$\mathcal{H}$-matrices as a compression tool in boundary integral equations

Luiz M. Faria [1]

[1]POEMS, INRIA/CNRS/ENSTA, Palaiseau, France

October 2, 2024

Overview

- This talk is about how dense matrices appear when solving PDEs by integral equation (IE) methods

$$\underbrace{\boxed{\text{PDE}} \rightarrow \boxed{\text{Integral equation}}}_{\textbf{continuous level}} \rightarrow \underbrace{\boxed{\text{Discretization}} \rightarrow \boxed{\text{Linear algebra}}}_{\textbf{discrete level}}$$

Overview

- This talk is about how dense matrices appear when solving PDEs by integral equation (IE) methods

$$\underbrace{\boxed{\text{PDE}} \rightarrow \boxed{\text{Integral equation}}}_{\textbf{continuous level}} \rightarrow \underbrace{\boxed{\text{Discretization}} \rightarrow \boxed{\text{Linear algebra}}}_{\textbf{discrete level}}$$

- More specifically, it is about the role played by $\mathcal{H}$-matrices in the solution pipeline

Overview

- This talk is about how dense matrices appear when solving PDEs by integral equation (IE) methods

$$\underbrace{\boxed{\text{PDE}} \to \boxed{\text{Integral equation}}}_{\textbf{continuous level}} \to \underbrace{\boxed{\text{Discretization}} \to \boxed{\text{Linear algebra}}}_{\textbf{discrete level}}$$

- More specifically, it is about the role played by $\mathcal{H}$-matrices in the solution pipeline
- And my experience developing an $\mathcal{H}$-matrices library and integrating it in `Inti.jl`

Overview

- This talk is about how dense matrices appear when solving PDEs by integral equation (IE) methods

$$\underbrace{\boxed{\text{PDE}} \rightarrow \boxed{\text{Integral equation}}}_{\textbf{continuous level}} \rightarrow \underbrace{\boxed{\text{Discretization}} \rightarrow \boxed{\text{Linear algebra}}}_{\textbf{discrete level}}$$

- More specifically, it is about the role played by $\mathcal{H}$-matrices in the solution pipeline
- And my experience developing an $\mathcal{H}$-matrices library and integrating it in `Inti.jl`

Disclaimer

- My interaction with $\mathcal{H}$-matrices is mostly through code/software

Overview

- This talk is about how dense matrices appear when solving PDEs by integral equation (IE) methods

$$\underbrace{\boxed{\text{PDE}} \to \boxed{\text{Integral equation}}}_{\textbf{continuous level}} \to \underbrace{\boxed{\text{Discretization}} \to \boxed{\text{Linear algebra}}}_{\textbf{discrete level}}$$

- More specifically, it is about the role played by $\mathcal{H}$-matrices in the solution pipeline
- And my experience developing an $\mathcal{H}$-matrices library and integrating it in `Inti.jl`

Disclaimer

- My interaction with $\mathcal{H}$-matrices is mostly through code/software
- They are an essential tool for many of the libraries I develop/use

Overview

- This talk is about how dense matrices appear when solving PDEs by integral equation (IE) methods

$$\underbrace{\boxed{\text{PDE}} \rightarrow \boxed{\text{Integral equation}}}_{\textbf{continuous level}} \rightarrow \underbrace{\boxed{\text{Discretization}} \rightarrow \boxed{\text{Linear algebra}}}_{\textbf{discrete level}}$$

- More specifically, it is about the role played by $\mathcal{H}$-matrices in the solution pipeline
- And my experience developing an $\mathcal{H}$-matrices library and integrating it in `Inti.jl`

Disclaimer

- My interaction with $\mathcal{H}$-matrices is mostly through code/software
- They are an essential tool for many of the libraries I develop/use
- But they are not my main research area...

A word about `Inti.jl`

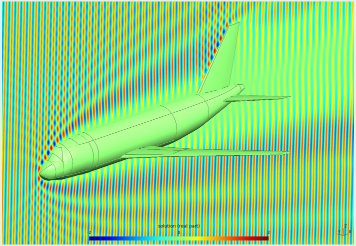
`Inti.jl` is a Julia library for solving boundary and volume integral equations. Features include:

- **High-order** singular and nearly-singular quadratures
- Support for **compression methods** (FMM, $\mathcal{H}$-matrices, IFGF)
- Handling of **complex meshes** in 2D and 3D
- Online **documentation** with tutorials and examples
- Open-source (LGPL-2.1) and under **active development**

https://github.com/IntegralEquations/Inti.jl

An example

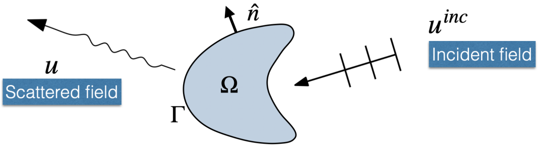
```julia
using Inti, IterativeSolvers, HMatrices
msh = import_mesh("airplane.msh")
Omega = Domain(msh; dim = 3)
Gamma = boundary(Omega)
Q = Quadrature(msh[Gamma], qorder = 3)
k = 4*pi
S, D = single_double_layer(
    pde = Helmholtz(k, dim=3),
    target = Q,
    source = Q,
    compression = (method = :hmatrix, tol = 1e-4),
    correction = (method = :dim, ),
)
L = I/2 + D - im*k*S # <== Ability to combine operators!!!
rhs = map(q -> -exp(im * k * dot(q.coords, [1, 0, 0])), Q)
sigma = gmres(L, rhs)
```



Geometry → Mesh → Solver → Visualization

From PDEs to integral equations

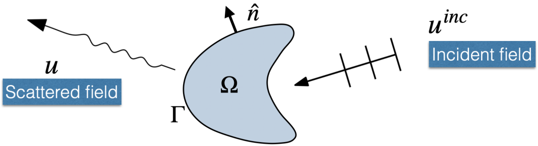
- Many PDEs of interest can be reformulated as integral equations
- Examples include: Laplace, Helmholtz, Stokes, and Maxwell equations



- When applicable, IEs can be a very efficient/accurate method
- Advantages
  - \+ Reduction of dimensionality (less DOFs)
  - \+ Exactly solve the PDE
- Drawbacks
  - \- Discretization leads to dense matrices
  - \- Integral operators are singular (challenging numerical integration)

From PDEs to integral equations

- Many PDEs of interest can be reformulated as integral equations
- Examples include: Laplace, Helmholtz, Stokes, and Maxwell equations



- When applicable, IEs can be a very efficient/accurate method
- Advantages
  + Reduction of dimensionality (less DOFs)
  + Exactly solve the PDE
- Drawbacks
  - Discretization leads to dense matrices
  - Integral operators are singular (challenging numerical integration)

Drawbacks significantly hinder usage of IEs in practice

From PDEs to integral equations

Poisson Dirichlet B.V.P.

$$\Delta u = f \qquad \boldsymbol{r} \in \Omega,$$
$$u = g \qquad \boldsymbol{x} \in \Gamma$$

1. Build a particular solution using a **volume potential**

$$u_p(\boldsymbol{x}) = \int_\Omega G(\boldsymbol{x}, \boldsymbol{y}) f(\boldsymbol{y}) \, \mathrm{d}\boldsymbol{y}$$

2. Use **boundary integral operators** to enforce the boundary conditions

$$u(\boldsymbol{x}) = u_p(\boldsymbol{x}) + \int_\Gamma G(\boldsymbol{x}, \boldsymbol{y}) \sigma(\boldsymbol{y}) \, \mathrm{d}s(\boldsymbol{y})$$

where σ solves

$$\int_\Gamma G(\boldsymbol{x}, \boldsymbol{y}) \sigma(\boldsymbol{y}) \, \mathrm{d}s(\boldsymbol{y}) = g(\boldsymbol{x}) - u_p(\boldsymbol{x}), \quad \boldsymbol{x} \in \Gamma$$

From PDEs to integral equations

"General" B.V.P.

$$\mathcal{L}u = f \qquad \boldsymbol{r} \in \Omega,$$
$$a\gamma_0 u + b\gamma_1 u = g \qquad \boldsymbol{x} \in \Gamma$$

From PDEs to integral equations

"General" B.V.P.

$$\begin{aligned} \mathcal{L}u &= f \qquad \boldsymbol{r} \in \Omega, \\ a\gamma_0 u + b\gamma_1 u &= g \qquad \boldsymbol{x} \in \Gamma \end{aligned}$$

Required machinery:

- Volume potential:

$$\mathcal{V}[f](\boldsymbol{x}) := \int_\Omega G(\boldsymbol{x},\boldsymbol{y}) f(\boldsymbol{y}) \,\mathrm{d}\boldsymbol{y}$$

- Boundary potentials:

$$\begin{aligned} \mathcal{S}[\sigma](\boldsymbol{x}) &:= \int_\Gamma G(\boldsymbol{x},\boldsymbol{y})\sigma(\boldsymbol{y}) \,\mathrm{d}s(\boldsymbol{y}), \\ \mathcal{K}[\sigma](\boldsymbol{x}) &:= \int_\Gamma (\gamma_{1,\boldsymbol{y}} G(\boldsymbol{x},\boldsymbol{y}))^\top \sigma(\boldsymbol{y}) \,\mathrm{d}s(\boldsymbol{y}) \end{aligned}$$

- As well as their gradients/traces...

Building blocks

Building blocks of integral equation methods

$$\mathcal{V}[f](\boldsymbol{x}) := \int_\Omega G(\boldsymbol{x},\boldsymbol{y}) f(\boldsymbol{y}) \, \mathrm{d}\boldsymbol{y}, \qquad \text{(Newtonian potential)}$$

$$S[\sigma](\boldsymbol{x}) := \int_\Gamma G(\boldsymbol{x},\boldsymbol{y}) \sigma(\boldsymbol{y}) \mathrm{d}s(\boldsymbol{y}), \qquad \text{(Single-layer)}$$

$$K[\sigma](\boldsymbol{x}) := \text{p.v.} \int_\Gamma \left[\gamma_{1,\boldsymbol{y}} G(\boldsymbol{x},\boldsymbol{y})\right]^T \sigma(\boldsymbol{y}) \mathrm{d}s(\boldsymbol{y}), \qquad \text{(Double-layer)}$$

$$K'[\sigma](\boldsymbol{x}) := \text{p.v.} \int_\Gamma \left(\gamma_{1,\boldsymbol{x}} G(\boldsymbol{x},\boldsymbol{y})\right) \sigma(\boldsymbol{y}) \mathrm{d}s(\boldsymbol{y}), \qquad \text{(Adjoint double-layer)}$$

$$T[\sigma](\boldsymbol{x}) := \text{f.p.} \int_\Gamma \gamma_{1,\boldsymbol{x}} \left[\gamma_{1,\boldsymbol{y}} G(\boldsymbol{x},\boldsymbol{y})\right]^T \sigma(\boldsymbol{y}) \mathrm{d}s(\boldsymbol{y}) \qquad \text{(Hypersingular)}$$

Building blocks

Building blocks of integral equation methods

$$\mathcal{V}[f](\boldsymbol{x}) := \int_\Omega G(\boldsymbol{x},\boldsymbol{y}) f(\boldsymbol{y}) \, \mathrm{d}\boldsymbol{y}, \qquad \text{(Newtonian potential)}$$

$$S[\sigma](\boldsymbol{x}) := \int_\Gamma G(\boldsymbol{x},\boldsymbol{y}) \sigma(\boldsymbol{y}) \mathrm{d}s(\boldsymbol{y}), \qquad \text{(Single-layer)}$$

$$K[\sigma](\boldsymbol{x}) := \text{p.v.} \int_\Gamma \left[\gamma_{1,\boldsymbol{y}} G(\boldsymbol{x},\boldsymbol{y})\right]^T \sigma(\boldsymbol{y}) \mathrm{d}s(\boldsymbol{y}), \qquad \text{(Double-layer)}$$

$$K'[\sigma](\boldsymbol{x}) := \text{p.v.} \int_\Gamma \left(\gamma_{1,\boldsymbol{x}} G(\boldsymbol{x},\boldsymbol{y})\right) \sigma(\boldsymbol{y}) \mathrm{d}s(\boldsymbol{y}), \qquad \text{(Adjoint double-layer)}$$

$$T[\sigma](\boldsymbol{x}) := \text{f.p.} \int_\Gamma \gamma_{1,\boldsymbol{x}} \left[\gamma_{1,\boldsymbol{y}} G(\boldsymbol{x},\boldsymbol{y})\right]^T \sigma(\boldsymbol{y}) \mathrm{d}s(\boldsymbol{y}) \qquad \text{(Hypersingular)}$$

IE methods are **competitive** $\iff$ **accurate** and **fast** evaluation operators.

A word on Nyström discretization

Second-kind integral equation

$$u(\boldsymbol{x}) + \int_{\mathcal{D}} \mathbb{K}(\boldsymbol{x}, \boldsymbol{y}) u(\boldsymbol{y}) \mathrm{d}E_{\boldsymbol{y}} = f(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathcal{D}$$

A word on Nyström discretization

Second-kind integral equation

$$u(\boldsymbol{x}) + \int_{\mathcal{D}} \mathbb{K}(\boldsymbol{x}, \boldsymbol{y}) u(\boldsymbol{y}) \mathrm{d}E_{\boldsymbol{y}} = f(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathcal{D}$$

1. Approximate integral by quadrature $\mathcal{Q} = \{\boldsymbol{x}_j, w_j\}_{j=1}^{N}$:

$$u(\boldsymbol{x}) + \sum_{j=1}^{N} w_j \mathbb{K}(\boldsymbol{x}, \boldsymbol{x}_j) u(\boldsymbol{x}_j) = f(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathcal{D}$$

A word on Nyström discretization

Second-kind integral equation

$$u(\boldsymbol{x}) + \int_{\mathcal{D}} \mathbb{K}(\boldsymbol{x}, \boldsymbol{y}) u(\boldsymbol{y}) \mathrm{d}E_{\boldsymbol{y}} = f(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathcal{D}$$

1. Approximate integral by quadrature $\mathcal{Q} = \{\boldsymbol{x}_j, w_j\}_{j=1}^N$:

$$u(\boldsymbol{x}) + \sum_{j=1}^{N} w_j \mathbb{K}(\boldsymbol{x}, \boldsymbol{x}_j) u(\boldsymbol{x}_j) = f(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathcal{D}$$

2. Impose equality at the quadrature points:

$$u(\boldsymbol{x}_i) + \sum_{j=1}^{N} w_j \mathbb{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) u(\boldsymbol{x}_j) = f(\boldsymbol{x}_i), \quad i = 1, \ldots, N$$

A word on Nyström discretization

Second-kind integral equation

$$u(\boldsymbol{x}) + \int_{\mathcal{D}} \mathbb{K}(\boldsymbol{x}, \boldsymbol{y}) u(\boldsymbol{y}) \mathrm{d}E_{\boldsymbol{y}} = f(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathcal{D}$$

1. Approximate integral by quadrature $\mathcal{Q} = \{\boldsymbol{x}_j, w_j\}_{j=1}^N$:

$$u(\boldsymbol{x}) + \sum_{j=1}^{N} w_j \mathbb{K}(\boldsymbol{x}, \boldsymbol{x}_j) u(\boldsymbol{x}_j) = f(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathcal{D}$$

2. Impose equality at the quadrature points:

$$u(\boldsymbol{x}_i) + \sum_{j=1}^{N} w_j \mathbb{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) u(\boldsymbol{x}_j) = f(\boldsymbol{x}_i), \quad i = 1, \ldots, N$$

3. Solve the resulting **linear system** for $u(\boldsymbol{x}_i), 1 \leq i \leq N$

A word on Nyström discretization

Second-kind integral equation

$$u(\boldsymbol{x}) + \int_{\mathcal{D}} \mathbb{K}(\boldsymbol{x}, \boldsymbol{y}) u(\boldsymbol{y}) \mathrm{d}E_{\boldsymbol{y}} = f(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathcal{D}$$

1. Approximate integral by quadrature $\mathcal{Q} = \{\boldsymbol{x}_j, w_j\}_{j=1}^N$:

$$u(\boldsymbol{x}) + \sum_{j=1}^{N} w_j \mathbb{K}(\boldsymbol{x}, \boldsymbol{x}_j) u(\boldsymbol{x}_j) = f(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathcal{D}$$

2. Impose equality at the quadrature points:

$$u(\boldsymbol{x}_i) + \sum_{j=1}^{N} w_j \mathbb{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) u(\boldsymbol{x}_j) = f(\boldsymbol{x}_i), \quad i = 1, \ldots, N$$

3. Solve the resulting **linear system** for $u(\boldsymbol{x}_i), 1 \leq i \leq N$

Not so straightforward due to singularities in $\mathbb{K}$...

Approximation challenges

$$\int_{\mathcal{D}} \mathbb{K}(\boldsymbol{x}_i, \boldsymbol{y})\varphi(\boldsymbol{y})\mathrm{d}E_{\boldsymbol{y}} \approx \sum_{j=1}^{N} w_j \mathbb{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)\varphi_j$$

Computational challenge

Approximation must be **efficient** and **accurate** for $\boldsymbol{x}_i \in \mathcal{Q} = \{\boldsymbol{x}_j\}_{j=1}^{N}$

Approximation challenges

$$\int_{\mathcal{D}} \mathbb{K}(\boldsymbol{x}_i, \boldsymbol{y})\varphi(\boldsymbol{y})\mathrm{d}E_{\boldsymbol{y}} \approx \sum_{j=1}^{N} w_j \mathbb{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)\varphi_j$$

Computational challenge

Approximation must be **efficient** and **accurate** for $\boldsymbol{x}_i \in \mathcal{Q} = \{\boldsymbol{x}_j\}_{j=1}^{N}$

- Efficient $\Rightarrow$ linear operator needs to be compressed

Approximation challenges

$$\int_{\mathcal{D}} \mathbb{K}(\boldsymbol{x}_i, \boldsymbol{y})\varphi(\boldsymbol{y})\mathrm{d}E_{\boldsymbol{y}} \approx \sum_{j=1}^{N} w_j \mathbb{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)\varphi_j$$

Computational challenge

Approximation must be **efficient** and **accurate** for $\boldsymbol{x}_i \in \mathcal{Q} = \{\boldsymbol{x}_j\}_{j=1}^{N}$

- Efficient $\Rightarrow$ linear operator needs to be compressed
- Accurate $\Rightarrow$ naive quadrature needs to be corrected

compression + correction framework

Computational challenge

Approximation must be **efficient** and **accurate** for $\boldsymbol{x} \in \mathcal{Q} = \{\boldsymbol{x}_j\}_{j=1}^{N}$

$$\int_{\mathcal{D}} \mathbb{K}(\boldsymbol{x}_i, \boldsymbol{y})\varphi(\boldsymbol{y})\mathrm{d}E_{\boldsymbol{y}} \approx \underbrace{\sum_{j=1}^{N} w_j \mathbb{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)\varphi_j}_{\textbf{compression}} + \underbrace{\sum_{j \in \mathcal{I}(i)} \tilde{w}_{i,j}\varphi_j}_{\textbf{correction}}$$

- $\Rightarrow$ **Compression**: FMM, $\mathcal{H}$-matrix, IFGF, $\mathcal{H}^2$-matrix, ...
- $\Rightarrow$ **Correction**: adaptive quadrature, density interpolation, QBX, Chebyshev rectangular-polar, locally corrected trapezoidal rule, ...

Typically, $\mathrm{card}(\mathcal{I}(i)) \ll N$ for $1 \leq i \leq N$, so corrections are sparse.

Compression methods

Compression methods

Provide efficient ways to approximate linear operators of the form

$$\{\varphi_j\}_{j=1}^{N} \mapsto \sum_{j=1}^{N} \mathbb{K}(\boldsymbol{x}_i, \boldsymbol{y}_j)\varphi_j, \quad 1 \leq i \leq M$$

to a **relative precision** ϵ in $\mathcal{O}((M+N)\log^{\gamma}(M+N))$ flops (vs. $\mathcal{O}(MN)$).

- $\mathcal{H}$-matrices
  - $\oplus$ Easy to implement
  - $\oplus$ Fast matrix-vector products
  - $\oplus$ Direct solvers
  - $\ominus$ Memory footprint
- FMM
  - $\oplus$ No “assembly” required
  - $\oplus$ Fast matrix-vector products
  - $\oplus$ Low memory footprint
  - $\ominus$ No direct solvers
  - $\ominus$ Kernel dependent

Compression methods

Compression methods

Provide efficient ways to approximate linear operators of the form

$$\{\varphi_j\}_{j=1}^N \mapsto \sum_{j=1}^{N} \mathbb{K}(\boldsymbol{x}_i, \boldsymbol{y}_j)\varphi_j, \quad 1 \leq i \leq M$$

to a **relative precision** ϵ in $\mathcal{O}((M+N)\log^\gamma(M+N))$ flops (vs. $\mathcal{O}(MN)$).

- $\mathcal{H}$-matrices
  - $\oplus$ Easy to implement
  - $\oplus$ Fast matrix-vector products
  - $\oplus$ Direct solvers
  - $\ominus$ Memory footprint
- FMM
  - $\oplus$ No "assembly" required
  - $\oplus$ Fast matrix-vector products
  - $\oplus$ Low memory footprint
  - $\ominus$ No direct solvers
  - $\ominus$ Kernel dependent

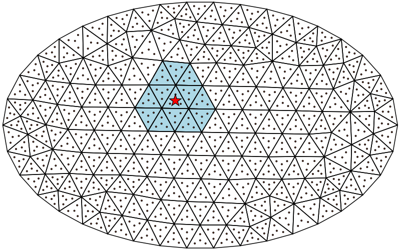
Crucial for the **performance** of integral equation solvers

Singularity handling

Correction methods

Compute **modified weights** $\tilde{w}_{i,j}$ such that

$$\int_{\mathcal{D}} \mathbb{K}(\boldsymbol{x}_i, \boldsymbol{y})\varphi(\boldsymbol{y})\mathrm{d}E_{\boldsymbol{y}} \approx \sum_{j=1}^{N} w_j \mathbb{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)\varphi_j + \sum_{j \in \mathcal{I}(i)} \tilde{w}_{i,j}\varphi_j$$

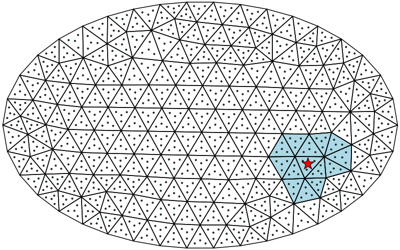


Singularity handling

Correction methods

Compute **modified weights** $\tilde{w}_{i,j}$ such that

$$\int_{\mathcal{D}} \mathbb{K}(\boldsymbol{x}_i, \boldsymbol{y})\varphi(\boldsymbol{y})\mathrm{d}E_{\boldsymbol{y}} \approx \sum_{j=1}^{N} w_j \mathbb{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)\varphi_j + \sum_{j \in \mathcal{I}(i)} \tilde{w}_{i,j}\varphi_j$$

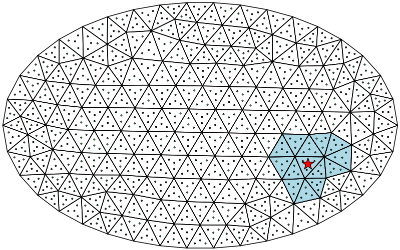


Singularity handling

Correction methods

Compute **modified weights** $\tilde{w}_{i,j}$ such that

$$\int_{\mathcal{D}} \mathbb{K}(\boldsymbol{x}_i, \boldsymbol{y})\varphi(\boldsymbol{y})\mathrm{d}E_{\boldsymbol{y}} \approx \sum_{j=1}^{N} w_j \mathbb{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)\varphi_j + \sum_{j\in\mathcal{I}(i)} \tilde{w}_{i,j}\varphi_j$$



- Crucial for the **accuracy** of integral equation solvers.
- Can often be recast in a **sparse matrix** format (e.g. CSC, CSR)

Take away message

$$\int_{\mathcal{D}} \mathbb{K}(\boldsymbol{x}_i, \boldsymbol{y})\varphi(\boldsymbol{y})\mathrm{d}E_{\boldsymbol{y}} \approx \underbrace{\sum_{j=1}^{N} w_j \mathbb{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)\varphi_j}_{\textbf{compression}} + \underbrace{\sum_{j\in\mathcal{I}(i)} \tilde{w}_{i,j}\varphi_j}_{\textbf{correction}}$$

Integrating compression is (slightly?) more involved than “black-box”...

Take away message

$$\int_{\mathcal{D}} \mathbb{K}(\boldsymbol{x}_i, \boldsymbol{y})\varphi(\boldsymbol{y})\mathrm{d}E_{\boldsymbol{y}} \approx \underbrace{\sum_{j=1}^{N} w_j \mathbb{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)\varphi_j}_{\textbf{compression}} + \underbrace{\sum_{j \in \mathcal{I}(i)} \tilde{w}_{i,j}\varphi_j}_{\textbf{correction}}$$

Integrating compression is (slightly?) more involved than "black-box"...

Why my own $\mathcal{H}$-matrix package?

- Add sparse correction, extract block diagonal, lazily composition, ...

Take away message

$$\int_{\mathcal{D}} \mathbb{K}(\boldsymbol{x}_i, \boldsymbol{y})\varphi(\boldsymbol{y})\mathrm{d}E_{\boldsymbol{y}} \approx \underbrace{\sum_{j=1}^{N} w_j \mathbb{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)\varphi_j}_{\textbf{compression}} + \underbrace{\sum_{j\in\mathcal{I}(i)} \tilde{w}_{i,j}\varphi_j}_{\textbf{correction}}$$

Integrating compression is (slightly?) more involved than "black-box"...

Why my own $\mathcal{H}$-matrix package?

- Add sparse correction, extract block diagonal, lazily composition, ...
- Ability to easily control *admissibility* criteria (kernel/mesh dependent)

Take away message

$$\int_{\mathcal{D}} \mathbb{K}(\boldsymbol{x}_i, \boldsymbol{y})\varphi(\boldsymbol{y})\mathrm{d}E_{\boldsymbol{y}} \approx \underbrace{\sum_{j=1}^{N} w_j \mathbb{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)\varphi_j}_{\textbf{compression}} + \underbrace{\sum_{j \in \mathcal{I}(i)} \tilde{w}_{i,j}\varphi_j}_{\textbf{correction}}$$

Integrating compression is (slightly?) more involved than “black-box”...

Why my own $\mathcal{H}$-matrix package?

- Add sparse correction, extract block diagonal, lazily composition, ...
- Ability to easily control *admissibility* criteria (kernel/mesh dependent)
- Support for tensor-valued kernels (e.g. elasticity)

Take away message

$$\int_{\mathcal{D}} \mathbb{K}(\boldsymbol{x}_i, \boldsymbol{y})\varphi(\boldsymbol{y})\mathrm{d}E_{\boldsymbol{y}} \approx \underbrace{\sum_{j=1}^{N} w_j \mathbb{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)\varphi_j}_{\textbf{compression}} + \underbrace{\sum_{j \in \mathcal{I}(i)} \tilde{w}_{i,j}\varphi_j}_{\textbf{correction}}$$

Integrating compression is (slightly?) more involved than "black-box"...

Why my own $\mathcal{H}$-matrix package?

- Add sparse correction, extract block diagonal, lazily composition, ...
- Ability to easily control *admissibility* criteria (kernel/mesh dependent)
- Support for tensor-valued kernels (e.g. elasticity)
- Julia support ☺

Take away message

$$\int_{\mathcal{D}} \mathbb{K}(\boldsymbol{x}_i, \boldsymbol{y})\varphi(\boldsymbol{y})\mathrm{d}E_{\boldsymbol{y}} \approx \underbrace{\sum_{j=1}^{N} w_j \mathbb{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)\varphi_j}_{\textbf{compression}} + \underbrace{\sum_{j \in \mathcal{I}(i)} \tilde{w}_{i,j}\varphi_j}_{\textbf{correction}}$$

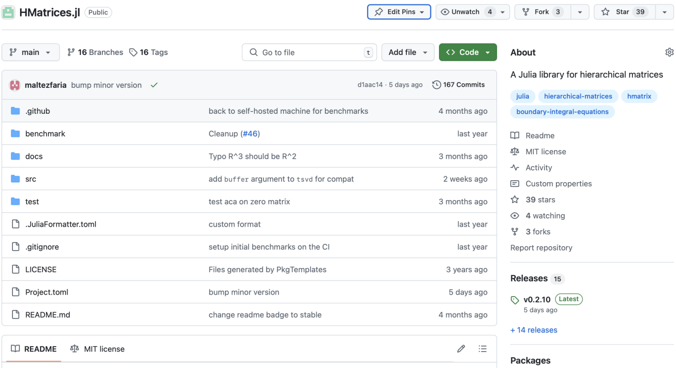
Integrating compression is (slightly?) more involved than "black-box"...

Why my own $\mathcal{H}$-matrix package?

- Add sparse correction, extract block diagonal, lazily composition, ...
- Ability to easily control *admissibility* criteria (kernel/mesh dependent)
- Support for tensor-valued kernels (e.g. elasticity)
- Julia support ☺

Would be happy to wrap a better/faster $\mathcal{H}$-matrix package!

HMatrices.jl



- ⊕ Tensor-valued kernels
- ⊕ "Easy" to bypass defaults
- ⊕ Parallel assembly, mat/vec
- ⊕ Cholesky/LU factorization

- ⊖ My first (Julia) library!
- ⊖ No effort at HPC
- ⊖ Poor parallelization
- ⊖ Minimal support these days

Final thoughts

- $\mathcal{H}$-matrices are central in IE methods

Final thoughts

- $\mathcal{H}$-matrices are central in IE methods
- Black-box approach has (surmountable?) limitations

Final thoughts

- $\mathcal{H}$-matrices are central in IE methods
- Black-box approach has (surmountable?) limitations
- Helpful to increase communication between users, developers, and researchers

Final thoughts

- $\mathcal{H}$-matrices are central in IE methods
- Black-box approach has (surmountable?) limitations
- Helpful to increase communication between users, developers, and researchers
- Too many libraries? Lack of standardized benchmarks? Join efforts?