FEM and BEM simulations with the Gypsilab framework

François Alouges*and Matthieu Aussal†

Abstract

Gypsilab is a MATLAB toolbox which aims at simplifying the development of numerical methods that apply to the resolution of problems in multiphysics, in particular, those involving FEM or BEM simulations. The specifities of the toolbox, in particular its ease of use, are shown together with the methodology we have followed for its development. Example codes that are short though representative enough are given both for FEM and BEM applications. A performance comparison with FreeFem++ is also provided.

1 Introduction

The finite element method (FEM) is nowadays extremely developed and has been widely used in the numerical simulation of problems in continuum mechanics, both for linear and non-linear problems. Many softwares exist among which we may quote free ones (e.g. FREEFEM++ [8], FENICS [12], FireDrake [13], Xlife++ [11], Feel++ [9], GetDP [14, 15], etc.) or commercial ones (e.g. COMSOL [16]). The preceding list is far from being exhaustive as the method has known many developments and improvements and is still under active study and use. Numerically speaking, and without entering into too much details, let us say that the peculiarity of the method is that it is based on a variational formulation that leads to sparse matrices the size of which is typically proportional to the number of unknowns. Direct or iterative methods can then be used to solve the underlying linear systems.

On the other hand, the boundary element method (BEM) is used for problems which can be expressed using a Green kernel. A priori restricted to linear problems, the method inherently possesses the faculty of handling free space solutions and is therefore currently used for solving Laplace equations, wave scattering (in acoustics, electromagnetics or elastodynamics), or Stokes equations for instance. Although it leads to dense matrices, which storage is proportional to the square of the number of unknowns, the formulation is usually

*CMAP - Ecole Polytechnique, Université Paris-Saclay, Route de Saclay, 91128, Palaiseau Cedex, France. francois.alouges@polytechnique.edu .

†CMAP - Ecole Polytechnique, Université Paris-Saclay, Route de Saclay, 91128, Palaiseau Cedex, France. matthieu.aussal@polytechnique.edu .

restricted to the boundary of the domain under consideration (e.g. the scatterer), which lowers the dimension of the object that needs to be discretized. Nevertheless, due to computer limitations, those methods may require a special compression technique such as the Fast Multipole Method (FMM) [5, 7], the $\mathcal{H}$-matrix storage [6] or the more recent Sparse Cardinal Sine Decomposition [1, 2, 3], in order to be applied to problems of sizes relevant for an accurate simulation. In terms of softwares available for the simulation with this kind of methods, and probably due to the technicality sketched above, the list is much shorter than before. In the field of academic, freely available softwares, one can quote BEM++ [10], or Xlife++ [11]. Commercial softwares using the method are for instance the ones distributed by ESI Group (VAone for acoustics [25] and E-Field [26] for electromagnetism), or Aseris [27]. Again, the preceding list is certainly not exhaustive.

Eventually, one can couple both methods, in particular to simulate multiphysics problems that involve different materials and for which none of the two methods apply directly. This increases again the complexity of the methodology as the user needs to solve coupled equations that are piecewise composed of matrices sparsely stored combined with other terms that contain dense operators or compressed ones. How to express such a problem? Which format should be used for the matrix storage? Which solver applies to such cases, an iterative or a direct one? Eventually, the writing of the software still requires abilities that might be out of his/her field of expertise.

As a sake of example, it is very important to notice that, to the knowledge of the authors, the only freely available software, which uses the BEM, and for which the way of programming is comparable to FreeFem++, is BEM++ that has been interfaced into FENICS.

The preceding considerations have led us to develop the toolbox openFEM which aims at simplifying and generalizing the development of FEM-BEM coupling simulation algorithms and methods. Written as a full Matlab suite of routines, the toolbox can be used to describe and solve various problems that involve FEM and/or BEM techniques. We have also tried to make the finite element programming as simple as possible, using a syntax comparable to the one used in FreeFem++ or FENICS/FireDrake and very close to the mathematical formulation itself which is used to discretize the problems. The toolbox is a part of a more general environment, called Gypsilab, that contains several toolboxes:

- openMSH for the handling of meshes;
- openDOM for the manipulation of quadrature formulas;
- openHMX that contains the full $\mathcal{H}$-matrix algebra [6], including the LU factorization and inversion;
- openFEM for the finite element and boundary element methods.

Other toolboxes are also developed inside Gypsilab that we plan to describe elsewhere.

Eventually, although the main goal is not the performance, the toolbox openFEM may handle, on classical computers, problems whose size reaches a few millions of unknowns for the FEM part and a few hundreds of thousands of unknowns for the BEM when one uses compression. For FEM applications, this performance is very much comparable to already proposed Matlab strategies [22, 23, 24], though with much higher generality and flexibility.

The present paper explains in some details the capabilities of the toolbox openFEM together with its use. Explanations concerning the implementation are also provided that give insight of the genericity of the approach. In order to simplify the exposition, we focus here on FEM or BEM applications, leaving coupled problems to a forthcoming paper.

Due to its simplicity of use, we strongly believe that the software described here could become a reference in the field. Indeed, applied mathematicians, interested in developing new FEM-BEM coupled algorithms need such tools in order to address problems of significant sizes for real applications. Moreover, the software is also ideal for the prototyping of academic or industrial applications.

2 Simple examples

We show in this Section a series of small example problems and corresponding Matlab listings.

2.1 A Laplace problem with Neumann boundary conditions

Let us start with the writing of a finite element program that solves the partial differential equation (PDE)

$$\left\{ \begin{array}{l} -\Delta u + u = f \text{ on } \Omega\,, \\ \dfrac{\partial u}{\partial n} = 0 \text{ on } \partial\Omega\,, \end{array} \right. \tag{1}$$

where the right-hand side function f belongs to $L^2(\Omega)$. Here Ω stands for a domain in $\mathbb{R}^2$ or $\mathbb{R}^3$.

The variational formulation of this problem is very classical and reads:

Find $u \in H^1(\Omega)$ such that $\forall v \in H^1(\Omega)$

$$\int_\Omega \nabla v(x) \cdot \nabla u(x)\, dx + \int_\Omega v(x) u(x)\, dx = \int_\Omega f(x) v(x)\, dx\,.$$

The finite element discretization is also straightforward and requires solving the same variational formulation where $H^1(\Omega)$ is replaced by one of its finite dimensional subspaces (for instance the set of continuous and piecewise affine on a triangular mesh of Ω in the case of linear P^1 elements).

We give hereafter the openFEM source code used to solve such a problem in the case where the domain under consideration is the unit disk in $\mathbb{R}^2$, and the function f is given by $f(x, y, z) = x^2$.

```
1  % Library paths
2  addpath('../openDom')
3  addpath('../openMsh')
4  addpath('../openFem')
5  % Mesh of the disk
6  N = 1000;
7  mesh = mshDisk(N,1);
8  % Integration domain
9  Omega = dom(mesh,3);
10 % Finite elements
11 Vh = fem(mesh, 'P1');
12 % Matrix and RHS
13 f = @(X) X(:,1).^2;
14 K = integral(Omega,grad(Vh),grad(Vh)) + integral(Omega,Vh,Vh);
15 F = integral(Omega, Vh, f);
16 % Solving
17 uh = K \ F;
18 figure
19 graph(Vh,uh);
```

We believe that the listing is very clear and almost self explanatory. Besides initialization (lines 2-4), one recognizes the meshing part in which a disk is meshed with 1000 vertices (lines 6-7), the definition of an integration domain (line 9), the finite element space (line 11), the variational formulation of the problem (lines 13-15) and the resolution (lines 17). Let us immediately insist on the fact that the operators constructed by the `integral` keyword are really matrices and vectors Matlab objects so that one can use classical Matlab functionalities for the resolution (here the backslash operator \). Ploting the solution (lines 20-21) leads to the figure reported in Fig. 1.

2.2 Fourier boundary conditions

We now consider the problem

$$\begin{cases} -\Delta u + u = 0 \text{ on } \Omega\,, \\ \dfrac{\partial u}{\partial n} + u = g \text{ on } \partial\Omega\,, \end{cases} \tag{2}$$

Again, the variational formulation of the problem is standard and reads as follows

Find $u \in H^1(\Omega)$ such that $\forall v \in H^1(\Omega)$

$$\int_\Omega \nabla v(x) \cdot \nabla u(x)\, dx + \int_\Omega v(x)u(x)\, dx + \int_{\partial\Omega} v(s)u(s)\, ds = \int_{\partial\Omega} g(s)v(s)\, ds\,.$$

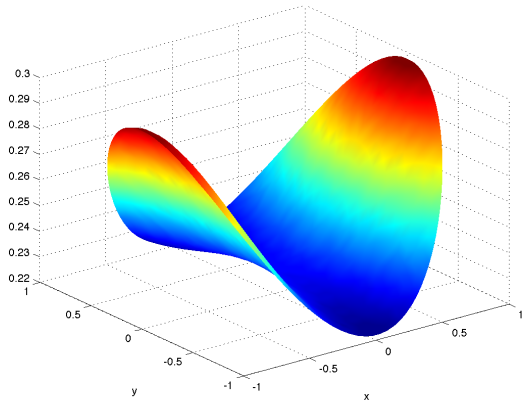

Figure 1: Numerical solution of (1) on a unit disk using GYPSILAB.

The preceding code is modified in the following way (we have taken the example where $g(s) = 1$).

```
1  % Library paths
2  addpath('../openDom')
3  addpath('../openMsh')
4  addpath('../openFem')
5  % Create mesh disk + boundary
6  N = 1000;
7  mesh = mshDisk(N,1);
8  meshb = mesh.bnd;
9  % Integration domains
10 Omega = dom(mesh,7);
11 Sigma = dom(meshb,3);
12 % Finite element space
13 Vh = fem(mesh,'P2');
14 % Matrix and RHS
15 K = integral(Omega, grad(Vh), grad(Vh)) ...
16       + integral(Omega, Vh, Vh) ...
17       + integral(Sigma, Vh, Vh);
18 g = @(x) ones(size(x,1),1);
19 F = integral(Sigma, Vh, g);
20 % Resolution
21 uh = K \ F;
22 figure
23 graph(Vh,uh);
```

Compared to the preceding example, a boundary mesh and an associated integration domain are also defined (lines 8 and 11). Let us note the piecewise quadratic (so-called P^2) element used (line 13) which leads to use more accurate

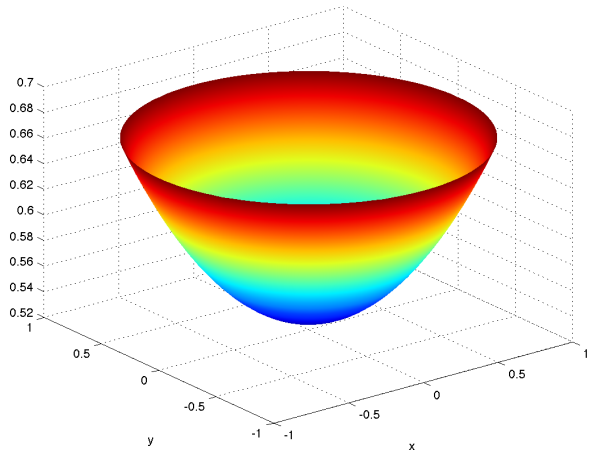

Figure 2: Numerical solution of (2) on a unit disk using GYPSILAB.

integration formulas, respectively with 7 points for the triangles (line 10) and 3 points per segment for the boundary mesh (line 11). Again, the result obtained is plotted in Fig. 2.

2.3 An eigenvalue problem

We end up this section by showing an example of a 3D problem, namely, the computation of the first eigenvalue/eigenvectors of the Laplace operator in the parallelepipedic volume $[0,1]\times[0,\frac{1}{2}]\times[0,\frac{1}{2}]$, with Dirichlet boundary conditions. Mathematically speaking the problem writes as finding couples (λ, u) solution to the eigenvalue problem

$$\left\{ \begin{array}{l} -\Delta u = \lambda u \text{ on } \Omega\,, \\ u = 0 \text{ on } \partial\Omega\,, \end{array} \right. \tag{3}$$

where $\Omega = [0,1] \times [0,\frac{1}{2}] \times [0,\frac{1}{2}]$. Now the problem is posed in 3D and we need to force (homogeneous) Dirichlet boundary conditions. The corresponding MATLAB listing becomes

```
1  % Library paths
2  addpath('../openDom')
3  addpath('../openMsh')
4  addpath('../openFem')
5  % Create mesh of the cube + boundary
6  N = 1e4;
7  mesh = mshCube(N,[1  0.5  0.5]);
8  meshb = mesh.bnd;
```

```
9  % Integration domains
10 Omega = dom(mesh,4);
11 % Finite element space
12 Vh = fem(mesh,'P1');
13 Vh = dirichlet(Vh,meshb);
14 % Matrix and RHS
15 K = integral(Omega, grad(Vh), grad(Vh)) ;
16 M = integral(Omega, Vh, Vh);
17 % Resolution
18 Neig = 10;
19 [V,EV] = eigs(K,M,Neig,'SM');
```

Notice the enforcement of Dirichlet boundary conditions on the finite element space (line 13), the assembling of the rigidity and mass matrices (lines 15-16). The results obtained by the preceding code for the computatoin of the first 10 eigenelements of the Laplace operator with Dirichlet boundary condition is given in Table 1.

Number	Exact	Numeric	Relative error
1	88.8264	90.1853	0.0153
2	118.4353	120.5672	0.0180
3	167.7833	171.4698	0.0220
4	207.2617	213.3081	0.0292
5	207.2617	213.4545	0.0299
6	236.8705	243.2719	0.0270
7	236.8705	244.4577	0.0320
8	236.8705	245.0540	0.0345
9	286.2185	296.6374	0.0364
10	286.2185	297.9819	0.0411

Table 1: Exact and approximated eigenvalues of the Laplacian with Dircihlet boundary conditions on the parallelepipedic domain $[0,1] \times [0,\frac{1}{2}] \times [0,\frac{1}{2}]$. For each of the first ten eigenvalue, we give its exact value, the one computed with the program before and the relative error.

3 Finite element programming in MATLAB

Programming the finite element method in MATLAB is very attractive and has already been considered by many people (see for instance [17, 18, 19, 20, 22, 23, 24]), probably because the language is easy to use and already contains the most recent linear solvers. It is important to notice that the language is usually very powerful when one uses its vectorial capabilities. Therefore, the traditional assembling of the matrices coming from finite element discretizations being often written using a loop over the elements with indirect adressing, might lead

to prohibitive execution times in MATLAB. This problem was identified a long time ago and several ways have already been proposed to circumvent it. In particular, in [22], are given and compared different alternatives that lead to very efficient assembling. Many languages are also compared (C++, MATLAB, PYTHON, FREEFEM++), and it is shown that the C++ implementation only brings a slight improvement in performance. Other MATLAB implementations are proposed in the literature (see e.g. [17, 19, 20, 23, 24]), but they all suffer from the lack of generality. The problem solved is indeed very often the Laplacian with piecewise linear finite elements and one needs to adapt the approach for any different problem.

We have followed yet another strategy that has the great advantage to be very general and easily adaptable to a wide variety of possible operators to build and solve, and which also enables the user to assemble matrices that come from the coupling between different finite elements. Moreover, we will see that the method also leads to reasonably good assembling times. To this aim, we give the following example from which one can understand the generality of the approach and the way OPENFEM is coded.

Let us consider the case of assembling the mass matrix. To be more precise, we call $\mathcal{T}$ a conformal triangulation[1] on which one has to compute the matrix A whose entries are given by

$$A_{ij} = \int_{\mathcal{T}} \phi_i(x)\phi_j(x)\,dx\,. \tag{4}$$

Here we have used the notation $(\phi_i)_{1\leq i\leq N}$ to denote the basis functions of the finite element (discrete) space of dimension N. To gain generality, the preceding integral is usually not computed exactly, but rather approximated using a quadrature rule. Thus, calling $(x_k,\omega_k)_{1\leq k\leq M}$ the set of all quadrature points over $\mathcal{T}$, we write

$$A_{ij} \sim \sum_{k=1}^{M} \omega_k \phi_i(x_k)\phi_j(x_k)\,. \tag{5}$$

Introducing now the two matrices W and C (respectively of size $M\times M$ and $M\times N$) defined by

$$W_{kk} = \omega_k \text{ for } 1\leq k\leq M\,, \text{ and } B_{kj} = \phi_j(x_k) \text{ for } 1\leq k\leq M,\ 1\leq j\leq N\,, \tag{6}$$

we may rewrite (5) as

$$A \sim B^t\, W\, B\,, \tag{7}$$

the approximation coming from the fact that a quadrature rule has been used instead of an exact formula. In particular, we notice that if the quadrature formula is exact in (5), then the approximation is in fact an equality.

From the preceding considerations the procedure that enables to assemble the sparse mass matrix can be summarized as:

[1] Triangulation usually means a 2D problem, while we would have to consider a tetrahedral mesh for 3D problems. This is not a restriction as we can see.

- Knowing the triangulation (resp. tetrahedral mesh), and a quadrature formula on a reference triangle (resp. tetrahedron), build the set of quadrature points/weights $(x_k, \omega_k)_{1 \leq k \leq M}$.
- Knowing the finite element used (or, equivalently, the basis functions $(\phi_j)_{1 \leq j \leq N}$) and the quadrature points $(x_k)_{1 \leq k \leq M}$, build the matrices W and B.
- Eventually, compute $A = B^t \, W \, B$.

Notice that the matrices W and B are usually sparse. Indeed, W is actually diagonal, while B has a non zero entry B_{jk} only for the quadrature points x_k that belong to the support of ϕ_j. In terms of practical implementation, the matrix W is assembled using a `spdiags` command while the matrix B is built using a vectorized technique.

The preceding procedure is very general and does only rely on the chosen finite element or the quadrature formula. Moreover, the case of more complicated operators can also be treated with only slight modifications. Indeed, if one considers the case of the Laplace operator, for which the so-called rigidity matrix is given by

$$K_{ij} = \int_{\mathcal{T}} \nabla \phi_i(x) \cdot \nabla \phi_j(x) \, dx \,, \tag{8}$$

one may write similarly

$$K_{ij} \sim \sum_{k=1}^{M} \omega_k \nabla \phi_i(x_k) \cdot \nabla \phi_j(x_k) \,, \tag{9}$$

from which one deduces

$$K \sim C_x^t \, W \, C_x + C_y^t \, W \, C_y + C_z^t \, W \, C_z \,,$$

where the matrix W is the same than before and the matrices C_x, C_y and C_z are given for $1 \leq k \leq M$, $1 \leq j \leq N$ by

$$C_{x,kj} = \frac{\partial \phi_j}{\partial x}(x_k) \,, \; C_{y,kj} = \frac{\partial \phi_j}{\partial y}(x_k) \,, \; C_{z,kj} = \frac{\partial \phi_j}{\partial z}(x_k) \,. \tag{10}$$

In openFEM all those matrices are built "on the fly" upon request of the user of the desired final matrix. This might slow down a bit the method to the benefit of a very little footprint in memory and much higher genericity.

4 Quick overview of openFEM

This section is not intended to be a user's manual. We just give the main functionalities of openFEM and refer the interested reader to the website [28]. openFEM follows the programming principles of the whole environment Gypsilab which tries to compute as much as possible the quantities "on the fly",

or in other words, to keep in memory as little information as possible. The main underlying idea is that storing many matrices (even sparse) might become memory consuming, and recomputing on demand the corresponding quantities does not turn to be the most costly part in usual computations. Having this idea in mind helps in understanding all the "philosophy" that we have followed for the development of the different toolboxes. Moreover, the whole library is object oriented and the toolboxes have been implemented as value classes.

4.1 The mesh and the integration domain

openFEM is built in strong interaction with the toolboxes openMSH and openDOM which are respectively devoted to handle meshes ans quadrature formula. Namely, we distinguish between two geometrical objects:

- The **mesh**. It is a purely geometric object with which one can compute only geometric quantities (e.g. normals, volumes, edges, faces, etc.). A mesh can be of dimension 1 (a curve), 2 (a surface) or 3 (a volume) but is always embedded in the dimension 3 space and is a simplicial mesh (i.e. composed of segments, triangles or tetrahedra). It is defined by three tables :
  - A list of vertices, which is a table of size $N_v \times 3$ containing the three dimensional coordinates of the vertices ;
  - A list of elements, which is a table of size $N_e \times (d+1)$, d being the dimension of the mesh and N_e the number of elements ;
  - A list of colours, which is a column vector of size $N_e \times 1$ defining a colour for each element of the mesh, this last table being optional.

  A typical msh object is given by the following structure.

```
>>mesh

mesh =

  2050x3 msh array with properties:

    vtx: [1083x3 double]
    elt: [2050x3 double]
    col: [2050x1 double]
```

  The openMSH toolbox does not yet contain a general mesh generator per se. Only simple objects (cube, square, disk, sphere, etc.) can be meshed. More general object may be nevertheless loaded using classical formats (`.ply`, `.msh`, `.vtk`, `.stl`, etc.). Since the expected structure for a mesh is very simple, the user may also his/her own wrapper to create the above tables.

Any operation on meshes (e.g. intersection or union of different meshes, extraction of the boundary, etc.) is coded inside the openMSH toolbox. Let us emphasize that upon loading, meshes are automatically cleaned by removing unnecessary or redondant vertices or elements.

- The **domain** which is a geometric object on which one can furthermore integrate. Numerically speaking, this is the concatenation of a mesh and a quadrature formula, identified by a number, that one uses with the corresponding simplices of the mesh, in order to integrate functions. The default choice is a quadrature formula with only one integration point located at the center of mass of the simplices.This is usually very inaccurate, and this is almost always mandatory to enhance the integration by taking a higher degree quadrature formula. A domain is defined using the `dom` keyword. For instance, the command

```
1 Omega = dom(mesh,4);
```

defines an integration domain `Omega` from the mesh, using an integration formula with 4 integration points. If such an integration formula is not available, the program returns an error. Otherwise, the command creates an integration domain with the structure shown by the following output.

```
>> omega = dom(mesh,4)

omega =

  dom with properties:

    gss: 4
    msh: [2050x3 msh]
```

We believe that making the construction of the quadrature formula very explicit helps the user to pay attention to this very important point, and make the right choice for his/her application. In particular, for finite element computing, the right quadrature formula that one needs to use depends on the order of the chosen finite element. Integration functionalities are implemented in the openDOM toolbox (see below).

4.2 The Finite Element toolbox (openFEM)

Finite element spaces are defined through the use of the class constructor `fem`. Namely, the command

```
1 Vh = fem(mesh, name);
```

creates a finite element space on the `mesh` (an arbitrary 2D or 3D mesh defined in $\mathbb{R}^3$) of type defined by `name`. At the moment of the writing of this paper 3 different families of finite elements are available:

- The Lagrange finite elements. The orders 0, 1 and 2 are only available for the moment. They corresponds to piecewise polynomials of degree 0, 1 and 2 respectively. They are available in 2D or 3D.

- The edge Nédélec finite element. It is a space of vectorial functions whose degrees of freedom are the circulation along all the edges of the underlying mesh. This finite element is defined in both 2D and 3D. In 2D is implemented a general form for which the underlying surface does not need to be flat.

- The Raviart-Thomas, also called Rao-Wilton-Glisson (RWG) finite elements. Also vectorial, the degrees of freedom are the fluxes through the edges (in 2D) or the faces (in 3D) of the mesh. Again, the 2D implementation is available for general non-flat surfaces.

For the two last families, only the lowest orders are available. The value of the variable `name` should be one of `'P0'`, `'P1'`, `'P2'`, `'NED'`, `'RWG'` respectively, depending on the desired finite element in the preceding list.

Besides the definition of the finite element spaces, the toolbox OPENFEM contains a few more functionalities, as the following.

- Operators. Operators can be specified on the finite element space itself. Indeed, we have already seen the example

```
1 A = integral(dom, grad(Vh), grad(Wh));
```

  which returns the matrix of the Laplacian. Available operators are:

  - `grad, div, curl`, which are differential operators that act on scalar or vectorial finite elements.
  - `curl, div, nxgrad, divnx, ntimes`. Those operators are defined when solving problems on a bidimensional surface in $\mathbb{R}^3$. Here `n` stands for the (outer) normal to the surface and all the differential operators are surfacic. Such operators are commonly used when solving problems with the BEM (see below).

- Plots. Basic functions to plot a finite element or a solution are available. Namely, we have introduced

```
1 plot(Vh);
```

  where `Vh` is a finite element space to plot the degrees of freedom the define its functions, and

```
1 surf(Vh,u);
```

  in order to plot a solution. In that case the figure produced consists in the geometry on which the finite element is defined coloured by the magnitude of the `u`. Eventually, as we have seen in the first examples presented in this paper, the command `graph` plots the graph of a finite element computed on a 2D flat surface.

```
graph(Vh,u);
```

4.3 The `integral` keyword (openDOM)

Every integration done on a domain is evaluated through the keyword `integral`. Depending on the context explained below, the returned value can be either a number, a vector or a matrix. More precisely, among the possible syntaxes are

- `I = integral(dom, f);`
  where `dom` is an integration domain on which the integral of the function `f` needs to be computed. In that case the function `f` should be defined in a vectorial way, depending on a variable `X` which can be a $N \times 3$ matrix. For instance the definitions

```
f = @(X) X(:,1).*X(:,2);
g = @(X) X(:,1).^2 + X(:,2).^2 + X(:,3).^2;
h = @(X) X(:,1) + X(:,2).*X(:,3);
```

  respectively stand for the (3 dimensional) functions

$$f(x,y,z) = xy\,,\ \ g(x,y,z) = x^2+y^2+z^2\,,\ \ h(x,y,z) = x + yz\,.$$

  Since domains are all 3 dimensional (or more precisely embedded in the 3 dimensional space), only functions of 3 variables are allowed.

- `I = integral(dom, f, Vh);`
  In that case, `f` is still a 3 dimensional function as before while `Vh` stands for a finite element space. The returned value `I` is a column vector whose entries are given by

$$I_i = \int_{dom} f(X)\phi_i(X)\,dX$$

  for all basis function ϕ_i of the finite element space.

- `I = integral(dom, Vh, f);`
  This case is identical to the previous one but now, the returned vector is a row vector.

- `A = integral(dom, Vh, Wh);`
  where both `Vh` and `Wh` are finite element spaces. This returns the matrix A defined by

$$A_{ij} = \int_{dom} \phi_i(X)\psi_j(X)\,dX$$

  where ϕ_i (resp. ψ_j) stands for the current basis function of `Vh` (resp. `Wh`).

- `A = integral(dom, Vh, f, Wh);`
  This is a simple variant where the entries of the matrix A are now given by

$$A_{ij} = \int_{dom} f(X)\phi_i(X)\psi_j(X)\,dX\,.$$

As a matter of fact, the leftmost occuring finite element space is assumed to correspond to test functions while the rightmost one to the unknowns.

5 Generalization of the approach to the BEM

It turns out that the preceding approach, described in section 3, can be generalized to the Boundary Element Method (BEM). In such contexts, after discretization, one has to solve a linear system where the underlying matrix is fully populated. Typical examples are given by the acoustic or electromagnetic scattering. Indeed, let us consider a kernel[2] $G(x, y)$ for which one has to compute the matrix A defined by the entries

$$A_{ij} = \int_{x \in \Sigma_1} \int_{y \in \Sigma_2} \phi_i(x) G(x, y) \psi_j(y) \, dx \, dy \, . \tag{11}$$

This is very typical when one implements the BEM with a Galerkin approximation, the functions $(\phi_i)_{1 \leq i \leq N_1}$ and $(\psi_j)_{1 \leq j \leq N_2}$, being basis functions of possibly different finite element spaces. Taking discrete integration formulas on Σ_1 and Σ_2 respectively defined by the points and weights $(x_k, \omega_k)_{1 \leq k \leq N_{int1}}$ and $(y_l, \eta_l)_{1 \leq l \leq N_{int2}}$, leads to the approximation

$$A_{ij} \sim \sum_k \sum_l \phi_i(x_k) \omega_k G(x_k, y_l) \eta_l \psi_j(y_l) \, , \tag{12}$$

which enables us to write in a matrix form

$$A \sim \Phi W_x G W_y \Psi \, . \tag{13}$$

In this formula, the matrices W_x and W_y are the diagonal sparse matrices defined as before in (6) which depend on the quadrature weights ω_k and η_l respectively. The matrices Φ and Ψ are the (usually sparse) matrices defined as in (7) for the basis functions ϕ_i and ψ_j respectively, and G is the dense matrix of size $N_{int1} \times N_{int2}$ given by $G_{kl} = G(x_k, y_l)$.

Again, building the sparse matrices as before, one only needs to further compute the dense matrix G and assemble the preceding matrix A with only matrix products.

5.1 Generalization of the `integral` keyword (openDOM)

In terms of the syntax, we have extended the range of the `integral` keyword in order to handle such integrals. Indeed, the preceding formulas show that there are very little differences with respect to the preceding FEM formulations. Namely, we now need to handle

- Integrations over 2 possibly different domains Σ_1 and Σ_2;

[2] For instance, in the case of acoustic scattering, G is the Helmholtz kernel defined by $G(x, y) = \frac{\exp(ik|x-y|)}{4\pi|x-y|}$.

- Any kernel depending on two variables provided by the user;
- As before, two finite element spaces that are evaluated respectively on x and y.

Furthermore, other formulations exist for the BEM, such as the so-called *collocation method*, in which one of the two integrals is replaced by an evaluation at a given set of points. This case is also very much used when one computes (see the section below) the radiation of a computed solution on a given set of points.

To handle all these situations three possible cases are provided to the user:

- The case with two integrations and two finite element spaces. This corresponds to computing the matrix

$$A_{ij} = \int_{x\in\Sigma_1}\int_{y\in\Sigma_2} \phi_i(x)G(x,y)\psi_j(y)\,dx\,dy\,,$$

  and is simply performed in GYPSILAB by the following command.

```
A = integral(Sigma1, Sigma2, Phi, G, Psi);
```

  As before, the first finite element space `Phi` is considered as the test-function space while the second one, `Psi`, stands for the unknown. The two domains on which the integrations are performed are given in the same order than the finite element spaces (i.e. `Phi` and `Psi` are respectively defined on Σ_1 and Σ_2).

- The cases with only one integration and one finite element space. Two possibilities fall in this category. Namely the computation of the matrix

$$B_{ij} = \int_{x\in\Sigma_1} \phi_i(x)G(x,y_j)\,dx\,,$$

  for a collection of points y_j and the computation of the matrix

$$C_{ij} = \int_{y\in\Sigma_2} G(x_i,y)\psi_j(y)\,dy\,,$$

  for a collection of points x_i. Both cases are respectively (and similarly) handled by the two following commands.

```
B = integral(Sigma1,y_j,Phi,G);
C = integral(x_i,Sigma2,G,Psi);
```

In all the preceding commands, G should be a MATLAB function that takes as input a couple of 3 dimensional variables `X` and `Y` of respective sizes $N_X \times 3$ and $N_Y \times 3$. It should also be defined in order to possibly handle sets of such points in a vectorized way. As a sake of example, $G(x,y) = \exp(ix\cdot y)$ can simply be declared as

```
G = @(X,Y) exp(1i*X*Y');
```

where it is expected that both X and Y are matrices that contain 3 columns (and a number of lines equal to the number of points x and y respectively).

5.2 Regularization of the kernels

It is commonly known that usual kernels that are used in classical BEM formulations (e.g. Helmholtz kernel in acoustics) are singular near 0. This creates a difficulty when one uses the BEM which may give very inaccurate results since the quadrature rules used for the x and y integration respectively may possess points that are very close one to another. However, the kernels often possess a singularity which is asymptotically known.

In the toolbox, we provide the user with a way to regularize the considered kernel by computing a correction depending on its asymptotic behavior. As a sake of example, we consider the Helmholtz kernel which is used to solve the equations for acoustics (see section 5.4 for much more details)

$$G(x,y) = \frac{e^{ik|x-y|}}{4\pi|x-y|}\,. \tag{14}$$

This kernel possess a singularity when $x \sim y$ which asymptotic expansion reads as

$$G(x,y) \sim_{x\sim y} \frac{1}{4\pi|x-y|} + O(1)\,.$$

The idea is that the remainder is probably well approximated using Gauss quadrature rule and we only need to correct the singular part coming from the integration of $\frac{1}{|x-y|}$. In `Gypsilab`, this reads as

```
1 A = 1/(4*pi)*(integral(Sigma,Sigma,Vh,Gxy,Vh));
2 A = A+1/(4*pi)*regularize(Sigma,Sigma,Vh,'[1/r]',Vh);
```

The first line, as we have already seen assemble the full matrix defined by the integral

$$A_{ij} = \int_\Sigma \int_\Sigma G(x,y)\phi_i(x)\phi_j(y)\,dx\,dy\,,$$

where $(\phi_i)_i$ stands for the basis functions of the finite element space `Vh`.

The second line, however, regularizes the preceding integral by considering only the asymptotic behavior of G. This latter term computes and returns the **difference** between an accurate computation and the Gauss quadrature evaluation of

$$\int_\Sigma \int_\Sigma \frac{\phi(x)\phi(y)}{4\pi|x-y|}\,dx\,dy\,.$$

The Gauss quadrature is evaluated as before while the more accurate integration is computed using a semianalytic method in which the integral in y is computed analytically while the one in x is done using a Gauss rule. The correction terms are only computed for pairs of integration points that are close enough. Therefore the corresponding correction matrix is sparse.

5.3 Coupling with OPENHMX

As it is well-known, and easily seen from the formula (13), the matrices computed for the BEM are fully populated. Indeed, usual kernels $G(x,y)$ (e.g. the

Helmholtz kernel) never vanish for any couple (x, y) which therefore leads to a matrix G in (13) for which no entry vanishes. Furthermore, the number of integration points N_{int1} and N_{int2} is very often much larger than the corresponding numbers of degree of freedom. This means that the matrix G will have a size much larger that the final size of the (still fully populated) matrix A[3]. Both these facts limit very much the applicability of the preceding approach on classical computers to a number of degrees of freedom of a few thousands, which is often not sufficient in practice. For this reason we also provide a coupling with the GYPSILAB toolbox OPENHMX [28] in order to assemble directly a hierarchical $\mathcal{H}-$matrix compressed version of the preceding matrices. Namely, for a given tolerance `tol`, the commands

```
A = integral(Sigma1,Sigma2,Phi,G,Psi,tol);
B = integral(Sigma1,y_j,Phi,G,tol);
C = integral(x_i,Sigma2,G,Psi,tol);
```

return the same matrices than before, but now stored in a hierarchical $\mathcal{H}-$matrix format, and approximated to the desired tolerance. In particular, this enables the user to use the $+$, $-$, $*$, $\backslash$, `lu` or `spy` commands as if they were classical MATLAB matrix objects.

These generalizations, together with the possibility of directly assemble $\mathcal{H}$-matrices using the same kind of syntax, seem to us one of the cornerstones of the OPENFEM package. To our knowledge, there is, at the moment, no comparable software which handles BEM or compressed BEM matrices defined in a way as general as here.

5.4 Acoustic scattering

As a matter of example, we provide hereafter the resolution of the acoustic scattering of a sound soft sphere and the corresponding program in GYPSILAB-OPENFEM. For this test case, one considers a sphere of unit radius $\mathbb{S}^2$, and an incident acoustic wave given by

$$p_{inc}(x) = \exp(ikx \cdot d) \tag{15}$$

where k is the current wave number and d is the direction of propagation of the wave. It is well known that the total pressure p_{tot} outside the sphere is given by $p_{tot} = p_{inc} + p_{sca}$ where the scattered pressure wave obeys the formula

$$p_{sca}(x) = \int_{\mathbb{S}^2} G(x, y)\lambda(y)\, d\sigma(y)\,. \tag{16}$$

[3] As a sake of example, when one uses P^1 finite elements but an integration on triangles with 3 Gauss points per triangle, there are 6 times more Gauss points than unknowns (in a triangular mesh, the number of elements scales like twice the number of vertices). Calling N the number of unknowns, the final matrix A has a size N^2 while the matrix corresponding to the interaction of Gauss points is of size $(6N)^2 = 36N^2$ which is much bigger.

In the preceding formula, the Green kernel of Helmholtz equation is given by (14) and the density λ is computed using the so-called single layer formula

$$-p_{inc}(x) = \int_{\mathbb{S}^2} G(x,y)\lambda(y)\, d\sigma(y)\,, \tag{17}$$

for $x \in \mathbb{S}^2$. This ensures that $p_{tot} = 0$ on the sphere. Solving the equation (17) with the Galerkin finite element method amounts to solve the weak form

$$\int_{\mathbb{S}^2}\int_{\mathbb{S}^2} \mu(x)G(x,y)\lambda(y)\, d\sigma(x)\, d\sigma(y) = -\int_{\mathbb{S}^2} \mu(x)p_{inc}(x)\, d\sigma(x)\,, \tag{18}$$

where the test function μ and the unknown λ belong to a discrete finite element space. We take the space P^1 defined on a triangulation $\mathcal{T}_h$ of $\mathbb{S}^2$.

```
% Library path
addpath('../openMsh')
addpath('../openDom')
addpath('../openFem')
addpath('../openHmx')
% Parameters
N    = 1e3;
tol  = 1e-3;
X0   = [0 0 -1];
% Spherical mesh
sphere = mshSphere(N,1);
S2   = dom(sphere,3);
% Radiative mesh - Vertical square
square     = mshSquare(5*N,[5 5]);
square.vtx = square.vtx(:,[1 3 2]);
% Frequency adjusted to maximum edge size
stp = sphere.stp;
k   = 1/stp(2)
f   = (k*340)/(2*pi)
% Incident wave
PW = @(X) exp(1i*k*X*X0');
% Green kernel: G(x,y) = exp(ik|x-y|)/|x-y|
Gxy = @(X,Y) femGreenKernel(X,Y,'[exp(ikr)/r]',k);
% Finite element space
Vh = fem(sphere,'P1');
% Operator \int_Sx \int_Sy psi(x)' G(x,y) psi(y) dx dy
LHS=1/(4*pi)*(integral(S2,S2,Vh,Gxy,Vh,tol));
LHS=LHS+1/(4*pi)*regularize(S2,S2,Vh,'[1/r]',Vh);
% Wave trace --> \int_Sx psi(x)' pw(x) dx
RHS = integral(S2,Vh,PW);
% Solve linear system [-S] * lambda = - P0
lambda  = LHS \ RHS;
% Radiative operator \int_Sy G(x,y) psi(y) dy
Sdom = 1/(4*pi)*integral(square.vtx,S2,Gxy,Vh,tol);
Sdom = Sdom+1/(4*pi)*regularize(square.vtx,S2,'[1/r]',Vh);
```

```
36 % Domain solution : Pdom = Pinc + Psca
37 Pdom = PW(square.vtx) - Sdom * lambda;
38 % Graphical representation
39 figure
40 plot(square,abs(Pdom))
41 title('Total field solution')
42 colorbar
43 view(0,0);
44 hold off
```

The preceding program follows the traditional steps for solving the problem. Namely, one recognizes the spherical mesh and domain (lines 11-12), the radiative mesh on which we want to compute and plot the solution, here a square (lines 14-15), the incident plane wave (line 21), the Green kernel definition (line 23), the finite element space (line 25), the assembling of the operator (line 27-28), the construction of the right-hand side (line 30), and the resolution of the problem (line 32). The rest of the program consists in computing from the solution λ of (17), the total pressure on the radiative mesh, and plot it on the square mesh. Notice that due to the presence of the `tol` parameter in the assembling of the operator (and also of the radiative operator), the corresponding matrices are stored as $\mathcal{H}-$matrices. Notice also that the key part of the method (assembling and resolution) are completely contained between lines 21-32. The figures of the total pressure are given in Fig. 3 for the two cases $N = 10^4$ and $N = 9 \cdot 10^4$. The $\mathcal{H}$-matrix produced in the former case is shown in Fig. 4.

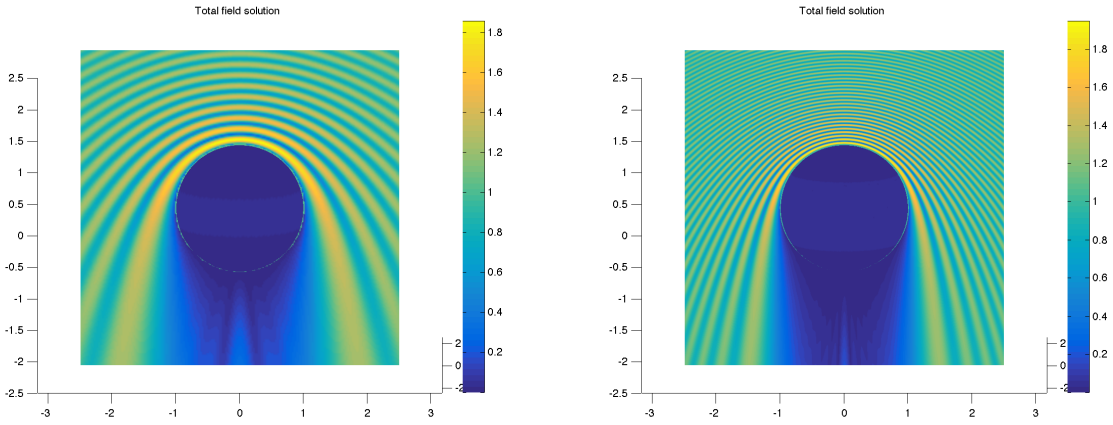

Figure 3: Magnitude of the pressure produced in the acoustic scattering by a unit sphere of a plane wave coming from above, using Gypsilab. On the left, the sphere is discretized with 10^4 vertices and the frequency is 10^3 Hz. On the right the sphere is discretized with $9 \cdot 10^4$ vertices and the frequency used is $3 \cdot 10^3$ Hz.

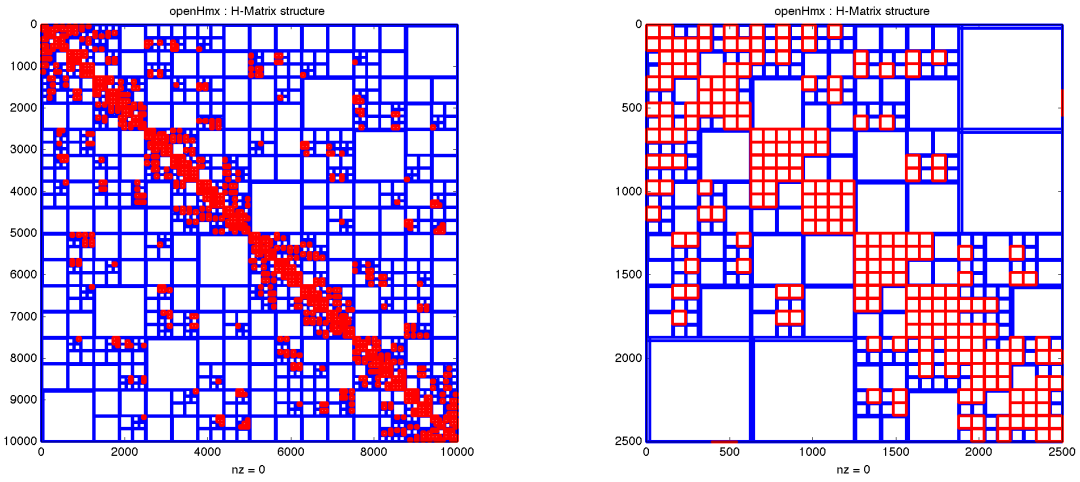

Figure 4: The $\mathcal{H}$-matrix produced in the case of the acoustic scattering with 10^4 unknowns. The left-hand side picture is obtained by using the `spy` command on the matrix itself. A zoom on the upper left part of the matrix (right) shows that each block conatins an information about the local rank.

5.5 Electromagnetic scattering

In electromagnetic scattering, formulations involving integral equations discretized with the BEM are also commonly used. We refer the reader to [4, 21] for an overview of classical properties of integral operators and discretizations. Three formulations are used to compute the magnetic current $J = n \times H$ on the surface of the scatterer. Namely, we distinguish

- The Electric Field Integral Equation (EFIE)

  $$TJ = -E_{inc,t}$$

  where the single layer operator T is defined by

  $$TJ = ik \int_\Sigma G(x,y)J(y)\,dy + \frac{i}{k}\nabla_x \int_\Sigma G(x,y)\mathrm{div}J(y)\,dy$$

  and $E_{inc,t}$ is the tangential component of the incident electric field.

- The Magnetic Field Integral Equation (MFIE)

  $$\left(\frac{1}{2} - n \times K\right) J = -n \times H_{inc,t}$$

  where the double layer operator K is defined by

  $$KJ = \int_\Sigma \nabla_y G(x,y)J(y)\,dy$$

and $H_{inc,t}$ is the tangential component of the incident magnetic field.

- The Combined Field Integral Equation (CFIE), used to prevent the ill-posedness of the preceding formulations at some frequencies. It is a linear combination of the Electric and Magnetic Field Integral Equations

$$\left(-\beta T + (1-\beta)\left(\frac{1}{2} - n \times K\right)\right) J = \beta E_{inc,t} - (1-\beta) n \times H_{inc,t}\,.$$

As before, the kernel is the Helmholtz Green kernel defined by (14).

The classical finite element formulation for this problem uses the Raviart-Thomas elements for J which are available in OPENFEM. The key part of the program assembling the CFIE operator and solving the scattering problem is given hereafter. For simplicity, we only focus on the assembling and solving parts and do not provide the initialization part and the radiation and ploting parts.

```
% Incident direction and field
X0 = [0 0 -1];
E  = [0 1  0];    % Polarization of the electric field
H  = cross(X0,E); % Polarization of the magnetic field
% Incident Plane wave (electromagnetic field)
PWE{1} = @(X) exp(1i*k*X*X0') * E(1);
PWE{2} = @(X) exp(1i*k*X*X0') * E(2);
PWE{3} = @(X) exp(1i*k*X*X0') * E(3);
%
PWH{1} = @(X) exp(1i*k*X*X0') * H(1);
PWH{2} = @(X) exp(1i*k*X*X0') * H(2);
PWH{3} = @(X) exp(1i*k*X*X0') * H(3);
% Green kernel function G(x,y) = exp(ik|x-y|)/|x-y|
Gxy    = @(X,Y) femGreenKernel(X,Y,'[exp(ikr)/r]',k);
Hxy{1} = @(X,Y) femGreenKernel(X,Y,'grady[exp(ikr)/r]1',k) ;
Hxy{2} = @(X,Y) femGreenKernel(X,Y,'grady[exp(ikr)/r]2',k) ;
Hxy{3} = @(X,Y) femGreenKernel(X,Y,'grady[exp(ikr)/r]3',k) ;
% Finite elements
Vh = fem(sphere,'RWG');
% Finite element mass matrix
Id = integral(sigma,Vh,Vh);
% Finite element boundary operator
T = 1i*k/(4*pi)*integral(sigma,sigma,Vh,Gxy,Vh,tol) ...
 -1i/(4*pi*k)*integral(sigma,sigma,div(Vh),Gxy,div(Vh),tol);
T = T + 1i*k/(4*pi)*regularize(sigma,sigma,Vh,'[1/r]',Vh) ...
 -1i/(4*pi*k)*regularize(sigma,sigma,div(Vh),'[1/r]',div(Vh));
% Finite element boundary operator
nxK = 1/(4*pi)*integral(sigma,sigma,nx(Vh),Hxy,Vh,tol);
nxK = nxK+1/(4*pi)*regularize(sigma,sigma,nx(Vh),'grady[1/r]',
    Vh);
% Left hand side
LHS = -beta * T + (1-beta) * (0.5*Id - nxK);
```

```
32 % Right hand side
33 RHS = beta*integral(sigma,Vh,PWE) ...
34          - (1-beta)*integral(sigma,nx(Vh),PWH);
35 % Solve linear system
36 J = LHS \ RHS;
```

As one can see, the program is a direct transcription of the mathematical weak formulation of the problem. This follows the same lines as in the acoustic scattering problem except for the operators that are different and the finite element used. Notice also the regularization of the double layer kernel in line 29.

6 Performances

6.1 Performances in FEM

In this section we compare GYPSILAB with FREEFEM++. The machine that we have used for this comparison possesses Xeon X5675 processors with a frequency of 3.07 GHz and 146 GB of memory. Although the machine is equipped with two such processors, meaning that up to 12 cores could be used for the computation, we only chose a single core to run the test, both for FREEFEM++ and MATLAB which is therefore launched using the `-singleCompThread` option. We have chosen in what follows to assemble three dimensional finite element matrices with three different finite elements, namely, the Lagrange piecewise linear P^1, the Raviart-Thomas RT^0 and the order 0 Nédélec elements. We use the notation $(\phi_i)_i$, $(\psi_i)_i$ and $(\theta_i)_i$ for the set of basis functions for those three finite elements respectively. In the tables below we show the computational time needed to assemble the three sparse matrices

$$A_{1,ij} = \int_C \nabla\phi_i \cdot \nabla\phi_j \, dx\,, \quad A_{2,ij} = \int_C \mathrm{div}(\psi_i)\mathrm{div}(\psi_j)\, dx$$

$$\text{and } A_{3,ij} = \int_C \mathrm{curl}(\theta_i) \cdot \mathrm{curl}(\theta_j)\, dx\,,$$

where C is the unit cube in $\mathbb{R}^3$ meshed regularly, with $(N+1)^3$ points and consider values for N that range from 20 up to 100 depending on the case. This leads to a number of degrees of freedom which is given as N_{dof}. Eventually, we have used FREEFEM++ version 3.34 and MATLAB R2017a to run the test. The first Table 2, shows the timings for the assembling of the Laplacian matrix A_1 with P^1 finite elements. In Table 3, we give the assembling time of the matrix A_2 where 3D Raviart-Thomas finite elements are used. Eventually, in Table 4, we give the assembling time of the matrix A_3 where 3D Nédélec finite elements are used. We observe that both FREEFEM++ and GYPSILAB give assembling times that are very similar, even for quite large matrices (up to roughly $10^6 \times 10^6$).

Here, there is a clear advantage in terms of speed for GYPSILAB. Indeed, for each case the time to assemble the matrix is roughly the half of that of FREEFEM++.

From these tables, we can observe that, although not yet as complete as FreeFem++, Gypsilab-openFEM has performances very much comparable. It is therefore a right tool for prototyping models in an efficient way.

N	FreeFem++	Gypsilab-openFEM	N_{dof}
20	0.40	0.46	9 261
30	1.36	1.21	29 791
40	3.12	2.81	68 921
50	6.50	5.43	132 651
60	10.9	9.98	226 981
70	17.03	15.9	357 911
80	25.77	24.77	531 441
90	37.89	36.17	753 571
100	50.37	50.28	1 030 301

Table 2: Timings for the assembling of the Laplace matrix A_1 for P^1 finite elements. The unit cube C is meshed with $(N+1)^3$ points, N ranging from 20 to 100 and the total number of degrees of freedom is given by N_{dof}.

N	FreeFem++	Gypsilab-openFEM	N_{dof}
20	1.01	0.74	98 400
30	3.78	2.38	329 400
40	8.10	5.77	777 600
50	15.67	11.63	1 515 000
60	28.0	20.64	2 613 600

Table 3: Timings for the assembling of the matrix A_2 for Raviart-Thomas finite elements. The unit cube C is meshed with $(N+1)^3$ points, N ranging from 20 to 60 and the total number of degrees of freedom is given by N_{dof}.

6.2 Performances in BEM

We report in this section the performances attained by the acoustic scattering of the sphere previously described. Here, the goal is not to compare with another software and we have used a 4 core machine to run the test. The Gypsilab toolbox takes advantage of the mutlicore parallelism. We give hereafter the timings for different meshes of the sphere corresponding to an increasing number N of degrees of freedom in the underlying system. The first part of Table 5 gives the timings to assemble the full BEM matrix for sizes ranging from 1000 to 150000 degrees of freedom. Above 10000 unknowns, the matrix of the kernel computed at the integration points no longer fits into the memory of the available machine and the swap in memory significantly slows down the code

N	FreeFem++	Gypsilab-openFEM	N_{dof}
20	2.8	1.11	59 660
30	9.24	3.82	197 190
40	22.78	9.42	462 520
50	43.6	19.91	897 650
60	74.71	34.61	1 544 580
70	117.67	56.13	2 445 310

Table 4: Timings for the assembling of the matrix A_3 for Nédélec finite elements. The unit cube C is meshed with $(N+1)^3$ points, N ranging from 20 to 70 and the total number of degrees of freedom is given by N_{dof}.

making the method impracticable. Therefore, we turn to use the hierarchical compression for the matrix, i.e. the $\mathcal{H}-$matrix paradigm available through the use of openHMX. This enables us to increase the size of reachable problems by an order of magnitude and more. This is reported in the bottom part of Table 5. Notice that the frequency of the problem is adapted to the precision of the mesh as shown in the last column of the table.

In order to see the effect of the frequency on the construction of the $\mathcal{H}$-matrix and the resolution, we have tried to fix the frequency to 316 Hz (the smallest value for the preceding case) and check the influence on the assembling, regularization and solve timings in the problem. The data are given in Table 6. It can be seen that the underlying matrix is much easier compressed and quickly assembled and the resolution time is also significantly reduced. Indeed, the time to assemble the $\mathcal{H}$-matrix becomes proportional to the number of unknowns.

7 Conclusion

The openFEM toolbox of the package Gypsilab is a numerical library written in full Matlab that allows the user to solve PDE using the finite element technique. Very much inspired by the FreeFem++ formalism, the package contains classical FEM and BEM functionalities. In this latter case, the library allows the user to store the operators in a $\mathcal{H}$-matrix format that makes it possible to factorize the underlying matrix and solve using a direct method the linear system. We are not aware of any comparable software that combines ease of use, generality and performances to the level reached by openFEM. We have shown illustrative examples in several problems ranging from classical academic Laplace problems to the Combined Field Integral Equation in electromagnetism. Eventually, a short performance analysis shows that the library possesses enough performance to run problems with a number of unknowns of the order of a million in reasonable times. A lot remains to be done, as extending the available finite elements, proposing different compression strategies or coupling FEM and BEM problems, that we wish to study now. In particular solving coupled FEM-

N_{dof}	T_{ass}	T_{reg}	T_{sol}	T^H_{ass}	T^H_{reg}	T^H_{sol}	Freq. (Hz)
1000	1.20	0.84	0.06	2.4	0.80	0.21	316
3000	8.5	1.51	0.66	9.11	1.73	2.23	547
5000	23.7	2.26	2.67	13.06	2.57	5.59	707
10000	102.2	10.4	23.0	27.28	5.24	17.68	1000
20000	NA	NA	NA	63.51	9.99	54.42	1414
40000	NA	NA	NA	170.26	21.18	187.39	2000
80000	NA	NA	NA	487.09	47.51	653.8	2828
150000	NA	NA	NA	1412.6	230.53	3664.7	3872

Table 5: Timings in seconds for assembling, regularize and solve the problem of acoustic scattering given in section 5.4. The second half of the table corresponds to the timings using the $\mathcal{H}-$matrix approach using in the GYPSILAB-openHMX package. For problems of moderate size it is slightly faster to use the classical BEM approach, while the sizes corresponding to the bottom lines are beyond reach for this method. Notice that when we solve this problem using the $\mathcal{H}-$matrices, a complete LU factorization is computed. This is not optimal in the present case, in particular when compared to other compression techniques such as the FMM, since the underlying linear system is solved only once (with only one right hand side).

BEM problems in GYPSILAB will be the subject of a forthcoming paper.

Finally, GYPSILAB is available under GPL licence [28] and therefore makes it a desirable tool for prototyping.

References

[1] Alouges, F. and Aussal, M.: The Sparse Cardinal Sine Decomposition and its Application to fast numerical convolution. Numerical Algorithms, 70(2), 427–448 (2015).

[2] Alouges, F., Aussal, M., Lefebvre-Lepot, A., Pigeonneau F. and Sellier, A.: Application of the sparse cardinal sine decomposition applied to 3D Stokes flows, International Journal of Comp. Meth. and Exp. Meas. 5(3) (2017).

[3] Alouges, F., Aussal M. and Parolin, E.: FEM-BEM coupling for electromagnetism with the Sparse Cardinal Sine Decomposition, accepted to ESAIM Procs (2017).

[4] Colton, D. and Kress, R.: Inverse Acoustic and Electromagnetic Scattering Theory, Second Edition. Springer-Verlag, New York, (1998)

[5] Greengard, L.: The rapid evaluation of potential fields in particle systems. MIT Press (1988)

[6] Hackbusch, W.: Hierarchische Matrizen. Springer (2009) .

N_{dof}	T^H_{ass}	T^H_{reg}	T^H_{sol}	Freq. (Hz)
10000	16.88	5.93	14.55	316
20000	32.56	11.7	34.31	316
40000	67.18	23.77	82.11	316
80000	134.33	49.43	179	316
150000	253.1	98.2	423	316

Table 6: Timings in seconds for assembling, regularize and solve the problem of acoustic scattering given in section 5.4 at a fixed frequency $f = 316Hz$ on the unit sphere. The number of unknown is given as N_{dof} and the $\mathcal{H}$-matrix compression technique is ueed.

[7] See http://www.cims.nyu.edu/cmcl/fmm3dlib/fmm3dlib.html .

[8] Hecht, F.: New development in FreeFem++, J. Numer. Math., 20, 3-4, 251–365 (2012). See also `http://www.freefem.org`.

[9] See `http://www.feelpp.org`.

[10] Śmigaj, W., Arridge, S., Betcke, T., Phillips, J. and Schweiger, M., Solving Boundary Integral Problems with BEM++, ACM Trans. Math. Software 41, 6:1–6:40 (2015).

[11] See `https://uma.ensta-paristech.fr/soft/XLiFE++/` .

[12] See `https://fenicsproject.org` .

[13] See `http://firedrakeproject.org`.

[14] Geuzaine, C. GetDP: a general finite-element solver for the de Rham complex,PAMM Volume 7 Issue 1. Special Issue: Sixth International Congress on Industrial Applied Mathematics (ICIAM07) and GAMM Annual Meeting, 7, 1010603–1010604, Zürich 2007, Wiley (2008).

[15] See `http://getdp.info`.

[16] See `https://www.comsol.fr`.

[17] Alberty, J., Carstensen, C. and Funken, S. A.: Remarks around 50 lines of Matlab: short finite element implementation, Numerical Algorithms 20, 117–137 (1999).

[18] Funken, S., Praetorius, D. and Wissgott, P. : Efficient implementation of adaptive P1-FEM in Matlab, Comput. Methods Appl. Math., 11(4), 460–490 (2011).

[19] Sutton, O. J.:The virtual element method in 50 lines of Matlab, Numerical Algorithms, 75(4), 1141–1159 (2017).

[20] Kwon, Y. W. and Bang, H.: The finite element method using Matlab, second edition, CRC Press, (2000).

[21] Nédélec, J.-C.: Acoustic and Electromagnetic Equations, Integral Representations for Harmonic Problems, Springer, 2001.

[22] Cuvelier, F., Japhet, C. and Scarella, G.: An efficient way to assemble finite element matrices in vector languages

[23] Anjam, I. and Valdman, J.: Fast MATLAB assembly of FEM matrices in 2D and 3D: Edge elements. Applied Mathematics and Computation. 267. (2014). doi:10.1016/j.amc.2015.03.105.

[24] Rahman, T. and Valdman, J.: Fast MATLAB assembly of FEM matrices in 2D and 3D: Nodal elements, Appl. Math. Comput. 219, 7151–7158 (2013).

[25] See `https://www.esi-group.com/fr/solutions-logicielles/performance-virtuelle/vibro-acoustique`.

[26] See `https://www.esi-group.com/software-solutions/virtual-environment/electromagnetics/cem-one/efield-time-domain`.

[27] See `https://imacs.polytechnique.fr/ASERIS.htm`.

[28] Gypsilab is freely available under GPL 3.0 license, see `www.cmap.polytechnique.fr/~aussal/gypsilab` , and on GitHub at `https://github.com/matthieuaussal/gypsilab`.