparsified matrices

The matrices we consider in the present contribution stem for the Galerkin discretization of the boundary integral formulation of some elastostatic problem. We will thus start by briefly describing this underlying continuous formulation as well as the associated discretization. We will also present the sparsification heuristic used so far at IFPEN in order to decrease the algorithmic complexity of matrix-vector products.

1.1 Underlying continuous problem

We are primarily interested in the solution of an elastostatic problem, looking for an equilibrium displacement field $\boldsymbol{v}$ in the exterior of a dislocation surface $S \subset \mathbb{R}^3$ that consists of the union of a collection $\mathcal{T} = \{\tau_j\}_{j=1}^N$ of flat polygons

$$S = \bigcup_{j=1}^{N} \tau_j \ , \qquad \tau_j = \text{polygons.}$$

The elements τ_j might intersect each other, which makes S a potentially very rough surface, see Figure 1. The stress field $\sigma(\boldsymbol{v})$ is in equilibrium over the area $\mathbb{R}^3 \backslash S$, and is submitted to a traction field $\boldsymbol{t} : S \to \mathbb{R}^3$ prescribed at S, which leads to the following system of equations

$$\begin{cases} \operatorname{div} \sigma(\boldsymbol{v}) = 0 \quad \text{in } \mathbb{R}^3 \backslash S, \\ [\![\sigma(\boldsymbol{v}) \cdot \boldsymbol{n}]\!] = 0 \quad \text{on } S, \\ \sigma(\boldsymbol{v}) \cdot \boldsymbol{n} = \boldsymbol{t} \quad \text{on } S, \\ \limsup_{|\boldsymbol{x}| \to \infty} |\boldsymbol{x}| \, |\boldsymbol{v}(\boldsymbol{x})| < +\infty. \end{cases} \tag{1}$$

Considering a normal $\boldsymbol{n}_\tau$ to each polygon $\tau \in \mathcal{T}$, the jump operator in (1) is defined on each τ by $[\![\boldsymbol{p}]\!]|_\tau := \boldsymbol{p}|_\tau^+ - \boldsymbol{p}|_\tau^-$ where $\boldsymbol{p}|_\tau^+$ denotes the trace of $\boldsymbol{p}$ taken from the side of τ where $\boldsymbol{n}_\tau$ is ingoing, and $\boldsymbol{p}|_\tau^-$ refers to the trace on the other side. Since we assume that the elastic solid is homogenous and isotropic, the stress field is given by the Hooke's Law in 3D $\sigma(\boldsymbol{v}) = 2\mu\epsilon(\boldsymbol{v}) + \lambda \mathrm{tr}(\epsilon(\boldsymbol{v}))\,\mathrm{Id}$, with $\epsilon(\boldsymbol{v}) = \frac{1}{2}(\nabla \boldsymbol{v} + \nabla \boldsymbol{v}^T)$ is the strain tensor and Id is the identity matrix. The two material parameters λ and μ are known as Lamé coefficients and verify the following relations

$$\lambda = \frac{E\nu}{(1+\mu)(1-2\nu)} \quad \text{and} \quad \mu = \frac{E}{2(1+\nu)},$$

where ν refers to the Poisson ratio, and E is Young's modulus. We are interested in a boundary integral reformulation of problem (1) that will consist in a non-local equation posed only on the surface S. Without giving too much detail, let us describe the bilinear form associated with such a formulation. First, we need to introduce the Green kernel of problem (1), which is given by the explicit formula

$$\mathscr{G}(\boldsymbol{x}) = \frac{1}{8\pi E} \frac{1+\nu}{1-\nu} \left(\frac{3-4\nu}{|\boldsymbol{x}|} \mathrm{Id} + \frac{\boldsymbol{x} \cdot \boldsymbol{x}^T}{|\boldsymbol{x}|^3} \right).$$

It is fundamental to observe, and to keep in mind, that this function is singular at $\boldsymbol{x} = 0$. Next, for any vector field $\boldsymbol{v} \colon S \to \mathbb{R}^3$, define the trace operator

$$T_\tau(\boldsymbol{v}) = \lambda \boldsymbol{n}_\tau \mathrm{div}(\boldsymbol{v}) + 2\mu \, (\boldsymbol{n}_\tau \cdot \nabla)\boldsymbol{v} + \mu \boldsymbol{n}_\tau \times \mathbf{curl}(\boldsymbol{v}).$$

Although the displacement field $\boldsymbol{v}$ solution to (1) satisfies $[\![\sigma(\boldsymbol{v}) \cdot \boldsymbol{n}]\!] = 0$ on S, a priori it jumps across S according to a slip field $[\![\boldsymbol{v}]\!] = \boldsymbol{u} : S \to \mathbb{R}^3$ that is the unknown of our

boundary integral formulation. The exact solution of problem (1) can be recovered from $\boldsymbol{u}$ by means of a so-called representation formula, see [13, §6.7],

$$\boldsymbol{v}(\boldsymbol{x}) = \int_S T_\tau^{\boldsymbol{y}}(\mathscr{G}(\boldsymbol{x}-\boldsymbol{y}))\boldsymbol{u}(\boldsymbol{y})ds(\boldsymbol{y}).$$

The boundary integral formulation is obtained simply by imposing the third equation of (1) taking the trace at S of the above representation formula. For an appropriate space $\mathbf{H}(S) = \Pi_{\tau\in\mathcal{T}}\mathrm{H}(\tau)^3$ of trace fields, the boundary integral variational formulation associated with problem (1) that we consider writes

$$\begin{array}{l}\text{Find } \boldsymbol{u}\in\mathbf{H}(S) \text{ such that} \quad a(\boldsymbol{u},\boldsymbol{v}) = f(\boldsymbol{v}) \ \ \forall \boldsymbol{v}\in\mathbf{H}(S)\\ \text{where } a(\boldsymbol{u},\boldsymbol{v}) := \displaystyle\sum_{\tau\in\mathcal{T}}\sum_{\tau'\in\mathcal{T}}\int_{\tau\times\tau'} T_\tau^{\boldsymbol{y}}(T_{\tau'}^{\boldsymbol{x}}\mathscr{G}(\boldsymbol{x}-\boldsymbol{y}))\boldsymbol{u}(\boldsymbol{x})\boldsymbol{v}(\boldsymbol{y})ds(\boldsymbol{x})ds(\boldsymbol{y}),\end{array} \tag{2}$$

and the right hand side is $f(\boldsymbol{v}) = \int_S \boldsymbol{t}\cdot\boldsymbol{v}\,ds$. In formula (2) the operator $T_\tau^{\boldsymbol{x}}$ is the operator T_τ applied with respect to the $\boldsymbol{x}$ variable. The operator $T_{\tau'}^{\boldsymbol{y}}$ is defined accordingly. The important feature here is that the integral kernel coming into play in this integral operator is singular at $\boldsymbol{x}=\boldsymbol{y}$ (which is possible only if $\tau\cap\tau'\neq\emptyset$), and it is regular otherwise. In particular, if τ and τ' are distant from each other, then the operator associated with $a(\ ,\ )$ is regularizing, and it will induce matrices with exponentially decreasing singular values.

1.2 Exact BEM matrices

The bilinear form in (2) is discretized by means of a Galerkin procedure, where each space $\mathrm{H}(\tau)$ is approximated by constant functions over τ. As a consequence three degrees of freedom (corresponding to the three directions of space) are associated with each elementary polygon τ, and the discrete variational formulation takes the form

$$\left\{\begin{array}{l}\text{Find } \boldsymbol{u}_h\in\mathbf{H}_h(S) \text{ such that}\\ a(\boldsymbol{u}_h,\boldsymbol{v}_h) = f(\boldsymbol{v}_h) \quad \boldsymbol{v}_h\in\mathbf{H}_h(S),\\ \text{where } \mathbf{H}_h(S) := \{\boldsymbol{v}_h(\boldsymbol{x}) = \sum_{\tau\in\mathcal{T}}\boldsymbol{\alpha}_\tau 1_\tau(\boldsymbol{x}),\ \boldsymbol{\alpha}_\tau\in\mathbb{R}^3\}.\end{array}\right. \tag{3}$$

For this discrete formulation, the order 0 Lagrange vector shape functions are defined as follows: First consider a numbering of the elementary polygons $\mathcal{T} = \{\tau_j\}_{j=1\ldots N}$, and let $\mathbf{e}_k, k=1,2,3$ refer to the canonical basis of $\mathbb{R}^3$. As shape functions, we then choose $\boldsymbol{\psi}_j, j = 1,\ldots,3N$ where $\boldsymbol{\psi}_{k+3(j-1)}(\boldsymbol{x}) := 1_{\tau_j}(\boldsymbol{x})\mathbf{e}_k \quad j=1\ldots N,\ k=1,2,3$. Each $\boldsymbol{\psi}_{3j-q}$ with $q=0,1,2$ is thus regarded as a function defined on S that is supported only on τ_j. The matrices we are dealing with are defined as $\mathrm{A} = (\mathrm{A}_{j,k})_{j,k=1\ldots3N}$ where

$$\mathrm{A}_{j,k} := a(\boldsymbol{\psi}_j,\boldsymbol{\psi}_k), \quad j,k = 1\ldots3N. \tag{4}$$

1.3 Sparsification heuristics

The bilinear form $a(\ ,\ )$ coming into play in (3) is non-local: there is no reason for $a(\boldsymbol{u}_h,\boldsymbol{v}_h)$ to vanish, even if the supports of $\boldsymbol{u}_h$ and $\boldsymbol{v}_h$ are disjoint. The direct consequence of this property is that the matrix $\mathrm{A} = (\mathrm{A}_{j,k})$ is fully populated. Without any special strategy, the computational complexity of a matrix-vector product will then be of order $\mathcal{O}(N^2)$. This is unbearable for any test case of decent size.

An approximation has to be applied to the matrix A in order to break this computational complexity. Let us describe what is the heuristic adopted so far at IFPEN in order to achieve this goal. This strategy shall be referred to later on as "sparsification procedure". First, note that for each pair (j,k), the coefficient $\mathrm{A}_{j,k}$ corresponds to the interaction between two mesh elements τ_j and τ_k. An approximate matrix $\mathrm{A}^{\mathrm{sp}} = (\mathrm{A}^{\mathrm{sp}}_{j,k})_{j,k=1\ldots3N}$ is then obtained by computing only the coefficients $\mathrm{A}_{j,k}$ such that

$$\mathrm{dist}(\tau_j,\tau_k) < \alpha\ \mathrm{diam}(\tau_k), \tag{5}$$

where $\alpha > 0$ is a parameter, $\mathrm{diam}(\tau_k)$ is the diameter of the circumscribed sphere to element τ_k and $\mathrm{dist}(\tau_j, \tau_k)$ is the distance between the barycenters of elements τ_j and τ_k.

Admittedly this may appear as a crude approach. However, this strategy is easily implementable. In addition, which is even more important, one should keep in mind that the requirements in terms of accuracy are rather loose. The main point of the present contribution is to examine whether other more sophisticated strategies may offer a better accuracy/compression trade-off, taking the sparsification procedure described above as a reference.

Figure 2 shows the resulting sparse patterns for two problems considered by IFPEN, corresponding to large faults (left) and a crack network (right). These two test cases will be included in the numerical comparison experiments Section 3.2 and correspond to Figure 7 and 8 respectively.

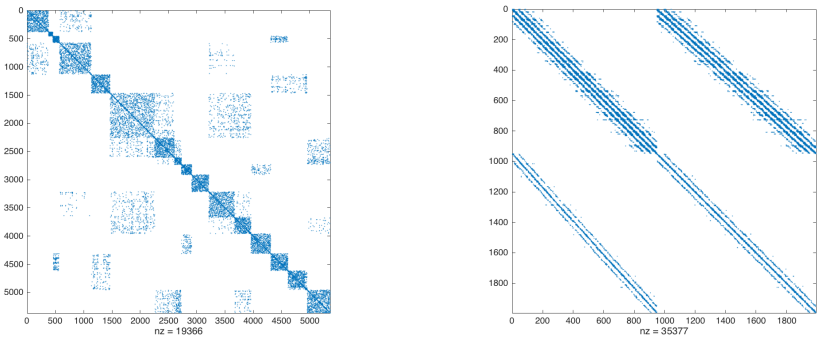


Figure 2: Sparse patterns resulting from the IFPEN heuristic sparsification procedure with $\alpha = 2$. Large faults problem (left), crack network problem (right).

2 Adaptive cross approximation and hierarchical matrices

In this section, we will present the alternative approach that we selected for comparison with the sparsfication procedure of the previous section. This alternative strategy rests on a (classical) combination of Adaptive Cross Approximation (ACA) presented in the next paragraph, and the hierarchical matrix (HM) format presented in a second paragraph.

2.1 Low rank approximation

In this paragraph, we consider a fully populated matrix $\mathrm{A} \in \mathbb{C}^{n\times n}$, $\mathrm{A} = (\mathrm{A}_{j,k})_{j,k=1\ldots n}$ with, a priori, none of its entries vanishing, its size n being potentially large. With no particular assumption on this matrix, the cost of a matrix-vector product is $\mathcal{O}(n^2)$. This cost is substantially reduced if we assume that A is of low rank though. We say that a matrix has the *low rank property*, with rank $k \leq n$, if there exist vectors $\mathbf{u}_j, \mathbf{v}_j \in \mathbb{C}^n, j = 1 \ldots k$, such that

$$\mathrm{A} = \sum_{j=1}^{k} \mathbf{u}_j \cdot \mathbf{v}_j^T \quad \text{with} \quad k < n.$$

Indeed if this representation holds, then a matrix-vector product requires $2kn$ flops, which is smaller than n^2 provided that the condition on k given above is satisfied. Matrices A encountered in applications rarely have the low rank property. A simple primary observation is that general matrices A can be written as a sum of rank one matrices through its Singular

Value Decomposition (SVD)

$$\mathrm{A} = \sum_{j=1}^{n} \sigma_j \, \mathbf{u}_j \cdot \mathbf{v}_j^T \quad \text{where spectrum}(\mathrm{A}^*\mathrm{A}) = \{\sigma_j^2\}_{j=1\dots n}, \tag{6}$$

where $(\mathbf{u}_j)_{j=1}^n, (\mathbf{v}_j)_{j=1}^n$ are orthonormal basis of $\mathbb{C}^n$ and $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$. A closer inspection of this formula leads to the conclusion, provided that the sequence (σ_j) decreases fast, that truncating the singular value decomposition (6) yields a good approximation of the matrix A. This is the essence of the next result (see [7, Appendix C]).

Proposition 2.1. *Let $\mathrm{A} \in \mathbb{C}^{n\times n}$ admit the singular value decomposition (6), and denote $\mathrm{A}^{(k)}$ the matrix obtained by truncating this decomposition at rank k, namely $\mathrm{A}^{(k)} := \sum_{j=1}^{k} \sigma_j \, \mathbf{u}_j \cdot \mathbf{v}_j^T$. Then we have the error estimates*

$$\|\mathrm{A} - \mathrm{A}^{(k)}\|_2^2 = \sigma_{k+1}^2 \quad \textit{and} \quad \|\mathrm{A} - \mathrm{A}^{(k)}\|_{\mathrm{F}}^2 = \sum_{j=k+1}^{n} \sigma_j^2,$$

where $\|\ \|_2$ refers to the matrix norm induced by the vector norm $|\mathbf{u}|_2 = (\sum_{j=1}^n |u_j|^2)$ for $\mathbf{u} = (u_j)_{j=1}^n \in \mathbb{C}$ and $\|\ \|_{\mathrm{F}}$ refers to the Frobenius norm given by $\|\mathrm{A}\|_{\mathrm{F}}^2 = \sum_{j,k=1\dots n} |\mathrm{A}_{j,k}|^2$.

Truncating the SVD is thus an efficient way to approximate a matrix, and so to reduce the cost of the matrix-vector product, provided that the singular values decrease fast. Assume that singular values decrease exponentially, say $\sigma_k \leq q^k$ for a fixed $q \in (0,1)$. Then for a relative error expressed in Frobenius norm of order $\varepsilon > 0$, it suffices to take $k \simeq \log_q \varepsilon$.

Unfortunately, computing the singular value decomposition of a matrix is expensive: it costs $\mathcal{O}(n^3)$ flops. To circumvent this issue, starting from an arbitrary matrix A whose singular values are assumed to decrease exponentially, the *Adaptive Cross Approximation* (ACA) algorithm provides a collection of vectors $\mathbf{u}_j, \mathbf{v}_j \in \mathbb{C}^n, j = 1 \dots n$ such that

$$\|\mathrm{A} - \widetilde{\mathrm{A}}^{(k)}\|_{\mathrm{F}} \leq C \|\mathrm{A} - \mathrm{A}^{(k)}\|_{\mathrm{F}} \qquad \text{where} \quad \widetilde{\mathrm{A}}^{(k)} = \sum_{j=1}^{k} \mathbf{u}_j \cdot \mathbf{v}_j^T \tag{7}$$

for some constant $C > 0$ independent of k. Moreover, which is probably the most interesting feature of this method, the cost of computing $\widetilde{\mathrm{A}}^{(k)}$ is $\mathcal{O}(kn)$. This cost is thus quasi-linear provided that the singular values decrease exponentially. Besides, the algorithm does not require to generate all the coefficients of the matrix, so that the cost of the storage is also $\mathcal{O}(kn)$. The detailed analysis of the ACA algorithm is out of the scope of this paper, we only report the algorithm itself (Algorithm 1) and refer the reader to [2, Chap.3] for further details. In our pseudo-code notation, for any vector $\mathbf{w} \in \mathbb{C}^n$, the number $\mathbf{w}(j)$ refers to the jth entry of $\mathbf{w}$; likewise for $\mathrm{A} \in \mathbb{C}^{n\times n}$, the number $\mathrm{A}(:,k)$ refers to the kth column, and $\mathrm{A}(j,:)$ refers to the jth row.

At the beginning of Algorithm 1, there is an initialization step for the choice of the index of the first j_*. For this initialization, one could take $j_* = 1$. Other choices may speed up the convergence of the algorithm (see [2, Section 3.4.3]). Algorithm 1 also involves a stopping criterion based on an error estimator. We took the *stopping criterion* given in [11]:

$$\begin{aligned} &|\mathbf{u}_r|_2 |\mathbf{v}_r|_2 \leq \varepsilon \|\textstyle\sum_{\ell=1}^{r} \mathbf{u}_\ell \cdot \mathbf{v}_\ell^T\|_{\mathrm{F}} \\ &\text{where} \quad \|\widetilde{\mathrm{A}}^{(r)}\|_{\mathrm{F}}^2 = \|\textstyle\sum_{\ell=1}^{r} \mathbf{u}_\ell \cdot \mathbf{v}_\ell^T\|_{\mathrm{F}}^2 = \sum_{j=1}^{r}\sum_{k=1}^{r} (\overline{\mathbf{u}}_j^T \mathbf{u}_k)(\overline{\mathbf{v}}_j^T \mathbf{v}_k). \end{aligned} \tag{8}$$

In this stopping criterion, the value $\varepsilon > 0$ is a fitting parameter whose choice depends on the degree of consistency that one wishes. The parameter controls the relative variation between two iterations since we have the following relation: $|\mathbf{u}_r|_2 |\mathbf{v}_r|_2 = \|\mathbf{u}_r \cdot \mathbf{v}_r^T\|_{\mathrm{F}} = \|\widetilde{\mathrm{A}}^{(r)} - \widetilde{\mathrm{A}}^{(r-1)}\|_{\mathrm{F}}$.

Algorithm 1 Partially Pivoted ACA

Initialize j_*
$r = 0$
while (stopping criterion not satisfied) **do**
  $\mathbf{w} = \mathrm{A}(j_*, :)^T - \sum_{\ell=1}^{r} \mathbf{u}_\ell(j_*)\, \mathbf{v}_\ell$
  $k_* = \mathrm{argmax}_{k=1\ldots n} |\mathbf{w}(k)|$
  $w_* = \mathbf{w}(k_*)$
  if $(w_* \neq 0)$ **then**
    $\mathbf{v}_{r+1} = \mathbf{w}$
    $\mathbf{w} = \mathrm{A}(:, k_*) - \sum_{\ell=1}^{r} \mathbf{v}_\ell(k_*)\, \mathbf{u}_\ell$
    $\mathbf{u}_{r+1} = w_*^{-1}\mathbf{w}$
    $j_* = \mathrm{argmax}_{j=1\ldots n} |\mathbf{w}(j)|$
    $r = r + 1$
  else
    terminate algorithm
  end if
end while

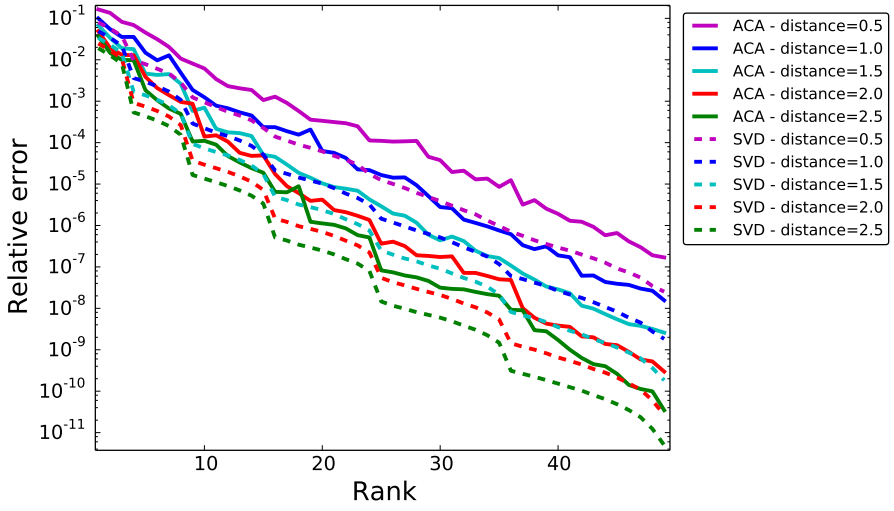


Figure 3: Relative error on the interaction matrix between two clusters of points with ACA (solid lines) and SVD (dashed lines) varying the distance between the two clusters.

Remark 2.2. To show the accuracy of ACA compared to a usual SVD, let us look at the interaction between two clusters of random points $X = \{\boldsymbol{x}_i\}$ and $Y = \{\boldsymbol{y}_i\}$ picked randomly according to a uniform law in two unit balls whose distance will vary. The interaction between two points $\boldsymbol{x}$ and $\boldsymbol{y}$ is given by $1/\|\boldsymbol{x} - \boldsymbol{y}\|$ so that the coefficients of the interaction matrix A are defined as

$$\mathrm{A}(i, j) = \frac{1}{4\pi|\boldsymbol{x}_i - \boldsymbol{y}_j|},$$

where $\boldsymbol{x}_i \in X$ and $\boldsymbol{y}_j \in Y$. In Figure 3, we show the relative error in Frobenius norm of $\mathrm{A}^{(k)}$ and $\widetilde{\mathrm{A}}^{(k)}$ with respect to A, where k is the rank. It can be observed that there is an exponential decrease in both cases, and this decrease is faster when the distance between the two clusters is greater, that is to say, when the interaction is more regular (see Section 2.2). Besides, we see that SVD gives a better approximation, which is expected since it can be proven that it gives the best approximation for a given rank.

2.2 Partition of the matrix in admissible blocks

Two important observations can be made for the matrix in (4) under consideration here. First, due to the singularity of the integral kernel, the singular values of this matrix do *not* decrease exponentially, i.e. the matrix is not admissible. As a consequence, the ACA compression strategy of Section 2.1 is not directly applicable. However, *sub-blocks* of the matrix A in (4) may be *admissible* (i.e. have exponentially decreasing singular values). To build an approximation of A allowing a substantial reduction of the cost of matrix-vector products, it would be sufficient to find a sparse matrix $\mathrm{A}_{\mathrm{nf}} \in \mathbb{C}^{3N\times 3N}$ (sometimes referred to as near field contribution) such that

$$\mathrm{A} = \mathrm{A}_{\mathrm{nf}} + \sum_{t\times s\in\mathscr{B}} \mathrm{A}_{t,s} \tag{9}$$

where $\mathscr{B}$ is a collection of subsets of $[\![1,3N]\!]\times[\![1,3N]\!]$, and each $\mathrm{A}_{t,s}$ is admissible. Here, for any subsets $t, s \subset [\![1,3N]\!]$, the submatrix $\mathrm{A}_{t,s}$ refers to a $3N\times 3N$ block with coefficients equal to zero, except for the sub-block obtained from A by restricting indices to the set $t\times s$.

With such a decomposition as (9), one may consider $\mathrm{A}' = \mathrm{A}_{\mathrm{nf}} + \sum_{t\times s\in\mathscr{B}} \mathrm{A}'_{t,s}$ as a good approximation for A, where each $\mathrm{A}'_{t,s}$ is obtained from $\mathrm{A}_{t,s}$ after application of the ACA compression procedure described in Section 2.1. Building the decomposition into sub-blocks efficiently can be achieved by using recursive algorithms described in the following. This constitutes the first step in building a *Hierarchical Matrix* (HM).

Remark that for the BEM matrices at hand (4), the Galerkin discretization establishes a correspondence between the numbering of unknowns and some *spatial distribution* of degrees of freedom. This gives an insight on how to find admissible (and thus compressible) sub-blocks inside the matrix A as described in the following.

For a ball $\mathrm{B}_s \subset \mathbb{R}^3$, define $s = \{j \in [\![1,3N]\!] \mid \mathrm{supp}(\boldsymbol{\psi}_j) \subset \mathrm{B}_s\}$. For another ball $\mathrm{B}_t \subset \mathbb{R}^3$, define $t \subset [\![1,3N]\!]$ accordingly. The more B_s and B_t are far from each other, the faster the singular values of $\mathrm{A}_{t,s}$ will decrease because the Green kernel, and so the interaction, will be more regular (see Remark 2.2 for an example). The distance between B_t and B_s that may be considered as sufficient for the admissibility of $\mathrm{A}_{t,s}$ depends on the radius of these balls. Such a criterion shall be referred later on as *admissibility criterion*. Current literature provides various admissibility criteria. It should be considered as problem dependent. In our case, we chose the following admissibility criterion (see [11, eq. 3.15])

$$\min\big(\mathrm{diam}(\mathrm{B}_t), \mathrm{diam}(\mathrm{B}_s)\big) < \eta\, \mathrm{dist}(\mathrm{B}_t, \mathrm{B}_s), \tag{10}$$

where

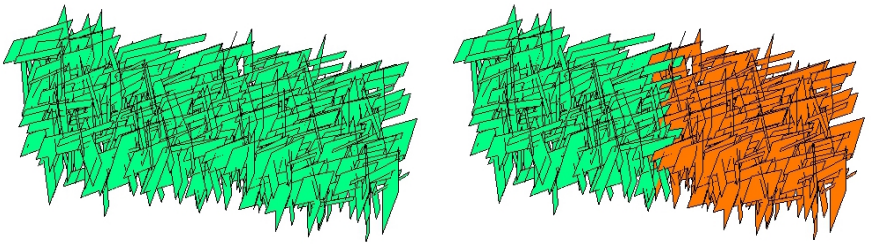
$$\mathrm{diam}(\mathrm{B_s}) = \max_{k_1,k_2\in s} |x_{k_1} - x_{k_2}|,$$
$$\mathrm{dist}(\mathrm{B_s}, \mathrm{B_t}) = \max_{k\in s, l\in t} |x_k - x_l|.$$

Here for $k \in s$, x_k is the barycenter of the mesh element corresponding to the Lagrange basis function $\boldsymbol{\psi}_j$, and $\eta > 0$ is a fitting parameter. For the present problem, the typical values of η that we considered range from $\eta = 0.1$ to $\eta = 10$. As suggested in [11] and in order to avoid the quadratic cost of the computation of $\mathrm{diam}(\mathrm{B_s})$, the practical implementation makes use of the more restrictive but more easily computable admissibility condition:

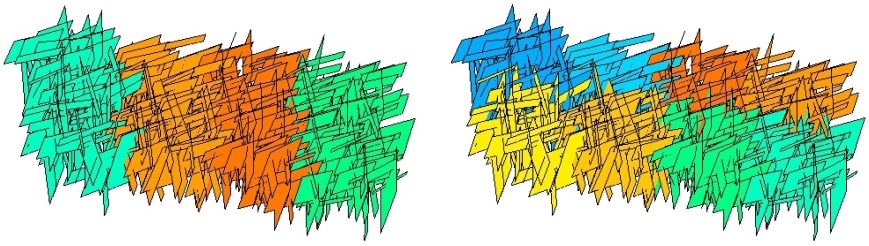
$$2\min\big(\max_{k\in s}|X_s - x_k|, \max_{l\in t}|X_t - x_l|\big) < \eta\,|X_t - X_s| - (\max_{k\in s}|X_s - x_k| + \max_{l\in t}|X_t - x_l|), \tag{11}$$

where X_s (resp. X_t) is the center of B_s (resp. B_t).ON A RAJOUTE LA DEFINITION DE DIAM ET DIST DU LIVRE Remark that relation (11) is close to (the inverse of) the IFPEN criterion (5) but takes advantage of the hierarchical structure of the cluster tree and keeps the symmetry.

Decomposition (9) involves a partition in admissible blocks, so we need to find them in an efficient way. To do so, we build a *cluster tree*, that is to say, we organize the set of

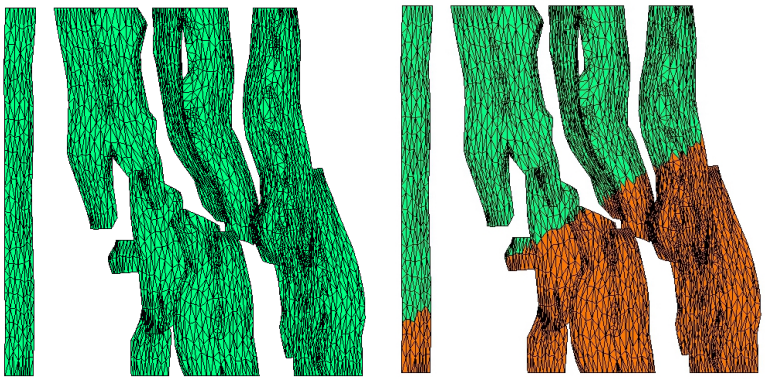


(a) Initial mesh. (b) Clusters at the first level.



(c) Clusters at the second level. (d) Clusters at the third level.

Figure 4: Cluster tree for a discrete fracture network.



(a) Initial mesh. (b) Clusters at the first level.

(c) Clusters at the second level. (d) Clusters at the third level.

Figure 5: Cluster tree for a fault network.

geometric elements as a binary tree such that each node of the tree is a cluster of geometric points.
The two sons of a cluster/node s are obtained by defining a separation hyperplane that goes through its barycenter X_s and is orthogonal to the direction of largest expanse of the cluster. This direction is obtained by computing the first eigenvector of the 3×3 covariance matrix C_s of the cluster:

$$C_s = \sum_{k \in s} (x_k - X_s)(x_k - X_s)^T. \tag{12}$$

The cluster is thus divided into two more ore less equal sons. This clustering algorithm can be found in [11]. Examples of the first four levels of such a cluster tree are given in Figures 4 and 5.

To build $\mathscr{B}$ as a collection of subsets of $[\![1, 3N]\!] \times [\![1, 3N]\!]$ corresponding to admissible blocks, we can look at pairs of clusters at the same level in the cluster tree and check if they are admissible according to criterion (10), starting from the root. If they are, we apply ACA to the corresponding sub-matrix, and if they are not, we look at the interactions between their sons. This recursive algorithm provides a block decomposition as in (9). Actually, if the block is admissible, we also check if the compression for the given ε is advantageous in terms of complexity: during the algorithm ACA, if $k(n+m)/(n \cdot m) \geq 1$ for a block $n \times m$ and k the current rank of the approximation, we stop and we do like if the block was not admissible because it is not worth compressing it.

3 Numerical results

3.1 Implementation

There already exist freely available libraries written in C or C++ that implement HM-ACA, see e.g. HLib (`http://hlib.org/`), H2Lib (`http://www.h2lib.org/`) or Ahmed (`https://github.com/xantares/ahmed`). Due to license restriction we redeveloped our own implementation of HM-ACA freely available on a GitHub repository at `https://github.com/xclaeys/ElastoPhi` and put under Lesser Gnu Public License (LGPL). Let us briefly comment on the most remarkable parts of the code and refer to its documentation for the details. The only external library we use is Eigen (`http://eigen.tuxfamily.org`), which is a Free Software.

The first important part is in the file `cluster.hpp` where the class `Cluster` is implemented. The constructor calls the function `build`, which recursively builds the cluster tree associated with a set of geometric points. More precisely, for a given cluster of points, it creates its two sons as described in the previous section (computing the center and the principal component) and calls the same function `build` on its two sons. Then the class `Block` contains a pair of clusters so that it is associated with their interaction. It has a function to check the admissibility of their interaction according to (10).

Then, in the file `lrmat.hpp` the class `LowRankMatrix` is implemented. Its constructor takes as input a submatrix and applies the ACA algorithm so that an instance of the class contains the collections of vectors defining the low rank approximation (7) of the submatrix.

Finally, we have all the tools to build the hierarchical matrix. The class `HMatrix`, implemented in the file `hmatrix.hpp`, contains two vectors of matrices, one for the low rank sub-matrices in $\mathrm{A}'_{t,s}$ and one for the dense sub-matrices in A_{nf}. Its constructor needs a set of geometric points so that it can build the associated cluster tree with the class `Cluster`. Then it recursively looks at blocks as described in the previous section using the class `Block` to check the admissibility. If it is admissible, it constructs a `LowRankMatrix` instance and adds it to its vector of low rank sub-matrices, otherwise it looks at the sub-blocks according to the cluster tree until it reaches the leaves (for the problem under consideration, they correspond to 3×3 sub-matrices) and stores them as dense sub-matrices.

With the headers contained in the folder `include`, we have already built some useful executables in the folder `src`. One of the main executables is `Compress`, which builds the hierarchical matrix with compressed and dense blocks for given parameters η (of the

admissibility test) and ε (of ACA compression), and then computes the compression rate, the relative error for a matrix-vector product and the relative error in Frobenius norm with respect to the dense matrix given in input. The executable `MultiCompression` and `CompaSparse` do the same but for various values of η and ε, and `CompaSparse` does it also for the IFPEN sparse matrix; they are the executables used to create the data postprocessed with the Python scripts of the folder `postprocessing` that generate the graphs of Section 3.2. Finally, there are some visualization executables. For example, the executable `VisuMesh` creates a file in Gmsh (`http://gmsh.info`) format to visualize the mesh of the network as in Figure 1, and the executable `VisuCluster` creates a file in Gmsh format to generate the images of Figures 4 and 5. Finally, the executable `VisuMatrix` creates the data postprocessed with the Python scripts to generate images as in Figure 6b.

3.2 Test cases and results

In this section we report a series of test cases to illustrate the performance of our code on different geometrical structures. Figures 6, 8, 9–11 refer to discrete networks of *fractures* (DFN), where each fracture is represented by a mesh element (a quadrangle); Figure 7 refers to a network of large *faults*, which have been triangulated and we consider each mesh triangle as a dislocation element. Both types of structure are considered in IFPEN applications.

For each geometrical structure we show first the corresponding mesh. Then, for the test case of smaller dimension (Figure 6), we visualize, for some pairs of η and ε, the local compression rate of each block of the HM-ACA matrix by coloring its entries using a color scale from 0 to 1. The local compression rate of an admissible block of dimension $n \times m$ with rank k in ACA approximation is $1 - k(n+m)/(n \cdot m)$ (the larger the compression rate, the more the matrix is compressed); the local compression rate of the non compressed blocks is set to 0. Note that a block is not necessarily a connected part of the matrix.

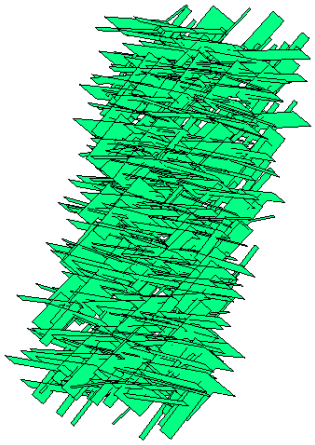
For all the cases, an error-compression graph summarizes the relevant results: for some pairs of values of the parameters η and ε, we report on the vertical axis the relative error in Frobenius norm of the corresponding HM-ACA matrix with respect to the dense matrix, and on the horizontal axis the achieved global compression rate. Next to each marker we indicate the value of ε ($\varepsilon = 1, 0.9, 0.5, 0.1, 0.01$) and the legend gives the value of η ($\eta = 10, 1$). The expression "0 blocks" means that the admissible blocks are approximated with zero rank matrices (i.e. zero matrices, but we just need to store $k = 0$) instead of computing their ACA approximation. Note that the 0 blocks strategy is close to the IFPEN Sparsification procedure, but with a different admissibility criterion. The global compression rate of a $3N \times 3N$ matrix A is defined as:

$$1 - \left(\sum_{t \times s \in \mathscr{B}} \frac{k(|t| + |s|)}{3N \cdot 3N} + \sum_{t \times s \in \mathscr{B}^c} \frac{|t||s|}{3N \cdot 3N} \right),$$

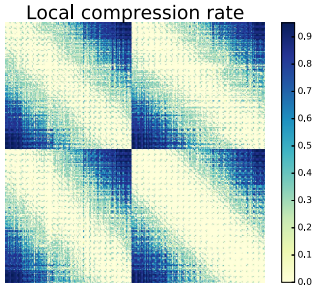
where $\mathscr{B}$ is the set of subsets of $[\![1, 3N]\!] \times [\![1, 3N]\!]$ corresponding to admissible (and then compressed) blocks introduced in Section 2.2.

Moreover, for the geometrical structures for which the sparse matrices obtained by IFPEN with the heuristic sparsification procedure are available, we report with a red triangular marker ("Sparsification" in the legend) the corresponding relative error in Frobenius norm and compression rate, given by $1 - n_{\bar{0}}/(3N \cdot 3N)$, where $n_{\bar{0}}$ is the number of non zero coefficients of the sparse matrix. Next to each Sparsification marker, the corresponding value for the sparsification parameter α ($\alpha = 2, 3, 4, 5, 6$) is given.

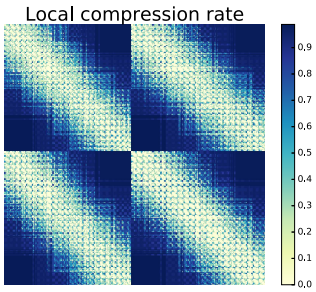
Looking at the first results in Figure 6, we remark that when ε increases, the relative error and the global compression rate increase as expected: recall that ε is used in the stopping criterion (8) for the ACA compression of each admissible block and the stopping criterion becomes less restrictive with bigger ε. For $\varepsilon \geq 1$ the ACA loop always stops after computing just 1 rank. Similarly, when η increases, the relative error and the global compression rate increase: indeed, η appears in the admissibility condition (10) and a bigger η allows



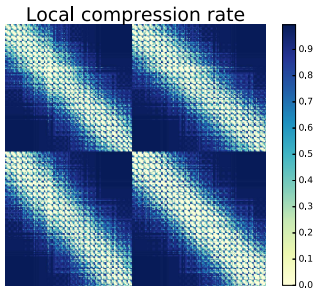
(a) Mesh.



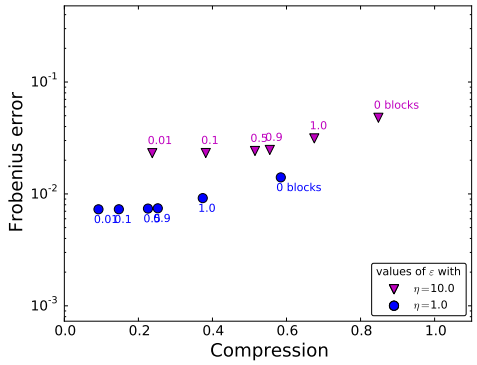
(b) Matrix compression for $\eta = 1, \varepsilon = 0.9$.



(c) Matrix compression for $\eta = 10, \varepsilon = 0.9$.



(d) Matrix compression for $\eta = 10, \varepsilon = 1$.

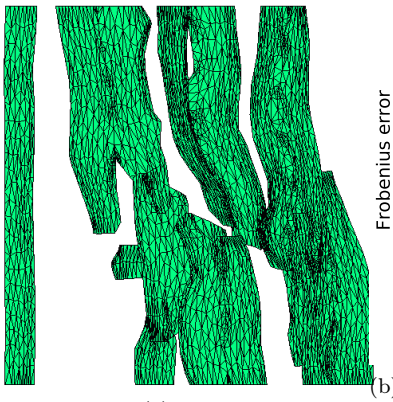


(e) Errors and compression rates for different values of η and ε.

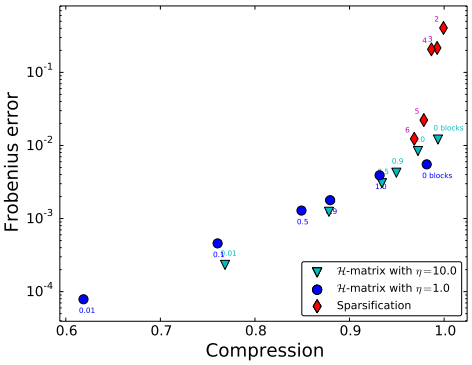
Figure 6: Results for the structure with 450 fractures.

the balls B_t and B_s to be farther while maintaining the admissibility of the corresponding block. These remarks indeed hold true for the other test cases as well.

Results for the network of large faults are shown in Figure 7, where we also give a comparison with the sparsification procedure of IFPEN. We can see that we obtain better results with HM-ACA: for instance, the most aggressive compression rate for HM-ACA is achieved with $\eta = 10$ and 0 blocks and gives an error of 0.012 for a compression rate of 99.3%, while the IFPEN heuristic procedure gives an error of 0.21 for a compression rate of 98.7%. The good behavior of HM-ACA can be explained by the regularity of the geometry, which translates into fewer near-field interactions. The situation is less clear when the geometry is less regular, as the following test cases for crack networks illustrate.

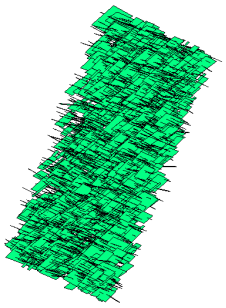


(a) Mesh.

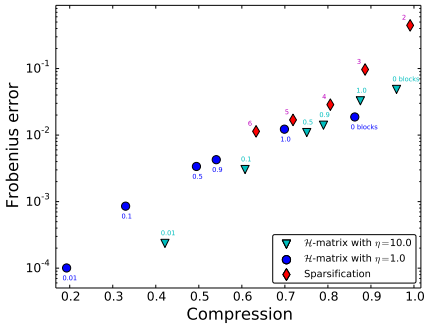


(b) Errors and compression rates for different values of η and ε.

Figure 7: Results for the structure of faults with 5364 mesh triangles.



(a) Mesh.



(b) Errors and compression rates for different values of η and ε.

Figure 8: Results for the structure with 1994 fractures.

The test case presented in Figure 8 consists in a crack network of 1994 fractures. Here, we still obtain better results with HM-ACA: with $\eta = 10$ and $\varepsilon = 0.9$, we have an error of 0.011 for a compression rate of 75%, while the IFPEN sparsification procedure achieves only a compression rate of 63.3% for the same error.

Figures 9,10 and 11 correspond to another test case with 3600 fractures of increasing density, as the considered volume goes from $V_a = 900 \times 300 \times 20$ in Figure 9 to $V_c = 300 \times 300 \times 20$ in Figure 11. First, we can remark that when the density increases, it becomes more difficult to obtain a good compression rate, regardless of the compression algorithm. For instance, for $\eta = 1$, $\varepsilon = 0.9$, the compression rate for V_1 (respectively V_3) is 76.2% (respectively 94.9%) and the relative error is 0.0072 (respectively 0.0037). Then, we see that for the lower density cases (Figures 9 and 10), we obtain little to no improvement using HM-ACA compared to the IFPEN sparsification heuristic. However, for the denser case (Figure 11) the use of HM-ACA can be justified: for instance, $\eta = 10$ and $\varepsilon = 0.01$ gives an error of 0.0059 for a compression rate of 78.5%, whereas the error obtained with the sparsification heuristic of IFPEN is 0.010 for a similar compression rate of 78.3%.

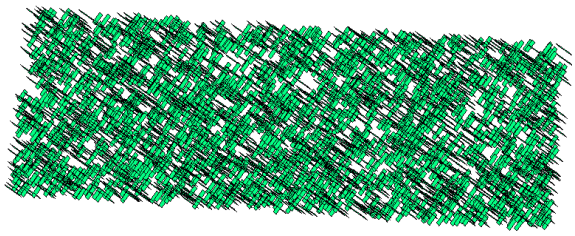
To conclude, even though the sparsification procedure of IFPEN can be implemented more easily and gives decent results, the HM-ACA strategy is the better approach for faults geometries, presumably because of their smoothness. However, in the situation of crack networks, where the geometry is less regular, HM-ACA gives mixed results: we observe improvements in the densest test case compared to the IFPEN strategy, but little to no gain in the less dense geometries. Overall, HM-ACA offers greater flexibility in the compression-accuracy trade-off, although at the cost of more involved calibration of the parameters.

4 Conclusion

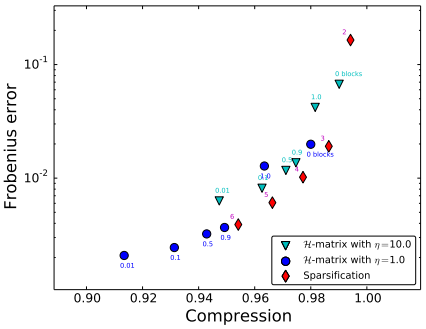
We have implemented a HM-ACA code and tested it on several matrices provided by IFPEN to study the efficiency of this kind of method for particularly complex geometries as in Figure 1. From our numerical results, HM-ACA proves to be a more accurate, or at least more flexible alternative to the heuristic sparsification procedure developed by IFPEN. It works particularly well for geometries coming from faults, where far interactions are predominant. With geometries coming from discrete fracture networks, lower compression rates are obtained, especially for dense networks. This is quite intuitive, since then there are less far interactions to be compressed.

Our implementation is clearly not optimal, and there is room for improvement. The main perspectives in this respect concerns the parallelization of the matrix-vector product in conjunction with a hierarchical matrix strategy (see [2, Section 2.3]) and the parallelization during the construction of the H-matrix format (see [2, Section 3.4.6]).

At a more theoretical level, another admissibility criterion more suited to fracture networks could be designed, for instance taking into account the anisotropy of fractures using ellipsoids instead of balls. However, it may be necessary to carry out a mathematical analysis of this problem to find the most appropriate geometrical criterion. Finally, looking for alternative acceleration strategies would also be an interesting research direction. Due to the low level of accuracy required for applications considered at IFPEN, and since the geometries under consideration are particularly rough (crack networks), probabilistic acceleration techniques might prove even better suited.

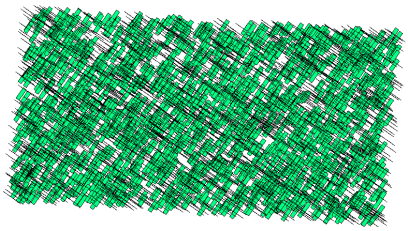


(a) Mesh.

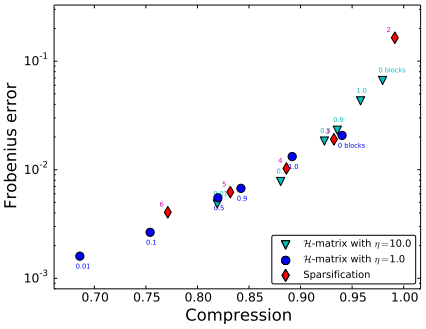


(b) Errors and compression rates.

Figure 9: Structure with 3600 fractures inside a volume $V_a = 900 \times 300 \times 20$.

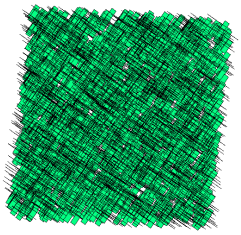


(a) Mesh.

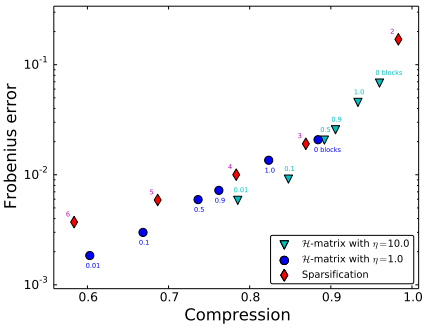


(b) Errors and compression rates.

Figure 10: Structure with 3600 fractures inside a volume $V_b = 600 \times 300 \times 20$.



(a) Mesh.



(b) Errors and compression rates.

Figure 11: Structure with 3600 fractures inside a volume $V_c = 300 \times 300 \times 20$.

References

[1] J. Barnes and P. Hut. A hierarchical $\mathcal{O}(N \log N)$ force-calculation algorithm. *Nature*, 324:446–449, 1986.

[2] M. Bebendorf. *Hierarchical matrices: A Means to Efficiently Solve Elliptic Boundary Value Problems*, volume 63 of *Lecture Notes in Computational Science and Engineering*. Springer-Verlag, Berlin, 2008.

[3] J.A. Board, J.W. Causey, J.F. Leathrum, A. Windemuth, and K. Schulten. Accelerated molecular dynamics simulation with the parallel fast multipole algorithm. *Chemical Physics Letters*, 198(1):89 – 94, 1992.

[4] T. A. Cruse. *Boundary element analysis in computational fracture mechanics*, volume 1 of *Mechanics: Computational Mechanics*. Kluwer Academic Publishers Group, Dordrecht, 1988.

[5] C. P. Davis and W. C. Chew. Frequency-independent scattering from a flat strip with tez-polarized fields. *IEEE Transactions on Antennas and Propagation*, 56(4):1008–1016, April 2008.

[6] L. Greengard. Fast algorithms for classical physics. *Science*, 265:909–914, 1994.

[7] W. Hackbusch. *Hierarchical Matrices: Algorithms and Analysis*, volume 49 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, 2016.

[8] J.P. Hirth and J. Lothe. *Theory of dislocations*. John Wiley & Sons, 1982.

[9] K.M. Li and H.Y. Wong. A review of commonly used analytical and empirical formulae for predicting sound diffracted by a thin screen. *Applied Acoustics*, 66(1):45–76, 2005.

[10] B. Lizé. *Fast direct solver for the boundary element method in electromagnetism and acoustics : H-Matrices. Parallelism and industrial applications*. Theses, Université Paris-Nord - Paris XIII, June 2014.

[11] S. Rjasanow and O. Steinbach. *The fast solution of boundary integral equations*. Mathematical and Analytical Techniques with Applications to Engineering. Springer, New York, 2007.

[12] V. Rokhlin. Rapid solution of integral equations of classical potential theory. *J. Comput. Phys.*, 60(2):187–207, 1985.

[13] O. Steinbach. *Numerical approximation methods for elliptic boundary value problems*. Springer, New York, 2008. Finite and boundary elements, Translated from the 2003 German original.

[14] I. Tolstoy. Exact, explicit solutions for diffraction by hard sound barriers and seamounts. *The Journal of the Acoustical Society of America*, 85(2):661–669, 1989.

[15] S. N. Vorobyov and L. M. Lytvynenko. Electromagnetic wave diffraction by semi-infinite strip grating. *IEEE Transactions on Antennas and Propagation*, 59(6):2169–2177, June 2011.