1 Summary: TL;DR

1.1 Kind of problem

Let N items, such as disks. Therefore, N+1 domains are considered:

- N interior domains
- one exterior domain

Note that the N interior domains should be *empty*.

In other words, in any case, let N *interfaces*. To each interface, some BIO are considered, and then we would like to solve a BIE.

- BIE: Boundary Integral Equation
- BIO: Boudnary Integral Operator

BIE means an weak formulation based on BIO(s).

1.2 Note about the mechanism behind

Roughly speaking:

- BIO is an `Operator`
- BIE is also an `Operator` which is then transformed into a `Matrix`

An `Operator` manages only *metadata* which is (more or less) pointers, i.e about the *structure*. By structure, we have to understand the size and the MPI rank, but also the block structure if it is.

Basically an `Operator` is represented by a COO matrix format:

```
A[row, col] = (row, col, *pointer)
```

i.e. an `Operator` is nothing more than a list of 3 items.

By construction, the constructor of the operator fills `(0, 0, *p)` where the pointer `p` points to itself ; then the method `addBlock` populates this list.

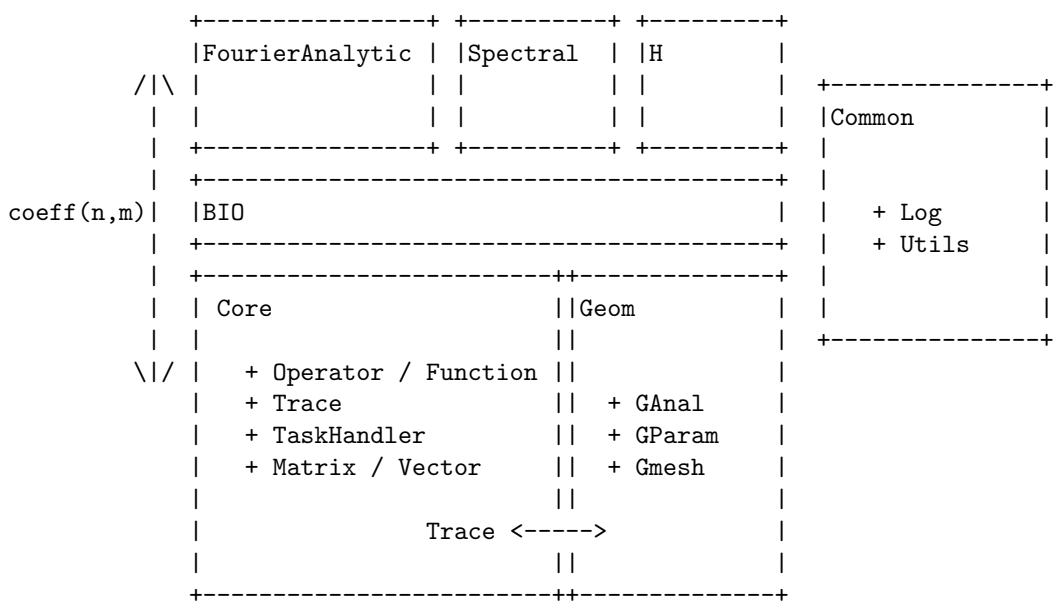
Once the population is satisfied, the method `assemb` launches the distribution and the assembly parts then return a `Matrix`.

2 Structures of the Files

```
+ doc/
|  + blah.md
|  + tuto.md
|  + etc...
|
+ ex/   (and/or tests)
|  + mudiff/
|  + stf/
|  + etc...
|
+ py/
|  + SWIG-binding-for-later
|
+ src/
|
+--+ BIO/
|  |
|  +--+ FourierAnalytic/
|  |     + Laplace/
|  |     + Helmholtz/
|  +--+ Spectral/
|  |     + Fourier/
|  |     + Tchebichef/
|  +--+ H/
|        + Laplace/
|        + Helmholtz/
|
+--+ Common/
|  |
|  +--+ Log
|     + Utils   (OverLayerPetsc)
|
+--+ Core/
|  |
|  +--+ Operator / Function
|     + Trace
|     + TaskHandler
|     + Matrix / Vector
|
+--+ Geom/
      + GAnal/
      + GParam/
      + GMesh/
```

3 Structure

More or less the oganization, and the details are given below.



3.1 Note

A `ClassThatRocks` starts by upper letter. A `functionThatRocks` starts by lower letter. A list of items finished by `s` as `beers`.

4 Common: `Log` / `Utils`

A class mimicking the `Message` class of `GetDP`: also used to initialize and finalize.

`Utils` encapsulates all the PETSc tools that we need inside a `class` (and maybe a `namespace`) e.g. `MatCreate`, `matmul` etc.

The advantages are: 1/ when the PETSc API changes, our update is easier, 2/ it is easy to change the matrix representation if we want to, i.e. only changing the `namespace` (useful for collaboration as Xavier) and 3/ this avoid to include all the *weird* PETSc types.

5 Geom

On the top, let a empty class, with a string name. Then all the geometry derives from this class. This allows to write generic and easy maintainable code, keeping in mind other possibilities such as mesh (future).

6 Trace

An *trace* represents the unknowns onto a `Geom`.

Note that `Trace` needs to have a numbering of the unknowns, i.e a list (bijection) between the *modes*. Even if it seems not useful, the key point of this `dofHandler` is to manage the so prone error indexing.

Moreover, this allows to have access the local numbering of a interface, which seems useful for debugging purposes (or for guru hacks).

- **attributes**
  - `kind` : `char` not useful for now
  - `geom` : pointer to `Geom`
  - `size`: `int`
  - `dofs` : `int[2M+1][2]` and by default (constructor),

$\sim$ [-M,..., -1, 0, 1,..., M] <—> [0, 1,..., 2M+1] $\sim$ where 2M+1 corresponds to `size`.

- **methods**
  - constructor and destructor
  - `setDofHandler` : `int[2M+1]` `|->` `int[2M+1][2]` it could be overloaded by the user.
  - `getDofHandler`

In *h*, `dofs` is the link between the nodes/elements numbering and the real numbering used.

In *spectral*, `dofs` is the link between the "physical modes" numbering and the the real numbering used.

7 TaskHandler

It is an independant part and it could contain a dictionnary between the interface index and the processus (`MPI rank`).

The default mapping could be: one interface per processus. However, if one iterface is really larger than the other ones, the user has the ability to map as he/she wants to distribute the load.

Moreover, this object could contain an update to *who* is *where*.

What I have in mind is: let assembly a mudiff matrix per block (one block per processus) and then let assembly the block diagonal preconditioner, the rebuilding is avoided.

In other words, this class also manages the useful *copy*.

- **attributes**
- `loads` : `int[N+1][2]`
- **methods**
  - constructor and destructor
  - `setAssignInt2Proc`
  - `getProc` gives the `Interface`(s) associated to a MPI rank
  - `getInterface` gives the MPI rank(s) associated to a `Interface`

The key point is: let PETSc manages as possible as it is possible

Note that how it is splitted between `TaskHandler` and `Operator` is not fully clear right now.

8 Operator (the core of the core)

It should only and simply be a layer on the top of the matrices of PETSc, i.e., a nice handler and common interaction of matrices, but to be clear `Operator` acts only on the `pointer` level.

`Operator` is only something that collect and distribute.

Moreover, it is the same object that manages all the matrices. Therefore, it is a *big* object.

Then, if the method `addBlock` is called, there is two options, switch to

1. block matrix format of PETSc (nested or monolitic etc.)

2. a classical matrix format

The first option seems clear and it allows to use mixed kind of blocks.

However, the global matrix could be turn into one of the classical formats and then assembly in this format.

In other words, or maybe more precisely, it would be nice to initialize all the *actions* (by pointers?), then when the `assemb` method is called, all the memory and computations are launched; i.e.,

1. performs from `rank0` the organization (or everybody works and organizes the same things instead of waiting and so this avoids some message passing)
2. launches, which means (more or less)
   - `MatCreate`
   - `MatSetValue`

and different ways are possible,

- a `MatCreate` per *block* and then the PETSc block feature
- only one `MatCreate` and `MatSetValue` calls a different function per *block*

Use a classical PETSc `Mat` format seems easier to let PETSc to distribute all the load.

Moreover, PETSc and `MatSetValue` needs a *band* and it seems easier, once all the blocks are added, i.e. all the *global* and *local* are known to assembly in only one format, letting PETSc handles the distributgion.

- **attributes**
  - `shape` : `int[2]` size of the returned `Matrix`
  - `coo` : `tCoo[]` and create a `struct` s.t. (row, col, val). with a nice list *appending*. Or
    - `cols` : list of integer
    - `rows` : list of integer
    - `vals` : list of `Operator` pointer
  - `Shape` : `int[2]` size block structure
- **methods**
  - constructor and destructor

- `setValue` : function that feeds the '`PetscScalar values[]`' field inside the function `MatSetValue`.
- `addBlock`
- `assemb`
- `update` : update the values of `shape` and `Shape`
- `diagonal` : get the diagonal blocks

9 FourierAnalytical

This is only functions.

```
Operator singleLayerHelm_Fourier(Interface test, Interface trial, wavenumber)
Operator doubleLayerHelm_Fourier(Interface test, Interface trial, wavenumber)
Operator adjointLayerHelm_Fourier(Interface test, Interface trial, wavenumber)
Operator hyperLayerHelm_Fourier(Interface test, Interface trial, wavenumber)
```

Inside them, they fill the `coeff` function of an `Operator`.

9.1 Why `(test, trial)` and not `(trial, test)` ?

because

$$A_{ij} =< \phi_i, A\psi_j >= \int \int A\psi_j \bar{\phi}_i$$

with ϕ is a test-function and ψ is a trial-function.

In other words, the test-functions correspond to the row and the trial-function correspond to the column.

10 Still remains ?

How to create the Right-Hand Side ?

In other words, a question remains: what does `projectFunction_Fourier` return ? especially in the context of block matrix.

There is two solutions:

1. add an object `Function`
2. add a method to `Trace` to project onto.

10.1 Function

A solution should be to add a `Fucntion` class, which represents a vector, as `Operator` represents a matrix.

|pointer |`Operator` |`Function`| |:———:|:———-:|———-:| |allocated |`Matrix` |`Vector` |

In this way, the prototype is

```
Function projectFunction_Fourier(Interface test, Complex func);

// with func such that
Complex func(tPoint p);

//with tPoint a new type, e.g,
typedef struct{
 float x;
 float y;
} tPoint;
// or a class Point
```

10.2 Trace.project

Do not know if it is a good solution. Because information need to be added to `Trace`, or at least links need to be done.

However, who is in charge to export a *vector* solution into a `pos/vtk` file ?

11 Matrix / Vector

`Mat` / `Vec` needs to be encapsulated into a class, in order to overload some basic arithmetic operation as `A*x`

Basically, these objects are the matrix and the vector of PETSc, to keep all the power and in-place tools, without reinventing the wheel.

- **attributes**
  - `shape` : `int[2]`
  - `kind` : char
- **methods**
  - constructor and destructor

- `matvec`
- `matmat`
- etc.

12 Example

to be written
