SIAM CSE: Minitutorial 4: Fast direct solvers for Elliptic PDEs Part II

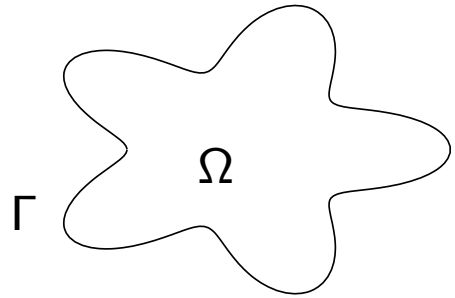
Adrianna Gillman, University of Colorado, Boulder
Per-Gunnar Martinsson, University of Texas, Austin

Download codes and slide: https://github.com/agillman20/SIAMCSE2025

Outline

1. Integral equations
2. Fast direct solver classification
3. Illustration of an inversion technique
4. Techniques for creating low rank factorization
5. Results for an integral equation
6. Extensions
7. Some available software

Model problem



Consider the problem

$$-\Delta u(\mathbf{x}) = 0, \qquad \mathbf{x} \in \Omega,$$
$$u(\mathbf{x}) = g(\mathbf{x}), \quad \mathbf{x} \in \Gamma.$$

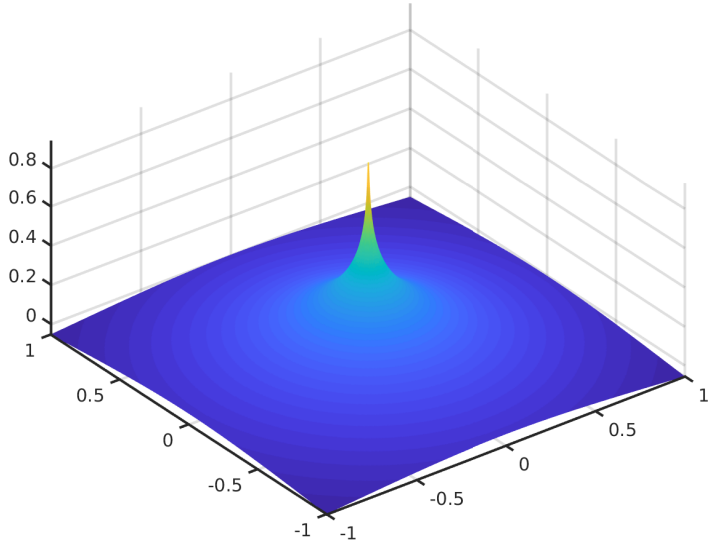
The union of physics and mathematics

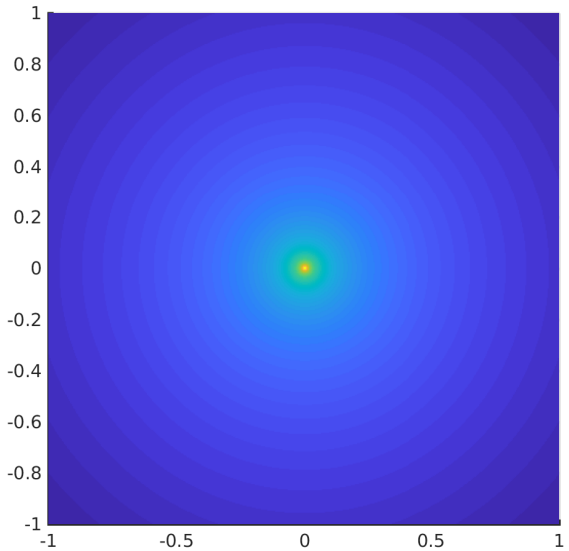
For a given point charge $\mathbf{x}_0 \in \mathbb{R}^2$, the solution of

$$-\Delta u(\mathbf{x}) = \delta(\mathbf{x} - \mathbf{x}_0), \quad \mathbf{x} \in \mathbb{R}^2$$

is

$$u(\mathbf{x}) = -\frac{1}{2\pi} \log |\mathbf{x} - \mathbf{x}_0|.$$





The union of physics and mathematics

For a given point charge $\mathbf{x}_0 \in \mathbb{R}^2$, the solution of

$$-\Delta u(\mathbf{x}) = \delta(\mathbf{x} - \mathbf{x}_0), \quad \mathbf{x} \in \mathbb{R}^2$$

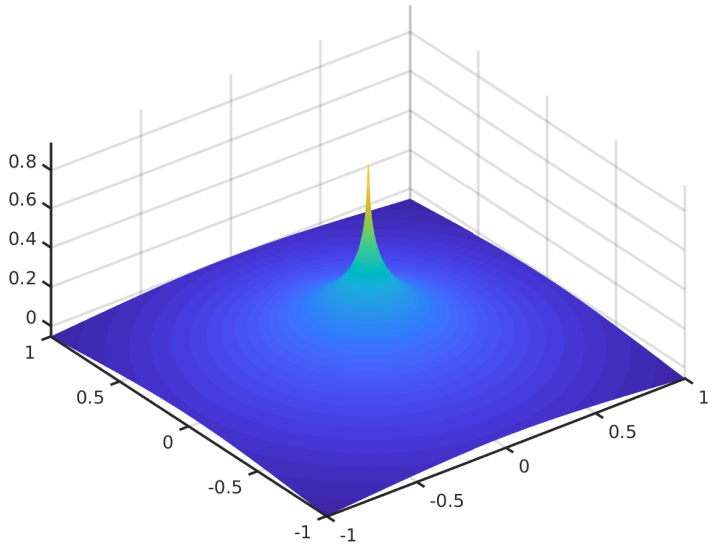
is

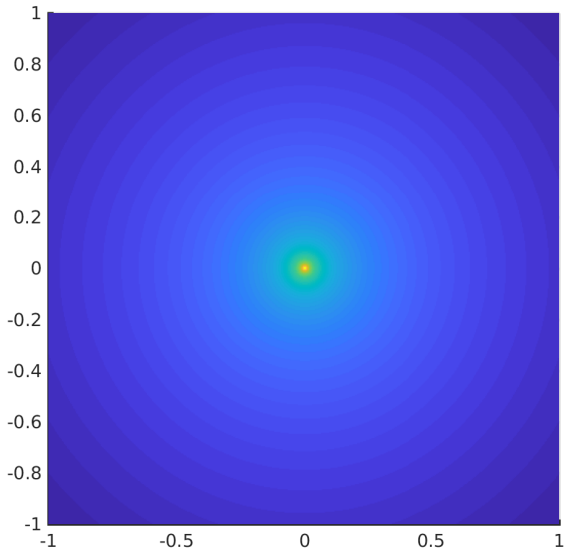
$$u(\mathbf{x}) = -\frac{1}{2\pi} \log |\mathbf{x} - \mathbf{x}_0|.$$

The fundamental solution $G(\mathbf{x}, \mathbf{y})$ is given by

$$G(\mathbf{x}, \mathbf{y}) = -\frac{1}{2\pi} \log |\mathbf{x} - \mathbf{y}|.$$

This allows us to move the point charge around.





The union of physics and mathematics

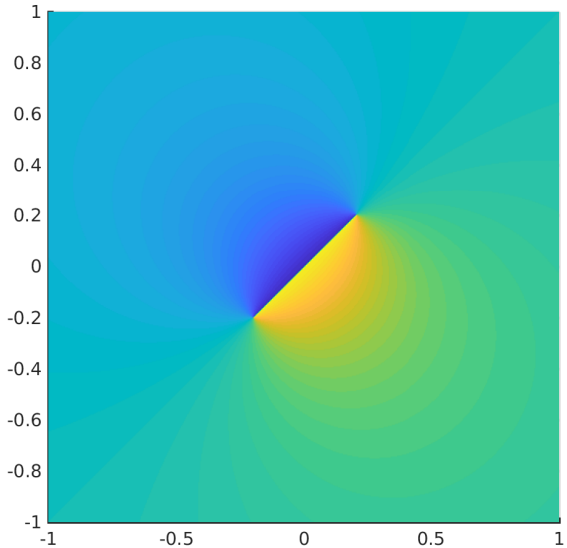
The double layer kernel

For a point $\mathbf{y}$ on the boundary of a curve, the double layer kernel

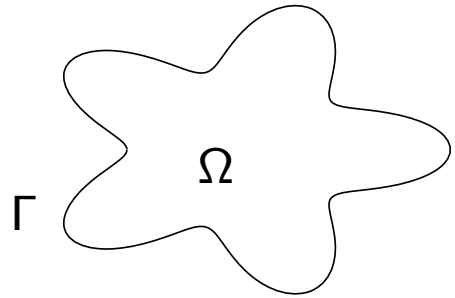
$$D(\mathbf{x}, \mathbf{y}) = \partial_{\nu_\mathbf{y}} G(\mathbf{x}, \mathbf{y})$$

is a solution of

$$-\Delta_\mathbf{x} u(\mathbf{x}) = \delta(\mathbf{x} - \mathbf{y}), \qquad \mathbf{x} \in \mathbb{R}^2.$$



Model problem



Consider the problem

$$\begin{aligned} -\Delta u(\mathbf{x}) &= 0, \qquad \mathbf{x} \in \Omega, \\ u(\mathbf{x}) &= g(\mathbf{x}), \quad \mathbf{x} \in \Gamma. \end{aligned}$$

The solution can be represented as a double layer potential

$$u(\mathbf{x}) = \int_\Gamma \partial_{\nu_\mathbf{y}} G(\mathbf{x}, \mathbf{y}) \phi(\mathbf{y}) ds(\mathbf{y}), \qquad \mathbf{x} \in \Omega,$$

where $\nu_\mathbf{y}$ is the outward normal at $\mathbf{y}$ and $G(\mathbf{x}, \mathbf{y})$ is the fundamental solution

$$G(\mathbf{x}, \mathbf{y}) = -\frac{1}{2\pi} \log |\mathbf{x} - \mathbf{y}|.$$

Then the boundary charge distribution ϕ satisfies the boundary integral equation

$$\frac{1}{2}\phi(\mathbf{x}) + \int_\Gamma \partial_{\nu_\mathbf{y}} G(\mathbf{x}, \mathbf{y}) \phi(\mathbf{y}) ds(\mathbf{y}) = g(\mathbf{x})$$

How do you discretize integral equations?

To discretize the integral equation using a Nyström method, pick an appropriate quadrature to approximate the integral. Then

$$g(\mathbf{x}) = \frac{1}{2}\phi(\mathbf{x}) + \int_\Gamma \partial_{\nu_\mathbf{y}} G(\mathbf{x},\mathbf{y})\phi(\mathbf{y})ds(\mathbf{y})$$
$$\sim \frac{1}{2}\phi(\mathbf{x}) + \sum_{j=1}^{N} \partial_{\nu_{\mathbf{x}_j}} G(\mathbf{x},\mathbf{x}_j)\phi(\mathbf{x}_j)w_j$$

Looking for the solution at the quadrature nodes and forcing the approximation to hold at these locations leads to a linear system where the i^{th} row is given by

$$g(\mathbf{x}_i) = \frac{1}{2}\phi(\mathbf{x}_i) + \sum_{j=1}^{N} \partial_{\nu_{\mathbf{x}_j}} G(\mathbf{x}_i,\mathbf{x}_j)\phi(\mathbf{x}_j)w_j$$

Model problem

Upon discretization, we have to solve a linear system of the form

$$\mathbf{A}\phi = (\frac{1}{2}\mathbf{I} + \mathbf{D})\phi = \mathbf{g},$$

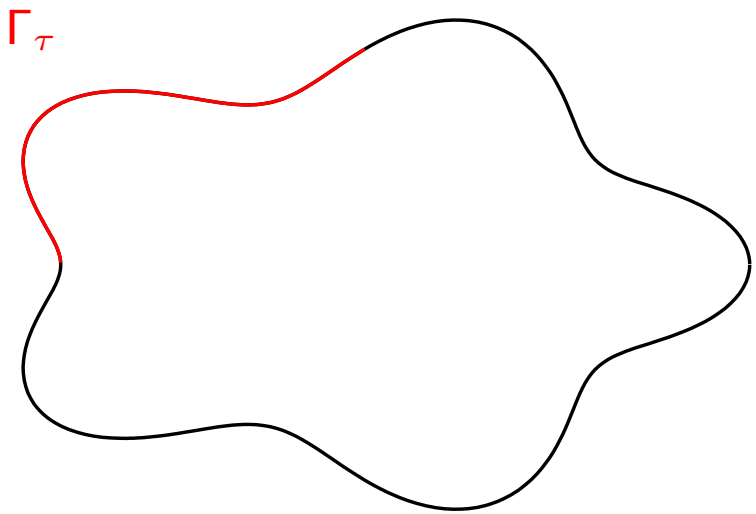
where $\mathbf{D}$ is a matrix that approximates the integral operator

$$\int_\Gamma \partial_{\nu_\mathbf{y}} G(\mathbf{x}, \mathbf{y})\phi(\mathbf{y}) ds(\mathbf{y}).$$

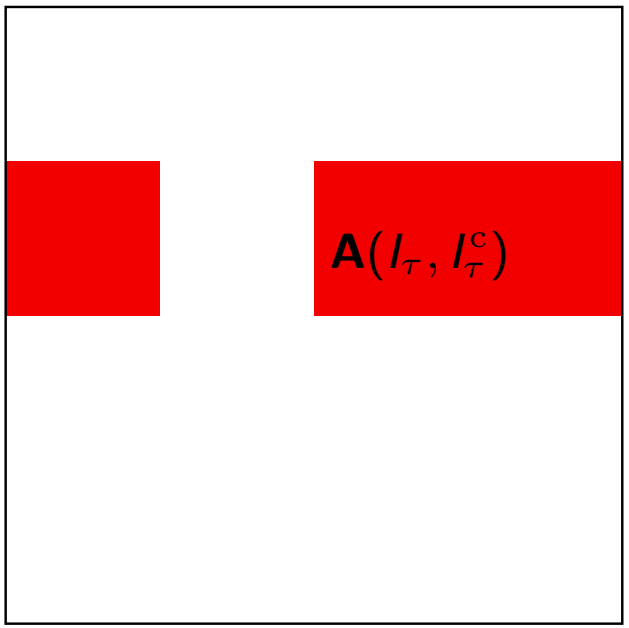
Properties of $\mathbf{A}$:

- Dense matrix.
- Size is determined by the number of discretization points.

Is the BIE data sparse?



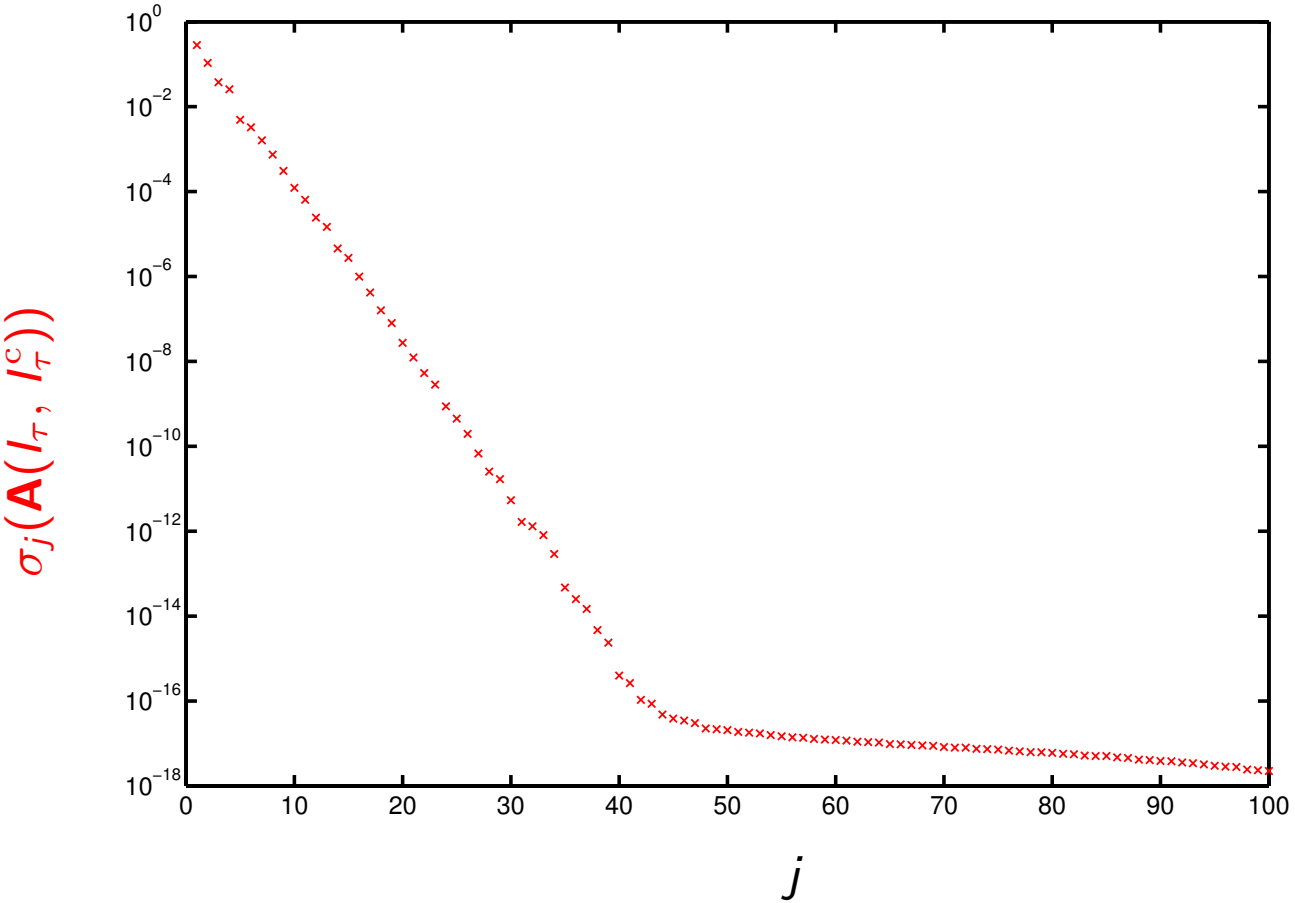
The contour Γ.



The matrix $\mathbf{A}$.

Is the BIE data sparse?

Singular values of $\mathbf{A}(I_\tau, I_\tau^c)$



To precision 10^{-10}, the matrix $\mathbf{A}(I_\tau, I_\tau^c)$ has rank 29.

Outline

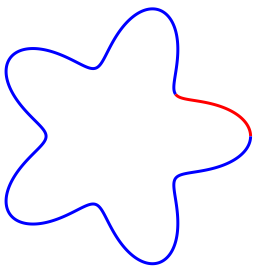
1. Integral equations
2. Fast solver classificiation
3. Illustration of an inversion technique
4. Techniques for creating low rank factorizations
5. Results for an integral equation
6. Extensions
7. Some available software

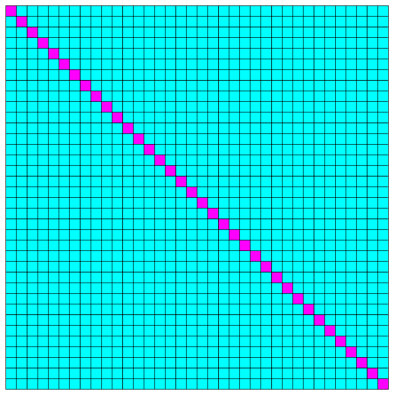
You have options:

- **Admissibility:** Weak vs strong
- **Tree structuure:** Hierarchical vs Flat
- **Factorization bases:** general vs nested
- **Techniques for constructing the factors:** Classic vs accelerated
- **Accelerating factorization:** Dense linear vs physics

Strong Vs Weak Admissibility

Weak admissibility: Compress interactions up to adjacent intervals

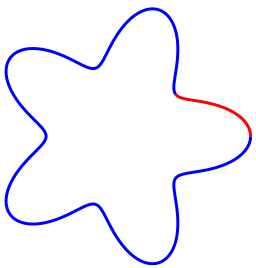


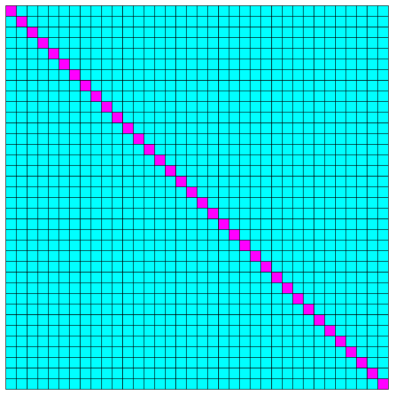


Interaction matrix has rank is 22 for $\varepsilon = 10^{-8}$

Strong Vs Weak Admissibility

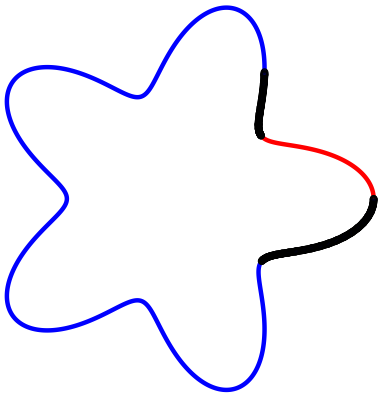
Weak admissibility: Compress interactions up to adjacent intervals

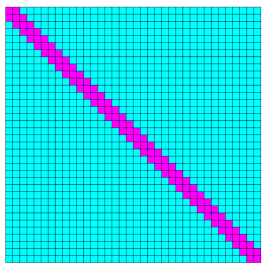




Interaction matrix has rank is 22 for $\varepsilon = 10^{-8}$

Strong admissibility: Compress interactions that are "far" away

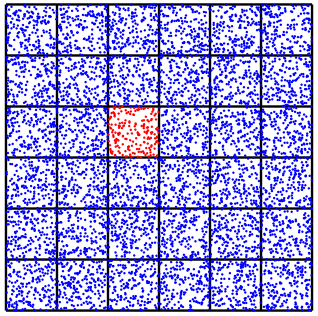




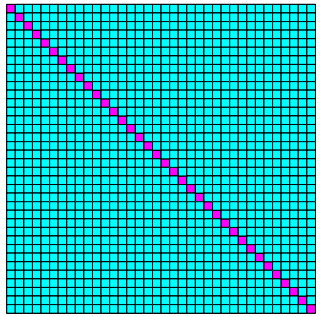
Interaction matrix has rank is 15 for $\varepsilon = 10^{-8}$

Strong Vs Weak Admissibility

Weak admissibility: Compress directly adjacent patches.



Points in a box $\Omega = [0,1]^2$.



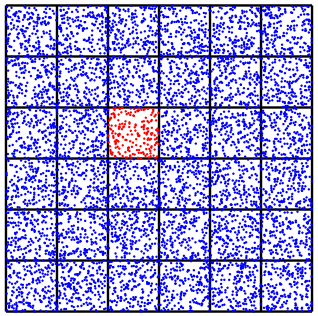
Matrix illustration.

Red sources induce potentials on blue points. Average rank=13.9 at $\varepsilon = 10^{-8}$.

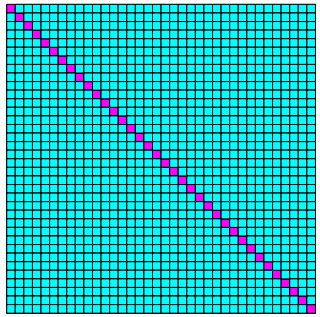
Magenta blocks are dense. Cyan blocks low rank. Many low rank blocks, but high ranks.

Strong Vs Weak Admissibility

Weak admissibility: Compress directly adjacent patches.



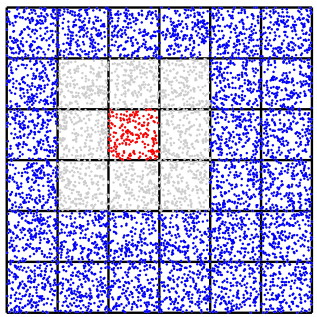
Points in a box $\Omega = [0,1]^2$.



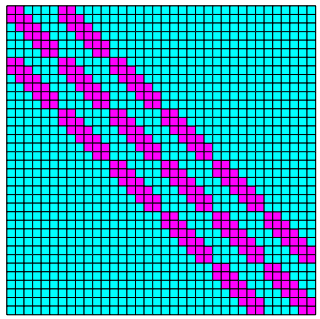
Matrix illustration.

Red sources induce potentials on blue points. Average rank=13.9 at $\varepsilon = 10^{-8}$.

Strong admissibility: Compress only "far-field" interactions.



Points in a box $\Omega = [0,1]^2$.



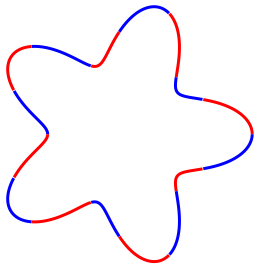
Matrix Illustration

Red sources induce potentials on blue points. Average rank=7.7 at $\varepsilon = 10^{-8}$.

Magenta blocks are dense. Cyan blocks low rank. Many low rank blocks, but high ranks.

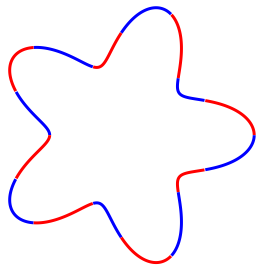
Flat vs Hierarchical

One level: Use a single tessellation of the domain

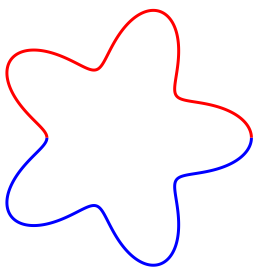


Flat vs Hierarchical

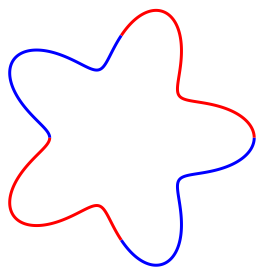
One level: Use a single tessellation of the domain



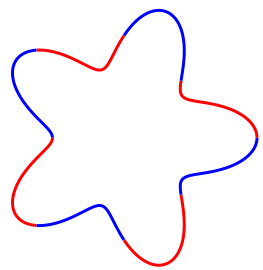
Hierarchical: Use a hierarchical tesselation



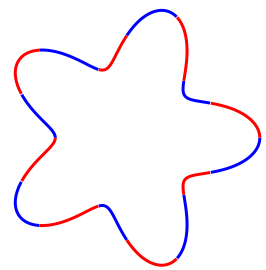
Level 1



Level 2



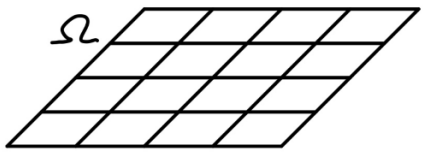
Level 3



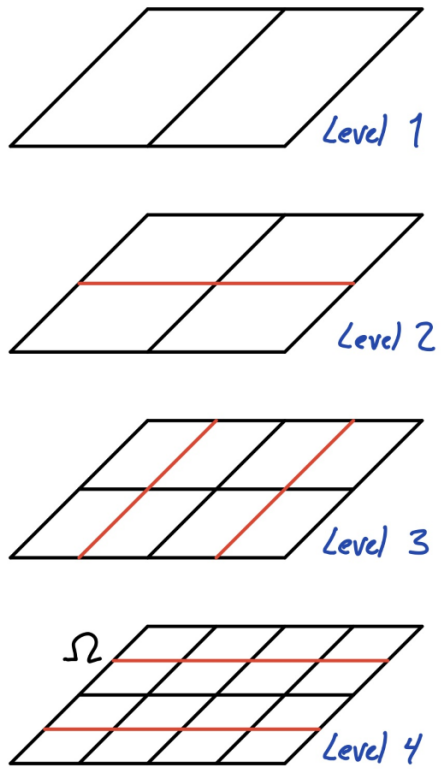
Level 4

Flat vs Hierarchical

Flat tessellation



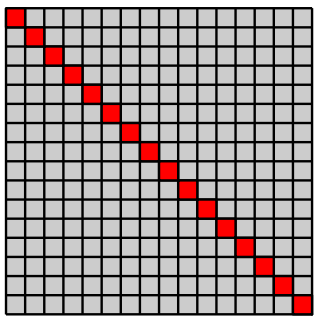
Hierarchical tessellation



Flat vs Hierarchical

Corresponding Matrix Format

One level



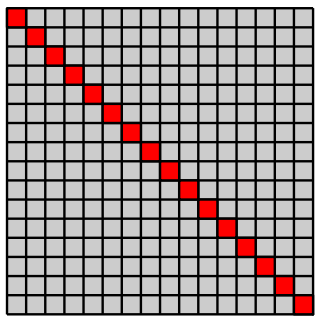
Pros: Easy to code and parallelize.

Cons: Not optimally scaling.

Flat vs Hierarchical

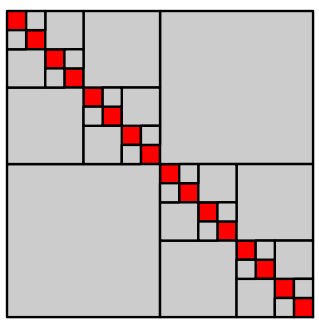
Corresponding Matrix Format

One level



Pros: Easy to code and parallelize.
Cons: Not optimally scaling.

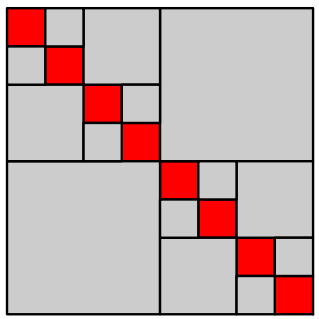
Hierarchical tessellation



Pros: Better asymptotic complexity (can be linear).
Cons: More complicated to code and analyze. Can be difficult to parallelize.

General vs Nested Bases

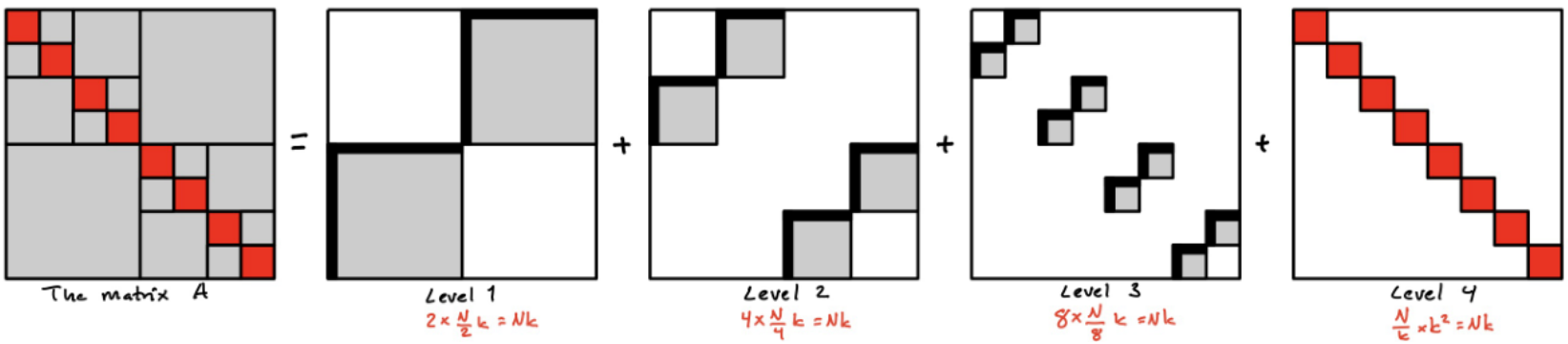
General bases: Let us consider a basic rank structured matrix:



Question: How much storage is required?

General vs Nested Bases

Observe that you can view the matrix as a *sum* over different "levels":



Let k denote the rank of the off-diagonal blocks.

At each level, the cost to store the factors is $\sim Nk$.

There are $\sim \log(N)$ levels, so total storage $\sim kN \log(N)$.

General vs Nested Bases

Nested bases: These were introduced to eliminate log-factors, and improve efficiency.

The idea is to define the low rank factors for the off-diagonal blocks *recursively* — the bases on one level are defined in terms of the bases on the next finer level.

Formally, this leads to a *multiplicative* representation, rather than an *additive* one.

For instance, it could take the form

$$\mathbf{A} = \mathbf{U}^{(3)}\big(\mathbf{U}^{(2)}\big(\mathbf{U}^{(1)}\,\mathbf{B}^{(0)}\,(\mathbf{V}^{(1)})^* + \mathbf{B}^{(1)}\big)(\mathbf{V}^{(2)})^* + \mathbf{B}^{(2)}\big)(\mathbf{V}^{(3)})^* + \mathbf{D}^{(3)}$$

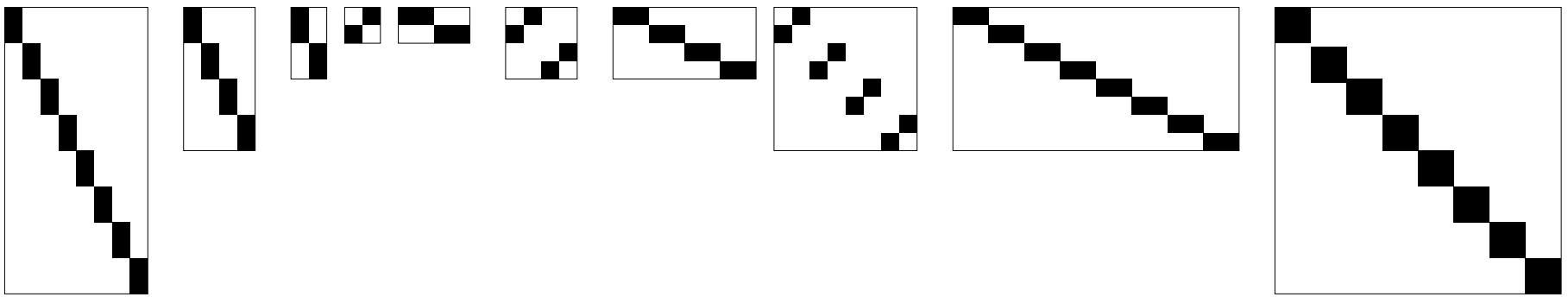
General vs Nested Bases

When the factorization takes the form

$$\mathbf{A} = \mathbf{U}^{(3)}\big(\mathbf{U}^{(2)}\big(\mathbf{U}^{(1)}\,\mathbf{B}^{(0)}\,(\mathbf{V}^{(1)})^*\big) + \mathbf{B}^{(1)}\big)(\mathbf{V}^{(2)})^* + \mathbf{B}^{(2)}\big)(\mathbf{V}^{(3)})^* + \mathbf{D}^{(3)},$$

the blocks have the following sparsity pattern:

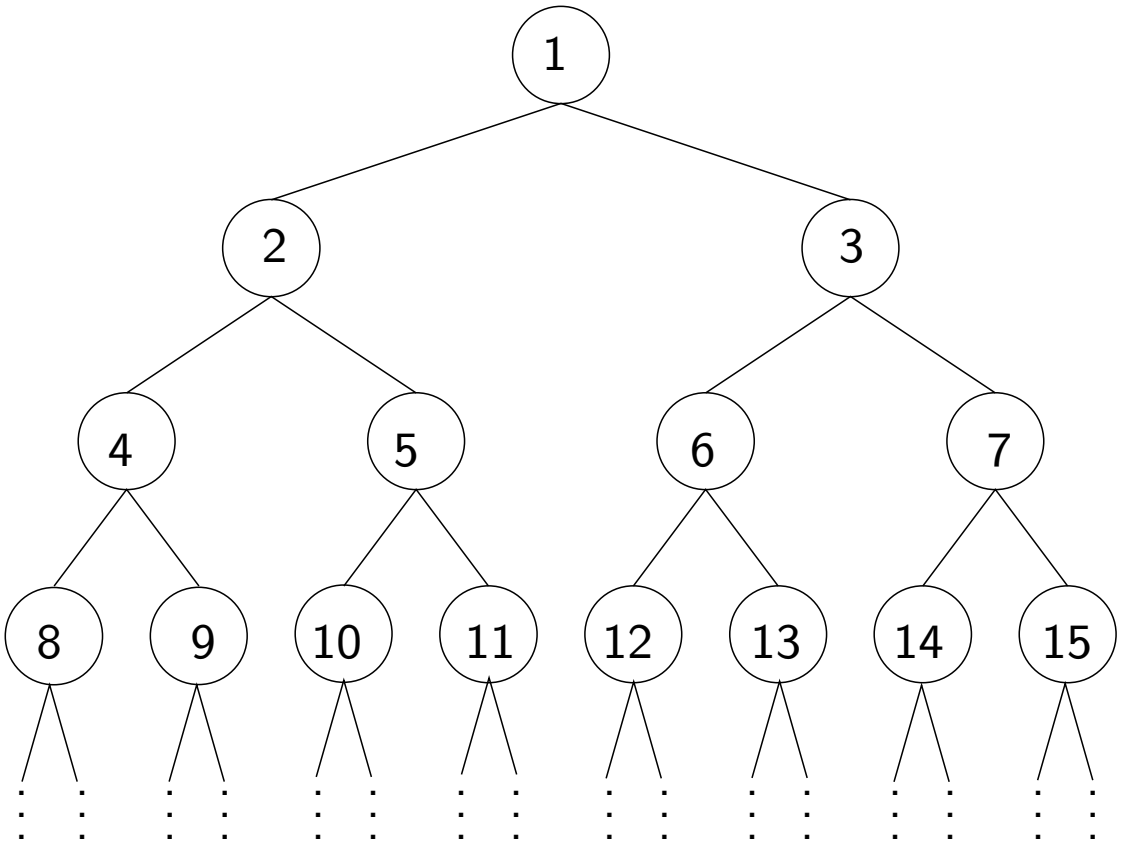
$\mathbf{U}^{(3)}$ $\mathbf{U}^{(2)}$ $\mathbf{U}^{(1)}$ $\mathbf{B}^{(0)}$ $(\mathbf{V}^{(1)})^*$ $\mathbf{B}^{(1)}$ $(\mathbf{V}^{(2)})^*$ $\mathbf{B}^{(2)}$ $(\mathbf{V}^{(3)})^*$ $\mathbf{D}^{(3)}$



The cost to store level ℓ is now $2^{-\ell}Nk$ $\rightarrow$ geometric sum and $O(kN)$ total storage.

Note: The classical Fast Multipole Method relies on nested bases.
This is in contrast to Barnes-Hut which (implicitly) uses general bases.

Binary tree for nested bases



Information is passed up and down the tree

Versions of fast direct solvers

We can now loosely organize some common rank-structured matrix "formats":

Admissibility	Flat
Weak	Block Separable
Strong	Block Low Rank

Versions of fast direct solvers

We can now loosely organize some common rank-structured matrix "formats":

Admissibility	Flat
Weak	Block Separable
Strong	Block Low Rank

Admissibility	Hierarchical	
	General	Nested bases
Weak	HODLR	HBS/HSS, recursive skeletonization
Strong	$\mathcal{H}$-matrices; Barnes-Hut	$\mathcal{H}^2$-matrices; FMM; strong recursive skeletonization

Note: *In principle, the term "$\mathcal{H}$-matrix" is extremely broad — every other format is a special case.*

However, the table reflects the standard usage of the term.

Versions of fast direct solvers

We can now loosely organize some common rank-structured matrix "formats":

Admissibility	Flat
Weak	Block Separable
Strong	Block Low Rank

$\Leftarrow$ easier to implement

$\Downarrow$ decrease flops

Admissibility	General	Hierarchical Nested bases
Weak	HODLR	HBS/HSS, recursive skeletonization
Strong	$\mathcal{H}$-matrices; Barnes-Hut	$\mathcal{H}^2$-matrices; FMM; strong recursive skeletonization

Note: *In principle, the term "$\mathcal{H}$-matrix" is extremely broad — every other format is a special case.*

However, the table reflects the standard usage of the term.

Versions of fast direct solvers: selection of references

- *$\mathcal{H}$- and $\mathcal{H}^2$-matrices:* Hackbusch (1999); Khoromskij, Hackbusch (2000); Börm, Grasedyck, Hackbusch (2003); ...
- *Recursive skeletonization:* Lee, Greengard, (1992); Starr, Rokhlin (1994); Michielssen, Boag, Chew (1996); Martinsson, Rokhlin (2005); Greengard, Gueyffier, Martinsson, Rokhlin (2009); Ho, Greengard (2012);Gillman, Young, Martinsson (2012); Ho, Ying (2016); ...
- *HSS matrices:* Xia, Chandrasekaran, Gu, and Li (2009); Xia (2012); Wang, Li, Xia, Situ, De Hoop (2013); Xi, Xia (2016); ...
- *Hierarchically off-diagonal low rank (HODLR) matrices:* Aminfara, Ambikasaran, Darve (2016); Massei, Robol, Kressner (2020); ...
- *Block low rank (BLR) matrices:* Amestoy, Ashcraft, Boiteau, Buttari, l'Excellent, Weisbecker (2015); Amestoy, Buttari, l'Excellent, Mary (2017); ...

Survey: Ballani & Kressner (2016). *Forthcoming: Martinsson & O'Neil (2025)*

Monographs: Bebendorf (2008). Börm (2010). Martinsson (2019).

Inversion with weak admissibility

Some quick facts:

- All off-diagonal blocks are compressed into low rank form.
- Ranks are higher than in strong admissibility.
- When off-diagonal blocks are *truely* low rank, the inversion is exact.
- For problems in 1D and 2D, weak admissibility often works great.
- For 3D problems, weak admissibility often works fine for surface BIEs.
- In more general geometries, ranks can get quite large, and you may want to swap to strong admissibility.

Formats based on weak admissibility include: HODLR, HSS, hierarchically block separable, etc.

Outline

1. Integral equations
2. Fast solver classificiation
3. Illustration of an inversion technique
4. Techniques for creating low rank factorizations
5. Results for an integral equation
6. Extensions
7. Some available software

Intuition for inversion of Hierarchically Block-Separable matrices

$$\text{We get } \mathbf{A} = \begin{bmatrix} \mathbf{D}_{11} & \mathbf{U}_1 \tilde{\mathbf{A}}_{12} \mathbf{V}_2^* & \mathbf{U}_1 \tilde{\mathbf{A}}_{13} \mathbf{V}_3^* & \mathbf{U}_1 \tilde{\mathbf{A}}_{14} \mathbf{V}_4^* \\ \mathbf{U}_2 \tilde{\mathbf{A}}_{21} \mathbf{V}_1^* & \mathbf{D}_{22} & \mathbf{U}_2 \tilde{\mathbf{A}}_{23} \mathbf{V}_3^* & \mathbf{U}_2 \tilde{\mathbf{A}}_{24} \mathbf{V}_4^* \\ \mathbf{U}_3 \tilde{\mathbf{A}}_{31} \mathbf{V}_1^* & \mathbf{U}_3 \tilde{\mathbf{A}}_{32} \mathbf{V}_2^* & \mathbf{D}_{33} & \mathbf{U}_3 \tilde{\mathbf{A}}_{34} \mathbf{V}_4^* \\ \mathbf{U}_4 \tilde{\mathbf{A}}_{41} \mathbf{V}_1^* & \mathbf{U}_4 \tilde{\mathbf{A}}_{42} \mathbf{V}_2^* & \mathbf{U}_4 \tilde{\mathbf{A}}_{43} \mathbf{V}_3^* & \mathbf{D}_{44} \end{bmatrix}.$$

Then $\mathbf{A}$ admits the factorization:

$$\mathbf{A} = \underbrace{\begin{bmatrix} \mathbf{U}_1 & & & \\ & \mathbf{U}_2 & & \\ & & \mathbf{U}_3 & \\ & & & \mathbf{U}_4 \end{bmatrix}}_{=\mathbf{U}} \underbrace{\begin{bmatrix} \mathbf{0} & \tilde{\mathbf{A}}_{12} & \tilde{\mathbf{A}}_{13} & \tilde{\mathbf{A}}_{14} \\ \tilde{\mathbf{A}}_{21} & \mathbf{0} & \tilde{\mathbf{A}}_{23} & \tilde{\mathbf{A}}_{24} \\ \tilde{\mathbf{A}}_{31} & \tilde{\mathbf{A}}_{32} & \mathbf{0} & \tilde{\mathbf{A}}_{34} \\ \tilde{\mathbf{A}}_{41} & \tilde{\mathbf{A}}_{42} & \tilde{\mathbf{A}}_{43} & \mathbf{0} \end{bmatrix}}_{=\tilde{\mathbf{A}}} \underbrace{\begin{bmatrix} \mathbf{V}_1^* & & & \\ & \mathbf{V}_2^* & & \\ & & \mathbf{V}_3^* & \\ & & & \mathbf{V}_4^* \end{bmatrix}}_{=\mathbf{V}^*} +$$

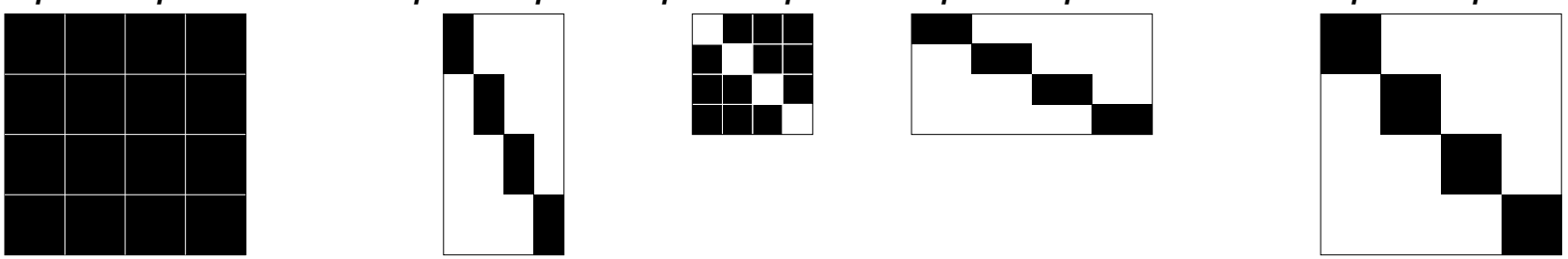
$$\underbrace{\begin{bmatrix} \mathbf{D}_{11} & & & \\ & \mathbf{D}_{22} & & \\ & & \mathbf{D}_{33} & \\ & & & \mathbf{D}_{44} \end{bmatrix}}_{=\mathbf{D}},$$

Intuition for inversion of Hierarchically Block-Separable matrices

We get $\mathbf{A} = \begin{bmatrix} \mathbf{D}_{11} & \mathbf{U}_1 \tilde{\mathbf{A}}_{12} \mathbf{V}_2^* & \mathbf{U}_1 \tilde{\mathbf{A}}_{13} \mathbf{V}_3^* & \mathbf{U}_1 \tilde{\mathbf{A}}_{14} \mathbf{V}_4^* \\ \mathbf{U}_2 \tilde{\mathbf{A}}_{21} \mathbf{V}_1^* & \mathbf{D}_{22} & \mathbf{U}_2 \tilde{\mathbf{A}}_{23} \mathbf{V}_3^* & \mathbf{U}_2 \tilde{\mathbf{A}}_{24} \mathbf{V}_4^* \\ \mathbf{U}_3 \tilde{\mathbf{A}}_{31} \mathbf{V}_1^* & \mathbf{U}_3 \tilde{\mathbf{A}}_{32} \mathbf{V}_2^* & \mathbf{D}_{33} & \mathbf{U}_3 \tilde{\mathbf{A}}_{34} \mathbf{V}_4^* \\ \mathbf{U}_4 \tilde{\mathbf{A}}_{41} \mathbf{V}_1^* & \mathbf{U}_4 \tilde{\mathbf{A}}_{42} \mathbf{V}_2^* & \mathbf{U}_4 \tilde{\mathbf{A}}_{43} \mathbf{V}_3^* & \mathbf{D}_{44} \end{bmatrix}$.

Then $\mathbf{A}$ admits the factorization:

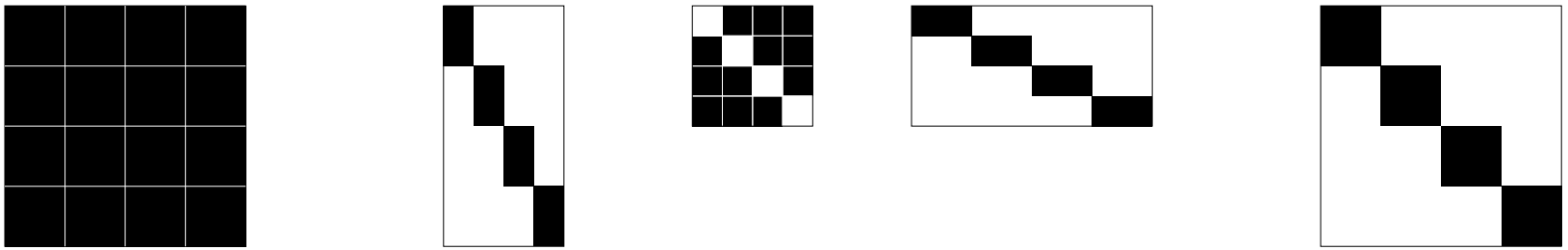
$\mathbf{A}$ ($pn \times pn$) $=$ $\mathbf{U}$ ($pn \times pk$) $\tilde{\mathbf{A}}$ ($pk \times pk$) $\mathbf{V}^*$ ($pk \times pn$) $+$ $\mathbf{D}$, ($pn \times pn$)



General vs Nested Bases

$\mathbf{A}$ $=$ $\mathbf{U}$ $\tilde{\mathbf{A}}$ $\mathbf{V}^*$ $+$ $\mathbf{D}$,

$p\,n \times p\,n$ $p\,n \times p\,k$ $p\,k \times p\,k$ $p\,k \times p\,n$ $p\,n \times p\,n$



Cost of applying $\mathbf{D}$: $O(p^2 n)$

Cost of applying $\mathbf{V}^*$: $O(kpn)$

Cost of applying $\tilde{\mathbf{A}}$: $O\left((kp)^2\right)$

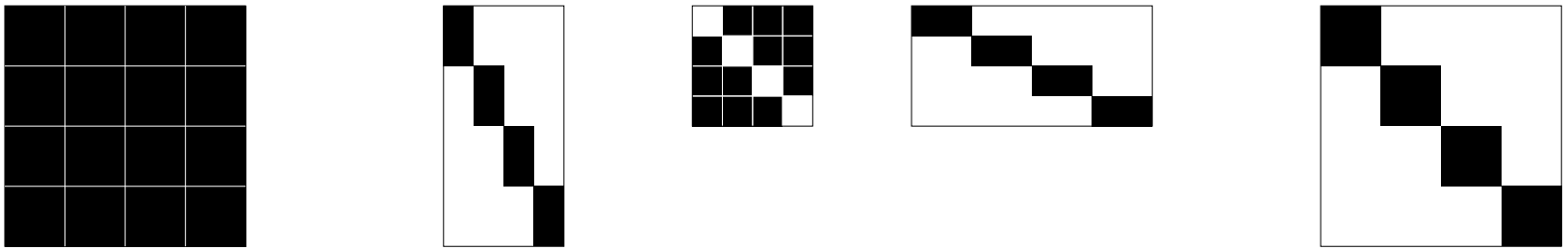
Cost of applying $\tilde{\mathbf{U}}$: $O(kpn)$

Recall: $p = N/n$

General vs Nested Bases

$\mathbf{A}$ $=$ $\mathbf{U}$ $\tilde{\mathbf{A}}$ $\mathbf{V}^*$ $+$ $\mathbf{D}$,

$p\,n \times p\,n$ $p\,n \times p\,k$ $p\,k \times p\,k$ $p\,k \times p\,n$ $p\,n \times p\,n$



Cost of applying $\mathbf{D}$: $O(p^2 n)$

Cost of applying $\mathbf{V}^*$: $O(kpn)$

Cost of applying $\tilde{\mathbf{A}}$: $O\left((kp)^2\right)$

Cost of applying $\tilde{\mathbf{U}}$: $O(kpn)$

Total cost: $O(N^2)$ but smaller constant.

The cost is dominated by the small dense matrix.

Recall: $p = N/n$

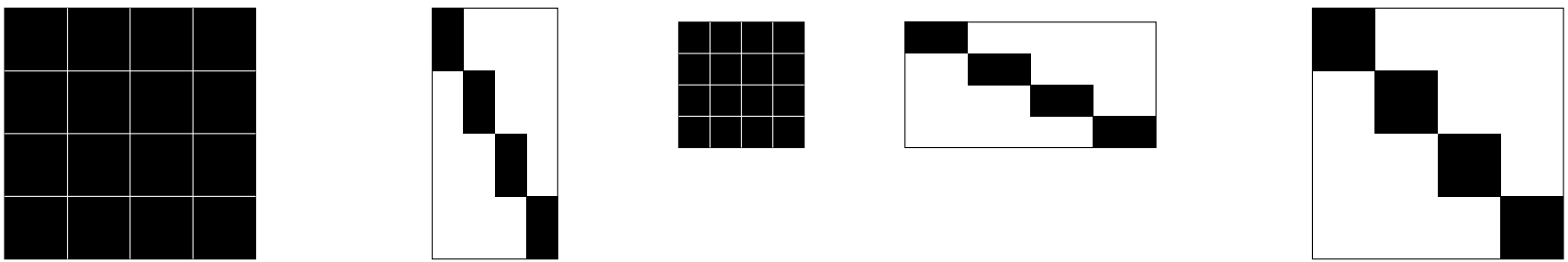
Intuition for inversion of Hierarchically Block-Separable matrices

Lemma: [Variation of Woodbury] If an $N \times N$ matrix $\mathbf{A}$ admits the factorization

$$\mathbf{A} = \mathbf{U}\,\tilde{\mathbf{A}}\,\mathbf{V}^* + \mathbf{D},$$

then

$$\underset{pn \times pn}{\mathbf{A}^{-1}} = \underset{pn \times pk}{\mathbf{E}}\ \underset{pk \times pk}{(\tilde{\mathbf{A}} + \hat{\mathbf{D}})^{-1}}\ \underset{pk \times pn}{\mathbf{F}^*} + \underset{pn \times pn}{\mathbf{G}},$$



where (provided all intermediate matrices are invertible)

$$\hat{\mathbf{D}} = (\mathbf{V}^*\,\mathbf{D}^{-1}\,\mathbf{U})^{-1}, \quad \mathbf{E} = \mathbf{D}^{-1}\,\mathbf{U}\,\hat{\mathbf{D}}, \quad \mathbf{F} = (\hat{\mathbf{D}}\,\mathbf{V}^*\,\mathbf{D}^{-1})^*, \quad \mathbf{G} = \mathbf{D}^{-1} - \mathbf{D}^{-1}\,\mathbf{U}\,\hat{\mathbf{D}}\,\mathbf{V}^*\,\mathbf{D}^{-1}.$$

Note: All matrices set in blue are block diagonal.

Intuition for inversion of Hierarchically Block-Separable matrices

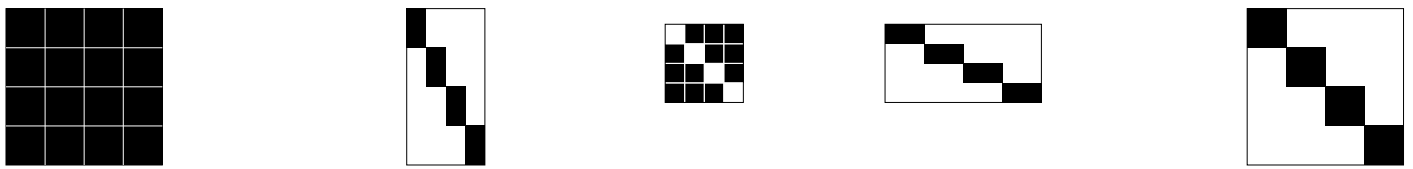
The Woodbury formula reduces the cost of inversion from $(p\,n)^3$ to $(p\,k)^3$.
This is not "fast" yet.

Intuition for inversion of Hierarchically Block-Separable matrices

The Woodbury formula reduces the cost of inversion from $(p\,n)^3$ to $(p\,k)^3$.
This is not "fast" yet.

What if we could factor **A** more?

$$\underset{p\,n \times p\,n}{\mathbf{A}} = \underset{p\,n \times p\,k}{\mathbf{U}}\ \underset{p\,k \times p\,k}{\tilde{\mathbf{A}}}\ \underset{p\,k \times p\,n}{\mathbf{V}^*} + \underset{p\,n \times p\,n}{\mathbf{D}},$$

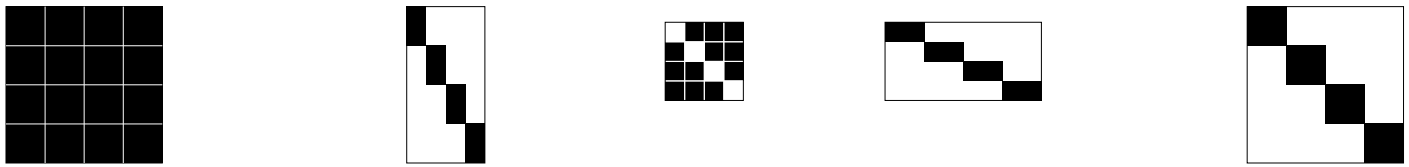


Intuition for inversion of Hierarchically Block-Separable matrices

The Woodbury formula reduces the cost of inversion from $(p\,n)^3$ to $(p\,k)^3$.
This is not "fast" yet.

What if we could factor $\mathbf{A}$ more?

$$\underset{p\,n\times p\,n}{\mathbf{A}} = \underset{p\,n\times p\,k}{\mathbf{U}}\ \underset{p\,k\times p\,k}{\tilde{\mathbf{A}}}\ \underset{p\,k\times p\,n}{\mathbf{V}^*} + \underset{p\,n\times p\,n}{\mathbf{D}},$$

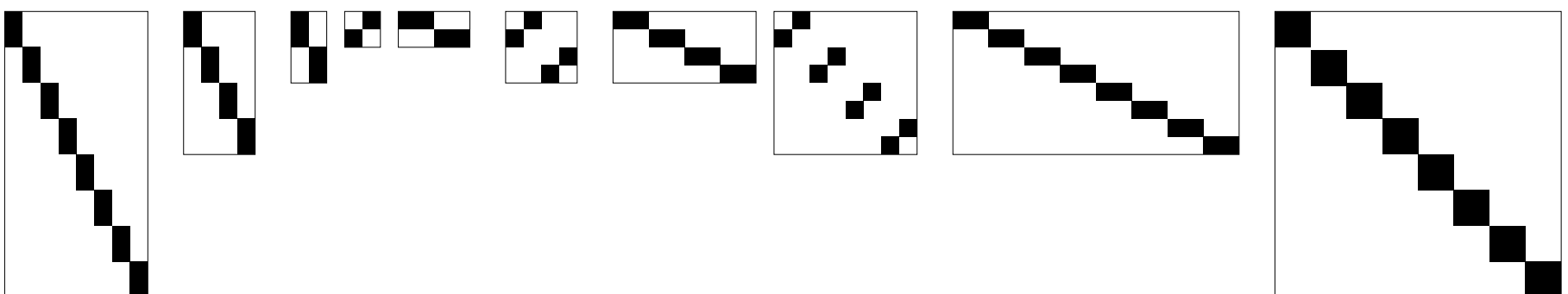


Using a telescoping factorization of $\mathbf{A}$:

$$\mathbf{A} = \mathbf{U}^{(3)}\left(\mathbf{U}^{(2)}\left(\mathbf{U}^{(1)}\,\mathbf{B}^{(0)}\,(\mathbf{V}^{(1)})^*\right) + \mathbf{B}^{(1)}\right)(\mathbf{V}^{(2)})^* + \mathbf{B}^{(2)}\Big)(\mathbf{V}^{(3)})^* + \mathbf{D}^{(3)},$$

Block structure of factorization:

$\mathbf{U}^{(3)}$ $\mathbf{U}^{(2)}$ $\mathbf{U}^{(1)}$ $\mathbf{B}^{(0)}$ $(\mathbf{V}^{(1)})^*$ $\mathbf{B}^{(1)}$ $(\mathbf{V}^{(2)})^*$ $\mathbf{B}^{(2)}$ $(\mathbf{V}^{(3)})^*$ $\mathbf{D}^{(3)}$



Outline

1. Integral equations
2. Fast direct solver classification
3. Illustration of an inversion technique
4. Techniques for creating low rank factorizations
5. Results for an integral equation
6. Extensions
7. Summary
8. Some available software

Creating the low-rank factorization of a block

Consider the tasks of creating a rank-k factorization of a matrix $\mathbf{B}$ of size $n \times m$. With loss of generalization, let $m >> n$.

- Brute force
  Pros: Just works Cons: Expensive $O(mn^2)$.

Creating the low-rank factorization of a block

Consider the tasks of creating a rank-k factorization of a matrix $\mathbf{B}$ of size $n \times m$. With loss of generalization, let $m >> n$.

- Brute force
  Pros: Just works Cons: Expensive $O(mn^2)$.

- Localized Harmonic expansions
  Pros: Analytic, easy control over accuracy Cons: Does not exist for general data sparse matrices.

Creating the low-rank factorization of a block

Consider the tasks of creating a rank-k factorization of a matrix $\mathbf{B}$ of size $n \times m$. With loss of generalization, let $m >> n$.

- Brute force
  Pros: Just works Cons: Expensive $O(mn^2)$.

- Localized Harmonic expansions
  Pros: Analytic, easy control over accuracy Cons: Does not exist for general data sparse matrices.

- Adaptive Cross Approximation
  Pros: Fast $O(nmk)$ Cons: Can fail.

Creating the low-rank factorization of a block

Consider the tasks of creating a rank-k factorization of a matrix $\mathbf{B}$ of size $n \times m$. With loss of generalization, let $m >> n$.

- Brute force
  Pros: Just works Cons: Expensive $O(mn^2)$.

- Localized Harmonic expansions
  Pros: Analytic, easy control over accuracy Cons: Does not exist for general data sparse matrices.

- Adaptive Cross Approximation
  Pros: Fast $O(nmk)$ Cons: Can fail.

- Randomized compression
  Pros: Fast $O(mnk)$ Cons: Requires the ability to apply the matrix.

The Interpolatory Decomposition

Let $\mathbf{B}$ denote an $m \times n$ matrix of (precise) rank k. Then $\mathbf{B}$ admits a factorization

$$\underset{m \times n}{\mathbf{B}} = \underset{m \times k}{\mathbf{U}} \quad \underset{k \times k}{\mathbf{B}^{(\mathrm{skel})}} \quad \underset{k \times n}{\mathbf{V}^*},$$

where

1. $\mathbf{B}^{(\mathrm{skel})} = \mathbf{B}(I_{\mathrm{row}}, I_{\mathrm{col}})$ is a submatrix of $\mathbf{B}$.
2. $\mathbf{U}$ and $\mathbf{V}$ both contain a $k \times k$ identity matrix.
3. No entry of $\mathbf{U}$ or $\mathbf{V}$ has magnitude greater than 1 (so $\mathbf{U}$ and $\mathbf{V}$ are well-conditioned).

(2) is for reduced storage

Physics based acceleration

For integral equations, it is possible to gain further acceleration in the factorization by exploiting physics.

Specifically, it is known that the feild due to point charges that are far away can be captured by a collection of representative charges.

In practice a collection of *proxy charges* are placed on a ball of radius R. The number of proxy charges n_{proxy} needed to achieve a desired accuracy depends on the problem and the radius R.

For example, for Laplace problems if the radius is 1.75 the size of the ball containing the region of interest, $n_{\text{proxy}} = 50$ is sufficient.

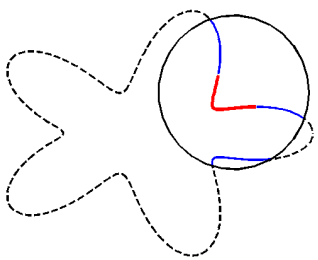
Physics based acceleration

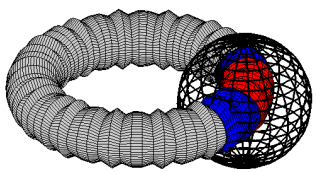
As a result we can compress

$$[\mathbf{A}_{\text{near}}\mathbf{A}_{\text{proxy}}]$$

instead of

$$\mathbf{A}_{\text{off}-\text{diag}}$$

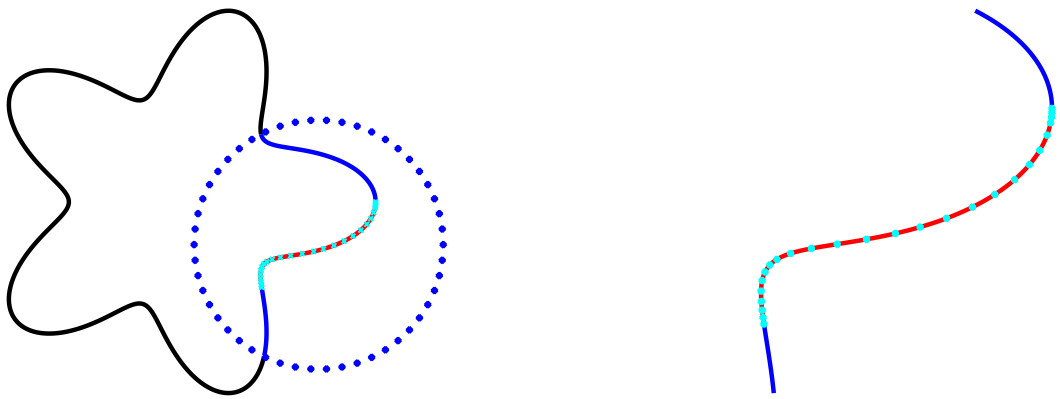




The proxy points live on the surfaces in black. The blue regions correspond to the near feild.

Physical connection with the factorization

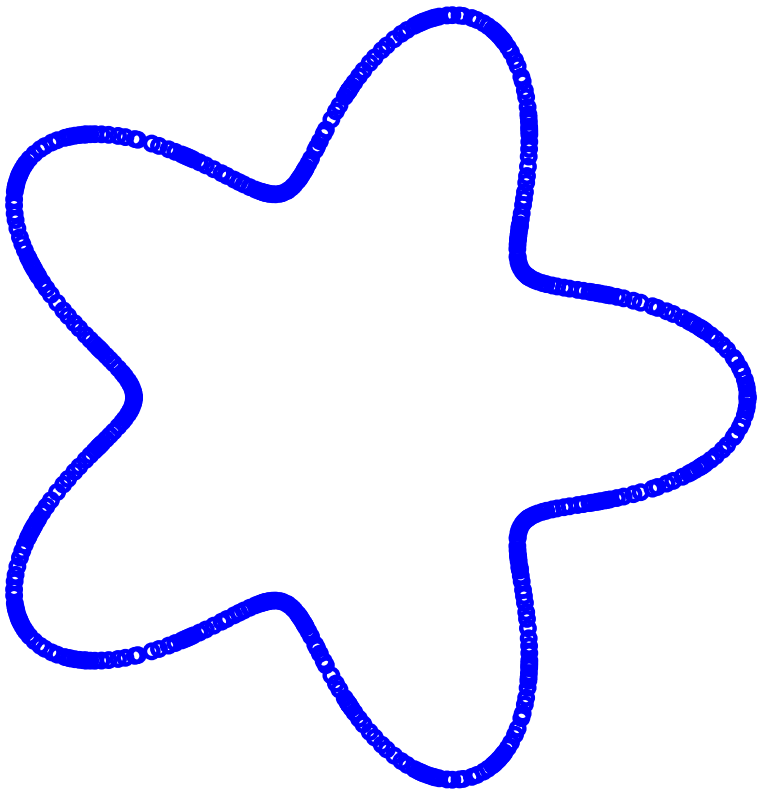
There are 2000 points on the star boundary, 250 points on the red segment and 300 points in the blue segments. 50 points are placed on the proxy surface.



The tolerance was set to 10^{-8} and there are 26 skeleton points marked in cyan.

Illustation of skeletons

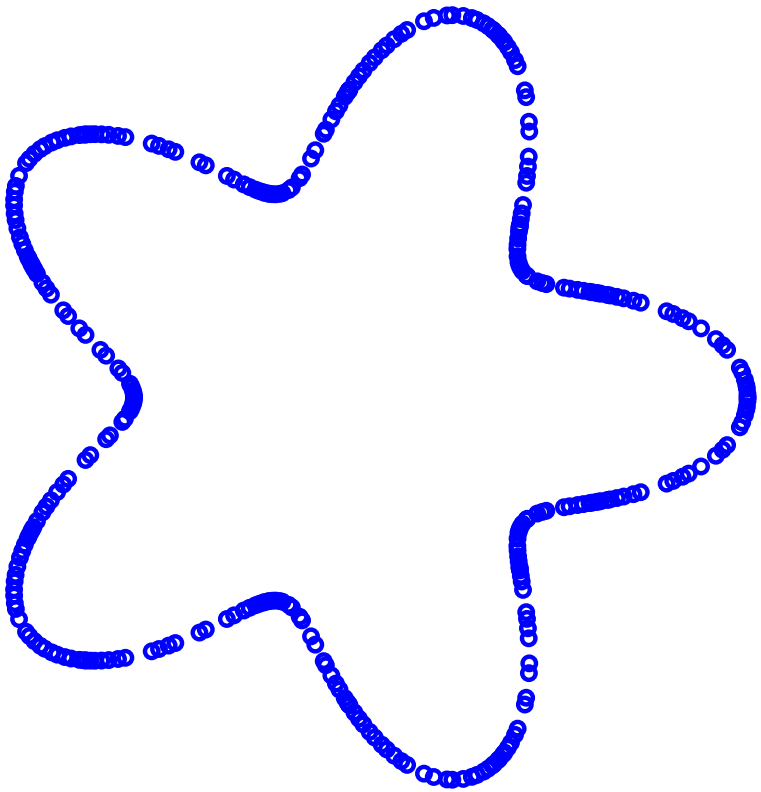
Star geometry



$n_{\text{skel}} = 544$

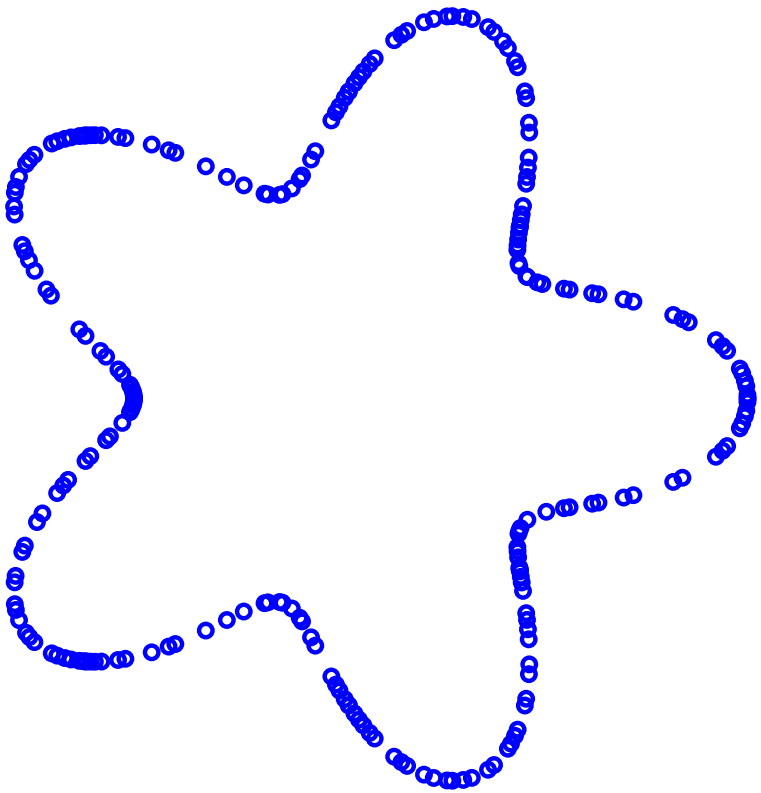
Illustation of skeletons

Star geometry



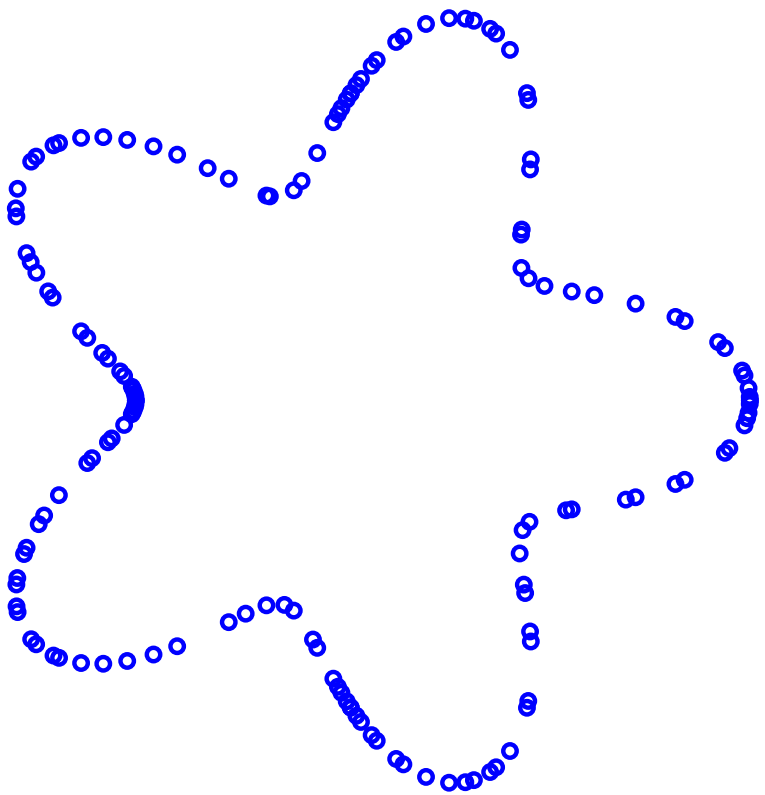
$n_{\text{skel}} = 356$

Illustation of skeletons



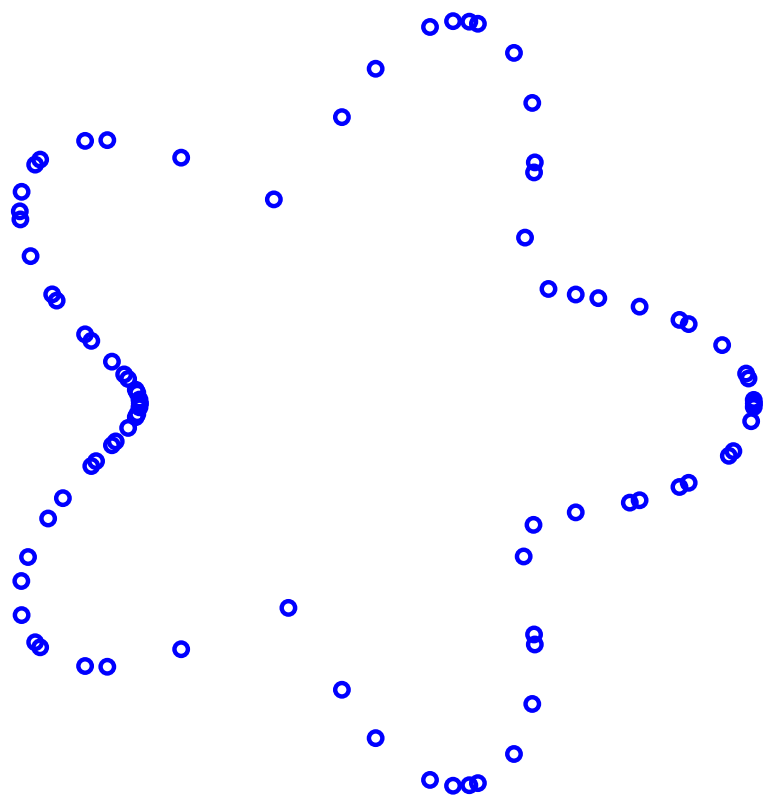
$n_{\text{skel}} = 242$

Illustation of skeletons



$n_{\text{skel}} = 151$

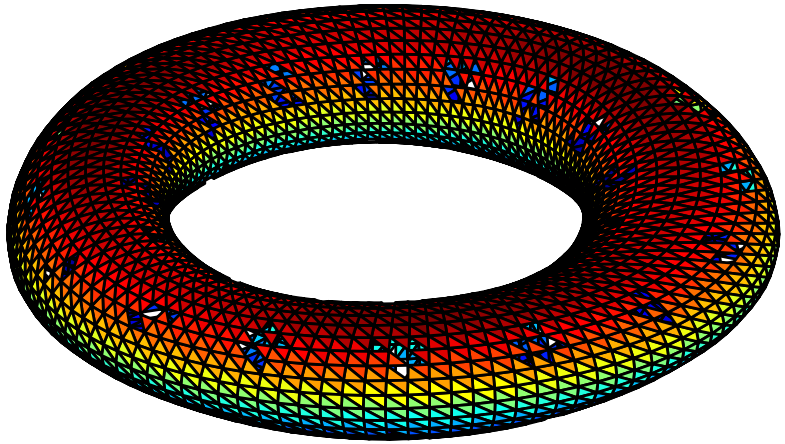
Illustation of skeletons



$n_{\text{skel}} = 85$

Illustation of skeletons

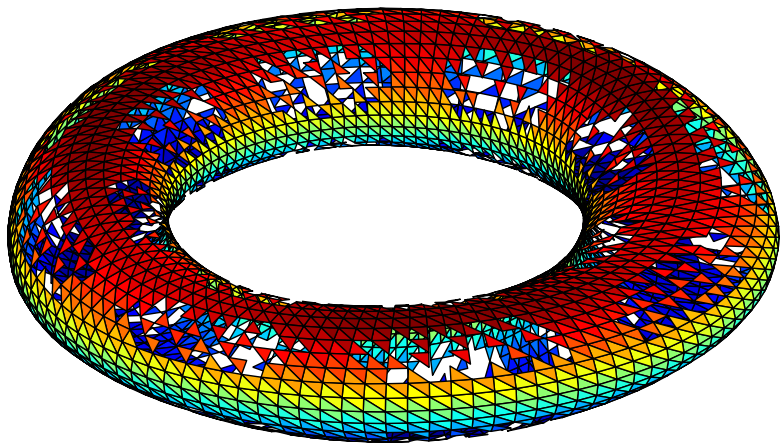
Torus



$n_{\text{skel}} = 5839$

Illustation of skeletons

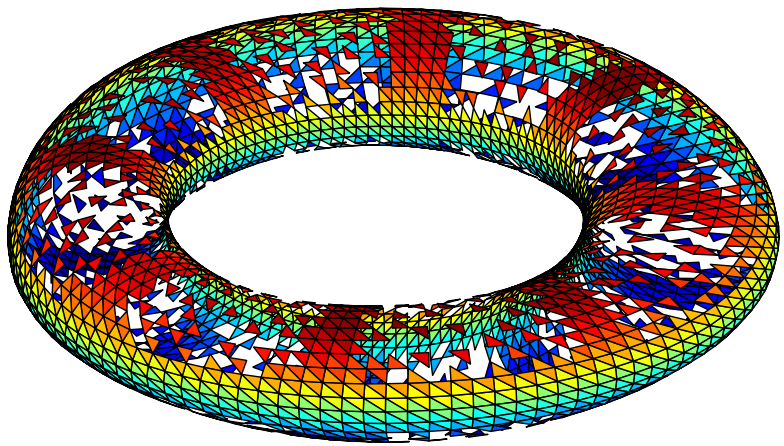
Torus



$n_{\text{skel}} = 4549$

Illustation of skeletons

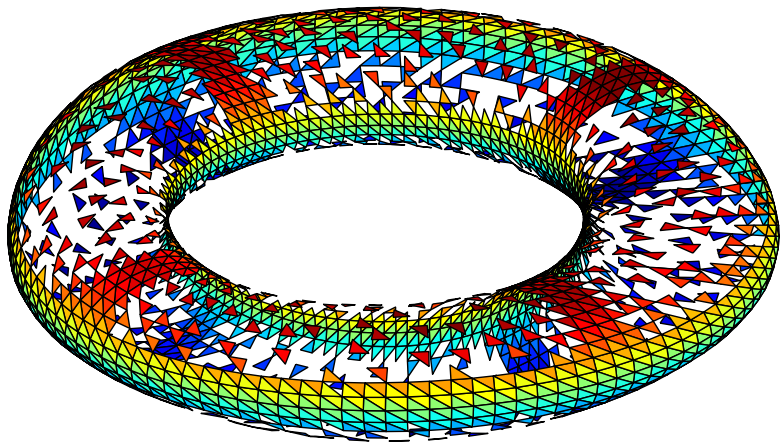
Torus



$n_{\text{skel}} = 3524$

Illustation of skeletons

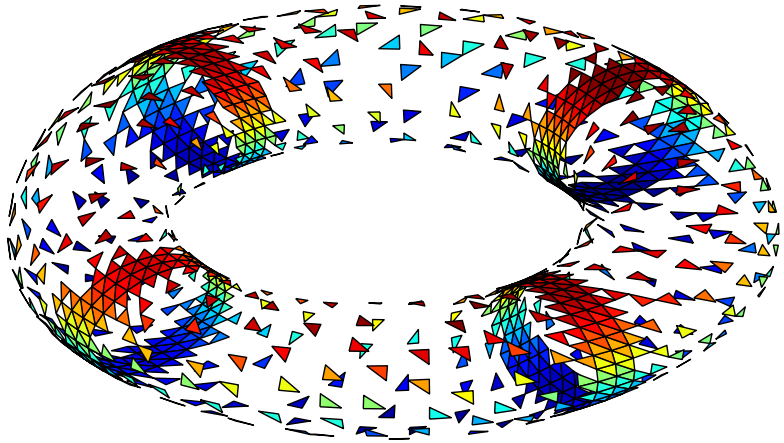
Torus



$n_{\text{skel}} = 2789$

Illustation of skeletons

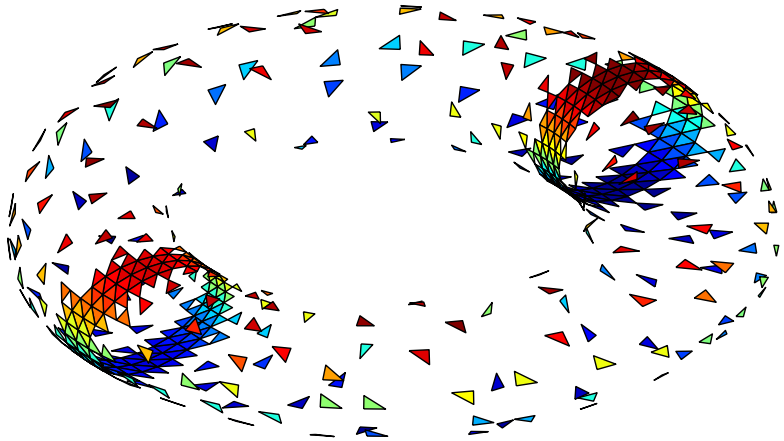
Torus



$n_{\text{skel}} = 1142$

Illustation of skeletons

Torus



$n_{\text{skel}} = 584$

Demo codes

Codes will expore the use of ID when creating a low rank factorization of an off-diagonal block of a matrix arising from an integral equation.

Matlab
ID factorization (mex) is thanks to Xin Xing

Python
ID factorization is thanks to Ken Ho and is part of scipy.

In practice, we use the package from Mark Tygert. Documentation

Demo codes

Demo codes

ID_test: code for playing with the ID.
Fun exercises:

1. Compare the cost of computing factorization via svd and ID for different martix sizes.
2. Look at where the skeleton points are located.

proxy_demo: Code for playing with the proxy surface.
Fun exercises:

1. Play with the size of the radius for a fixed number of proxy points. How does that impact the cost? Does the accuracy change?
2. Play with the number of proxy points for a couple of different relative radius sizes; i.e. $rel_rad = 1.1, 1.75, 2$. How does that impact the number of proxy points impact the cost? How does the accuracy change for a given radius size?

Outline

1. Integral equations
2. Fast direct solver classification
3. Illustration of an inversion technique
4. Techniques for creating low rank factorizations
5. Results for an integral equation
6. Extensions
7. Summary
8. Some available software

Numerical examples

These numerical examples were run on standard office desktops.

Most of the programs are written in Matlab (some in Fortran 77).

The reported CPU times have two components:

(1) Pre-computation (inversion, LU-factorization, constructing a Schur complement)

(2) Time for a single solve once pre-computation is completed

Numerical examples

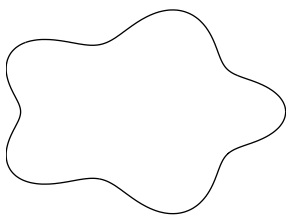
We invert a matrix approximating the operator

$$[A\,\phi](\mathbf{x}) = \frac{1}{2}\,\phi(\mathbf{x}) + \int_\Gamma D(\mathbf{x},\mathbf{y})\,\phi(\mathbf{y})\,ds(\mathbf{y}), \qquad \mathbf{x} \in \Gamma,$$

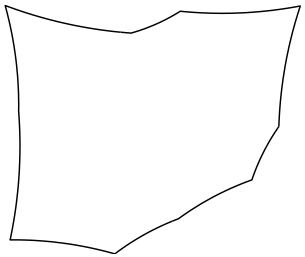
where D is the double layer kernel associated with Laplace's equation,

$$D(\mathbf{x},\mathbf{y}) = \frac{1}{2\pi}\,\frac{\mathbf{n}(\mathbf{y})\cdot(\mathbf{x}-\mathbf{y})}{|\mathbf{x}-\mathbf{y}|^2},$$

and where Γ is either one of the contours:



Smooth star

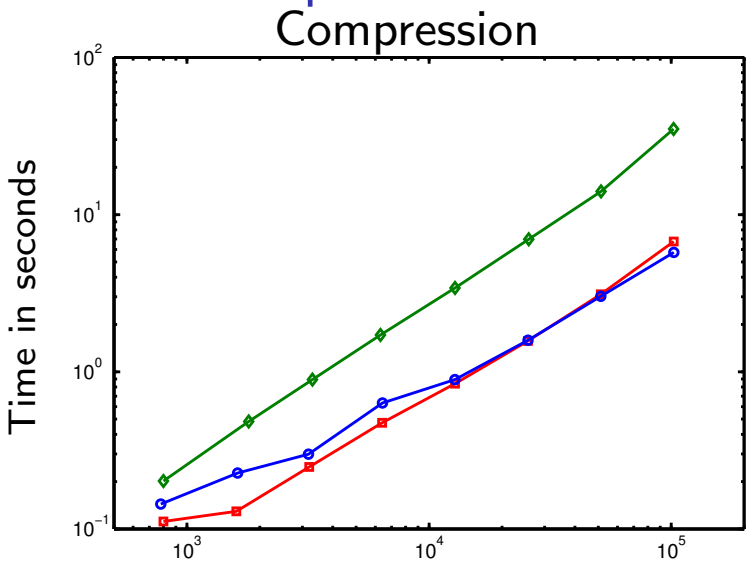


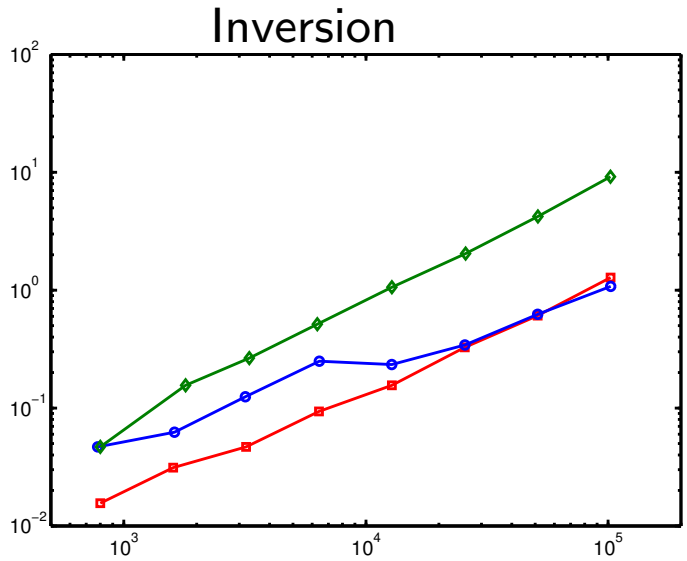
Star with corners
(local refinements at corners)

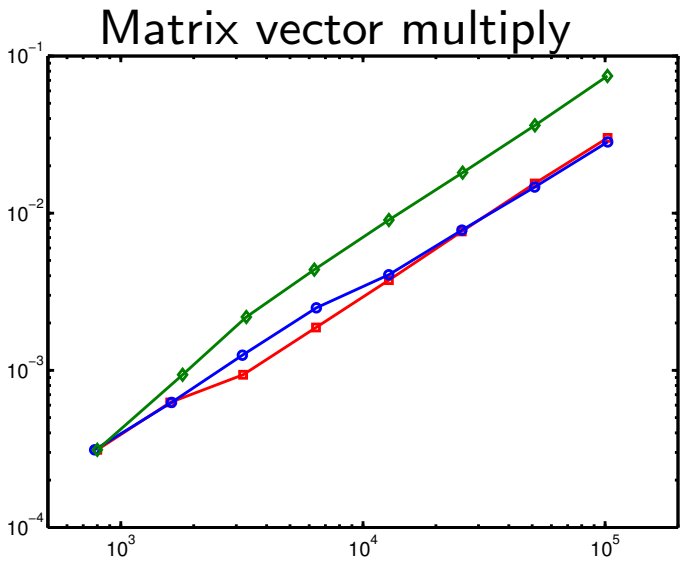
Snake
(# oscillations $\sim N$)

Examples from "A direct solver with $O(N)$ complexity for integral equations on one-dimensional domains," with P. Young, and P.G. Martinsson.

Numerical examples



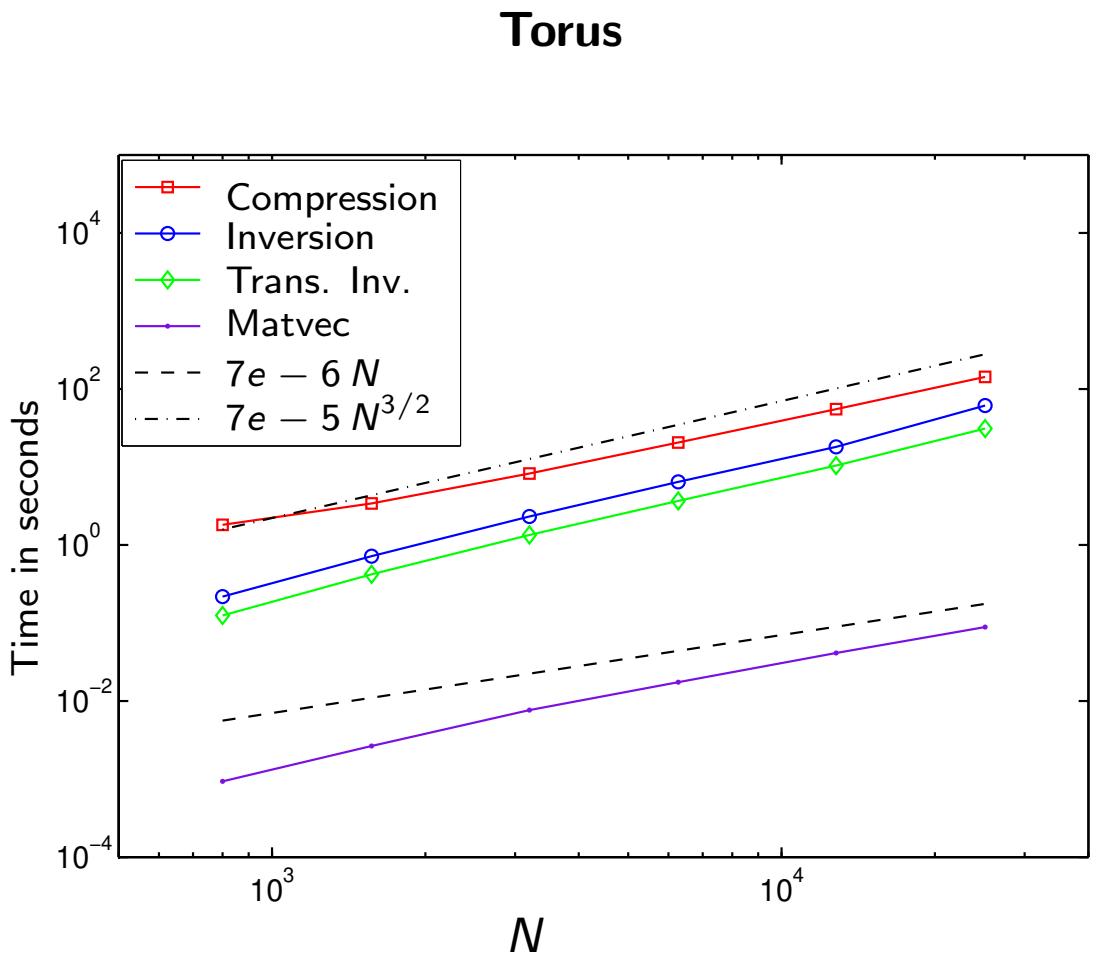




Within each graph, the three lines correspond to the three examples considered:

□ Smooth star ○ Star with corners ◇ Snake

Numerical results



Outline

1. Integral equations
2. Fast direct solver classification
3. Illustration of an inversion technique
4. Techniques for creating low rank factorizations
5. Results for an integral equation
6. Extensions
7. Summary
8. Some available software

Getting the most of your used flops

While the fast direct solvers have optimal or nearly optimal computational cost, they are expensive to construct.

The most common justification for using them include

- Needing to solve the same system many times
- The system is ill-conditioned
- An iterative solver would be too slow.

There is an effort to extend the range of problems for which the fast solvers can be applied.
For some applications, it is possible to use a precomputed fast direct solver for systems that are "close" to the original system.

Extending the use of a direct solver

What do we mean by a system being close to the original system?

$$\mathbf{A}_{\text{new}} = \mathbf{A}_{\text{old}} + \mathbf{LR}$$

where **LR** is a low rank update to the original system.

Extending the use of a direct solver

What do we mean by a system being close to the original system?

$$\mathbf{A}_{\text{new}} = \mathbf{A}_{\text{old}} + \mathbf{LR}$$

where **LR** is a low rank update to the original system.

Why is it advantegous to write the new system in this way?

Woodbury Formula: inversion of a low-rank update

$$(\mathbf{A}_{\text{old}} + \mathbf{LR})^{-1} = \mathbf{A}_{\text{old}}^{-1} - \mathbf{A}_{\text{old}}^{-1}\mathbf{L}\left(\mathbf{I} + \mathbf{R}\mathbf{A}_{\text{old}}^{-1}\mathbf{L}\right)^{-1}\mathbf{R}\mathbf{A}_{\text{old}}^{-1}$$

Extending the use of a direct solver

What do we mean by a system being close to the original system?

$$\mathbf{A}_{\text{new}} = \mathbf{A}_{\text{old}} + \mathbf{LR}$$

where **LR** is a low rank update to the original system.

Why is it advantegous to write the new system in this way?

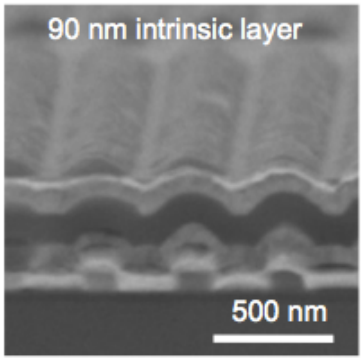
Woodbury Formula: inversion of a low-rank update

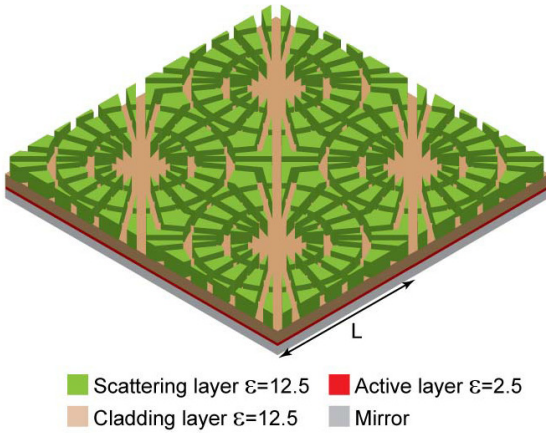
$$(\mathbf{A}_{\text{old}} + \mathbf{LR})^{-1} = \mathbf{A}_{\text{old}}^{-1} - \mathbf{A}_{\text{old}}^{-1}\mathbf{L}\left(\mathbf{I} + \mathbf{R}\mathbf{A}_{\text{old}}^{-1}\mathbf{L}\right)^{-1}\mathbf{R}\mathbf{A}_{\text{old}}^{-1}$$

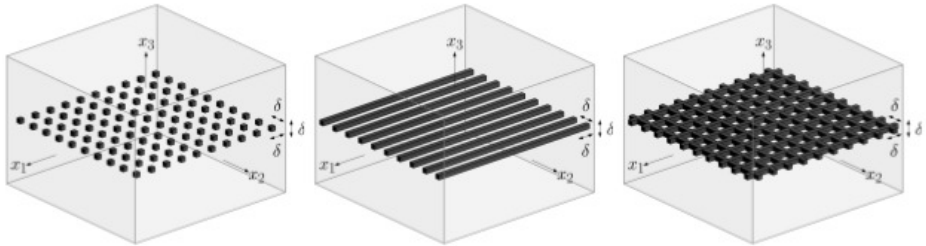
Thus far this has been used for:

- Periodic scattering and stokes problems
- Local changes in the geometry.

Applications of involving periodic geometries



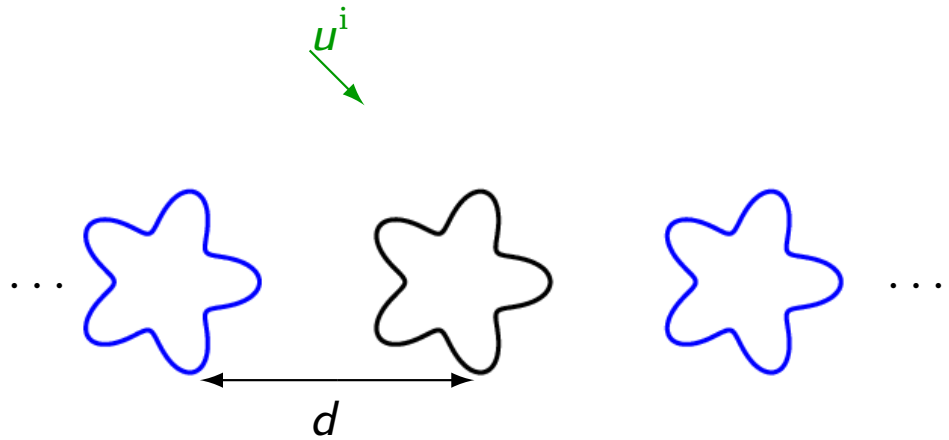




(a) Case (i) - discrete obstacles (b) Case (ii) - parallel wires (c) Case (iii) - wire mesh

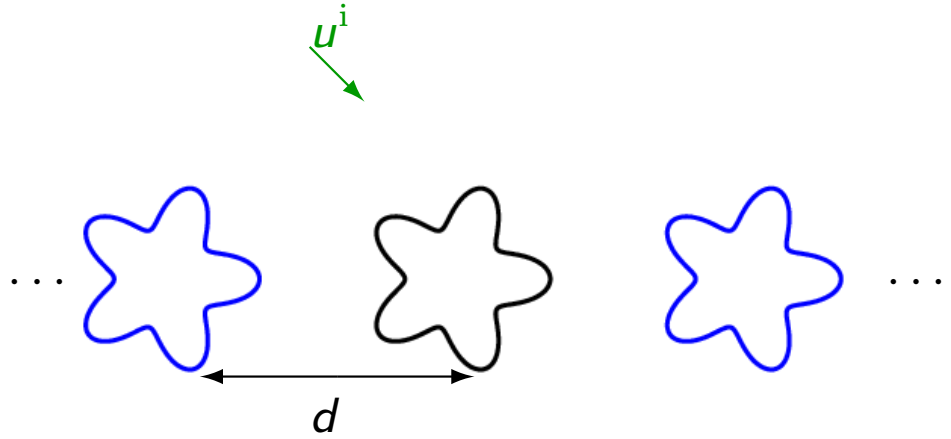
Figure 2: The domain Ω^δ in the three cases under consideration.

Definition of quasi-periodic scattering



- Let $\Omega \subset \mathbb{R}^2$ denote one obstacle. Then the collection of obstacles is expressed as $\Omega_{\mathbb{Z}} = \{\mathbf{x} : (x + nd, y) \in \Omega \text{ for some } n \in \mathbb{Z}\}$.
- The obstacles are hit by an incident plane wave $u^{\mathrm{i}} = e^{\mathrm{i}\mathbf{k}\cdot\mathbf{x}}$ where $|k| = \omega$.
- Our goal is to find the total field $u^{\mathrm{total}} = u^{\mathrm{inc}} + u$.
- Utilize the fact that each part of the field is quasi-periodic: i.e. $u(x + d, y) = \alpha u(x, y)$ where $\alpha = e^{\mathrm{i}\omega d \cos\theta}$ denotes the Bloch phase and θ is the angle of the incident wave.

Differential equation

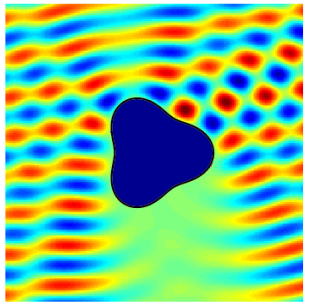


$$(\Delta + \omega^2)u(\mathbf{x}) = 0 \qquad \mathbf{x} \in \mathbb{R}^2 \setminus \Omega_{\mathbb{Z}}$$

$$u(\mathbf{x}) = -u^{\mathrm{i}}(\mathbf{x}) \qquad \mathbf{x} \in \partial\Omega_{\mathbb{Z}}$$

$$u \quad \text{'radiative' as} \quad \mathrm{y} \to \pm\infty$$

Single object scattering



Consider the problem

$$(\Delta + \omega^2)u^{\mathrm{s}}(\mathbf{x}) = 0 \qquad \mathbf{x} \in \mathbb{R} \setminus \Omega$$
$$u^{\mathrm{s}}(\mathbf{x}) = u^{\mathrm{i}}(\mathbf{x}) \qquad \mathbf{x} \in \partial\Omega$$
u^{s} 'radiative' far from Ω

The solution can be represented as a double layer potential

$$u^{\mathrm{s}}(\mathbf{x}) = \int_{\Gamma} \partial_{\boldsymbol{\nu}} G_\omega(\mathbf{x}, \mathbf{y})\tau(\mathbf{y})ds(\mathbf{y}), \qquad \mathbf{x} \in \Omega,$$

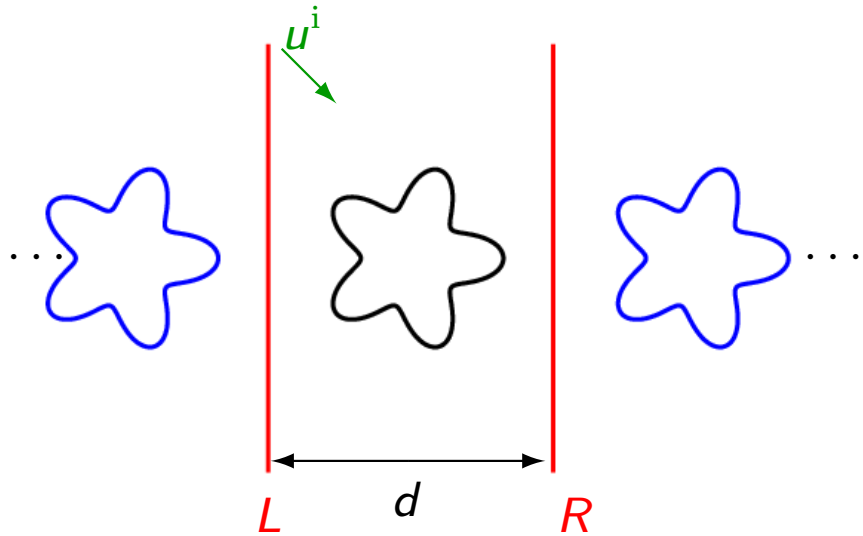
where $\boldsymbol{\nu}$ is the outward normal and $G_\omega(\mathbf{x}, \mathbf{y})$ is the fundamental solution

$$G_\omega(\mathbf{x}, \mathbf{y}) = \frac{i}{4} H_0^{(1)}(\omega|\mathbf{x} - \mathbf{y}|).$$

Then the boundary charge distribution τ satisfies the boundary integral equation

$$-\frac{1}{2}\tau(\mathbf{x}) + \int_{\Gamma} \partial_{\boldsymbol{\nu}} G_\omega(\mathbf{x}, \mathbf{y})\tau(\mathbf{y})ds(\mathbf{y}) = u^{\mathrm{i}}(\mathbf{x})$$

One approach to solving the periodic problem



Let the solution be represented as a double layer potential plus a quasi-periodic potential

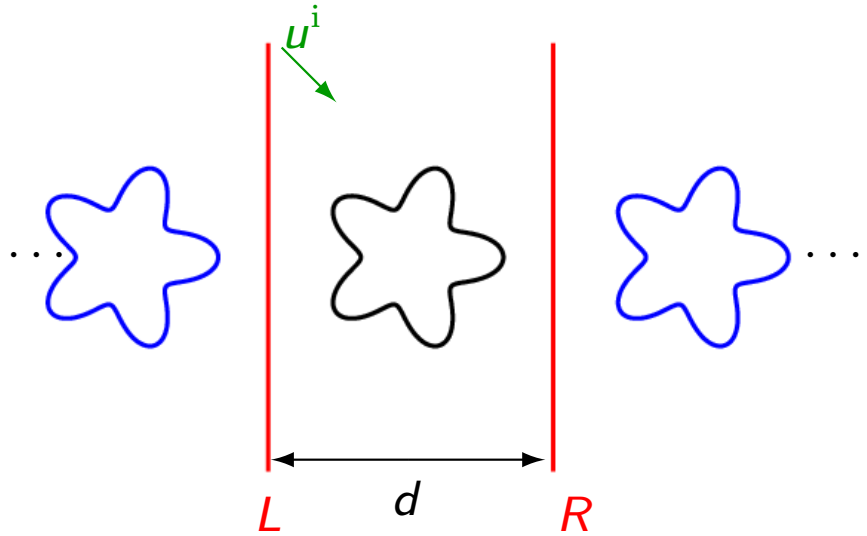
$$u(\mathbf{x}) = \sum_{j=-1}^{1} \alpha^j \int_{\partial\Omega} \partial_\nu G_\omega(\mathbf{x}, \mathbf{y} + j\mathbf{d}) \tau(\mathbf{y}) ds(\mathbf{y}) + u_{QP}[\xi].$$

New condition: vanishing 'discrepancy'

$$\begin{cases} u_L - \alpha^{-1} u_R &= 0 \\ u_{nL} - \alpha^{-1} u_{nR} &= 0 \end{cases}$$

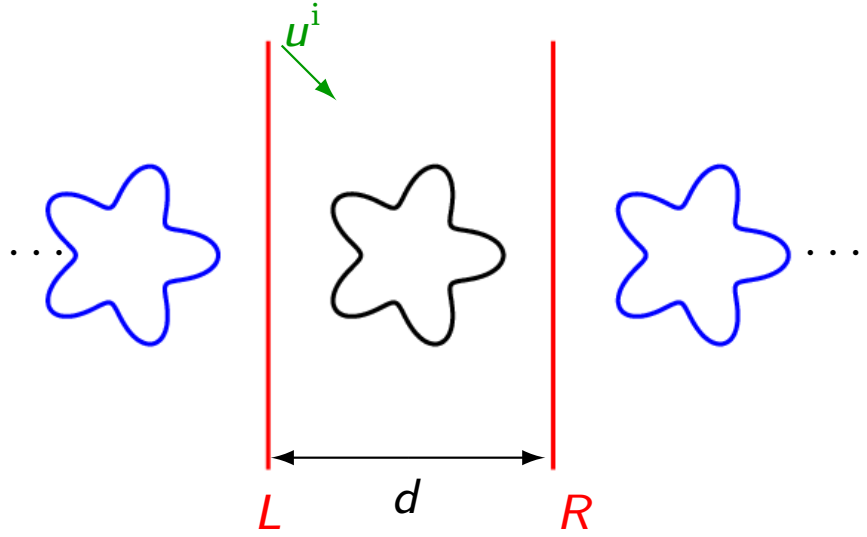
L. Greengard and A. Barnett (2011)

One approach to solving the periodic problem



$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{Q} \end{bmatrix} \begin{bmatrix} \tau \\ \xi \end{bmatrix} = \begin{bmatrix} -\mathbf{u}^i \\ \mathbf{0} \end{bmatrix}$$

One approach to solving the periodic problem



$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{Q} \end{bmatrix} \begin{bmatrix} \tau \\ \xi \end{bmatrix} = \begin{bmatrix} -\mathbf{u}^{\mathrm{i}} \\ \mathbf{0} \end{bmatrix}$$

Instead of computing a pseudo-inverse of the matrix, we can compute the solution via a 2×2 block solve.

$$\xi = (\mathbf{Q} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B})^{-1}\mathbf{A}^{-1}\mathbf{u}^{\mathrm{inc}}$$

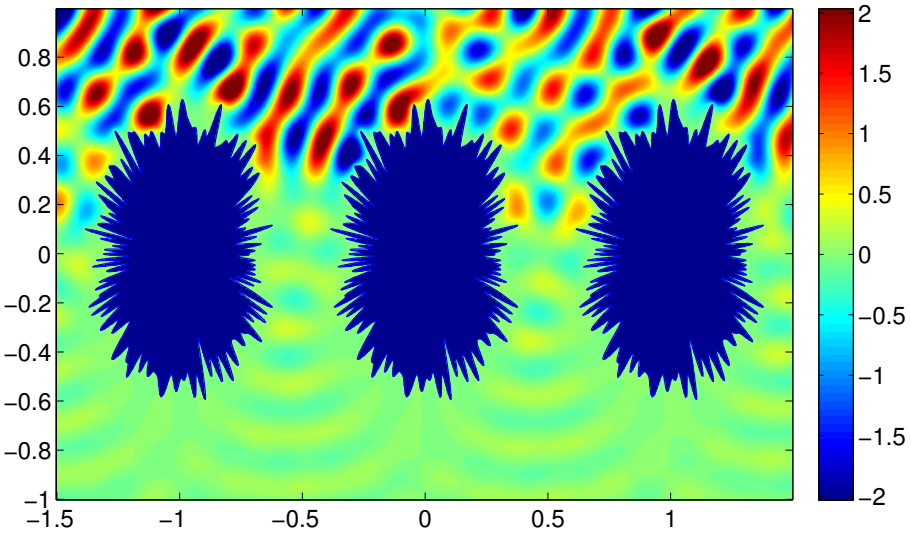
$$\tau = \mathbf{A}^{-1}\mathbf{u}^{\mathrm{inc}} - \mathbf{A}^{-1}\mathbf{B}\xi$$

$\mathbf{A} = \mathbf{A}_0 + \alpha^{-1}\mathbf{A}_{-1} + \alpha\mathbf{A}_1$

For a problem with a fixed wave number ω, many incident waves will share a Bloch phase so the inversion technique can be reused. Cost: $O(N + M^3)$

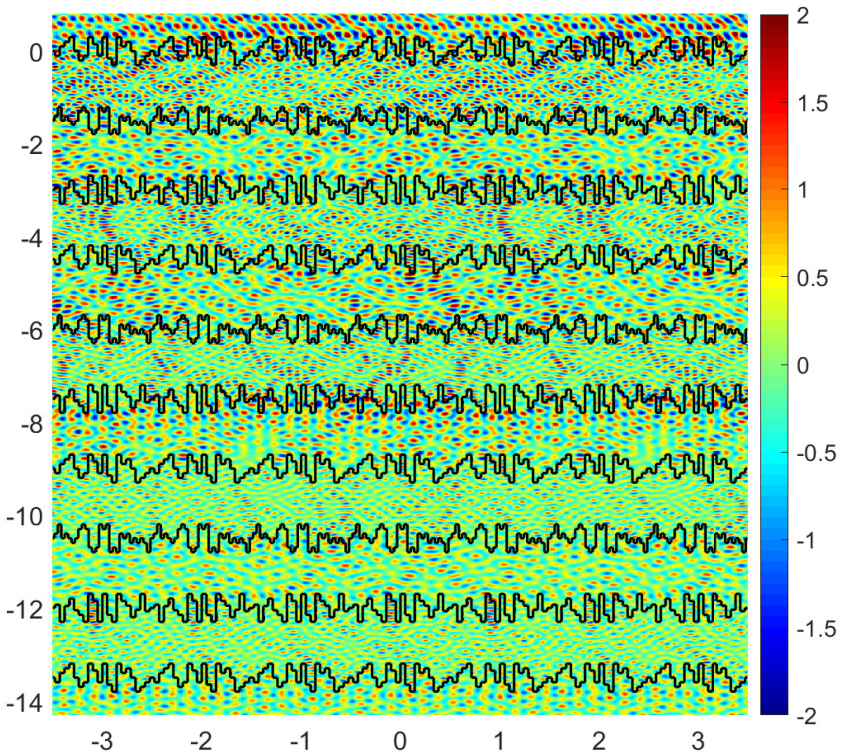
Example

Multiple incident waves



The FMM + GMRES takes one hour (248 iterations) to solve for the densities for one incident wave.
The fast direct solver takes 19.1 minutes to solve 200 densities.
(4.1 minutes of precomputation and 15 minutes for the block solves.)
Example from "A fast direct solver for quasi-periodic scattering problems," with A. Barnett, 2013.

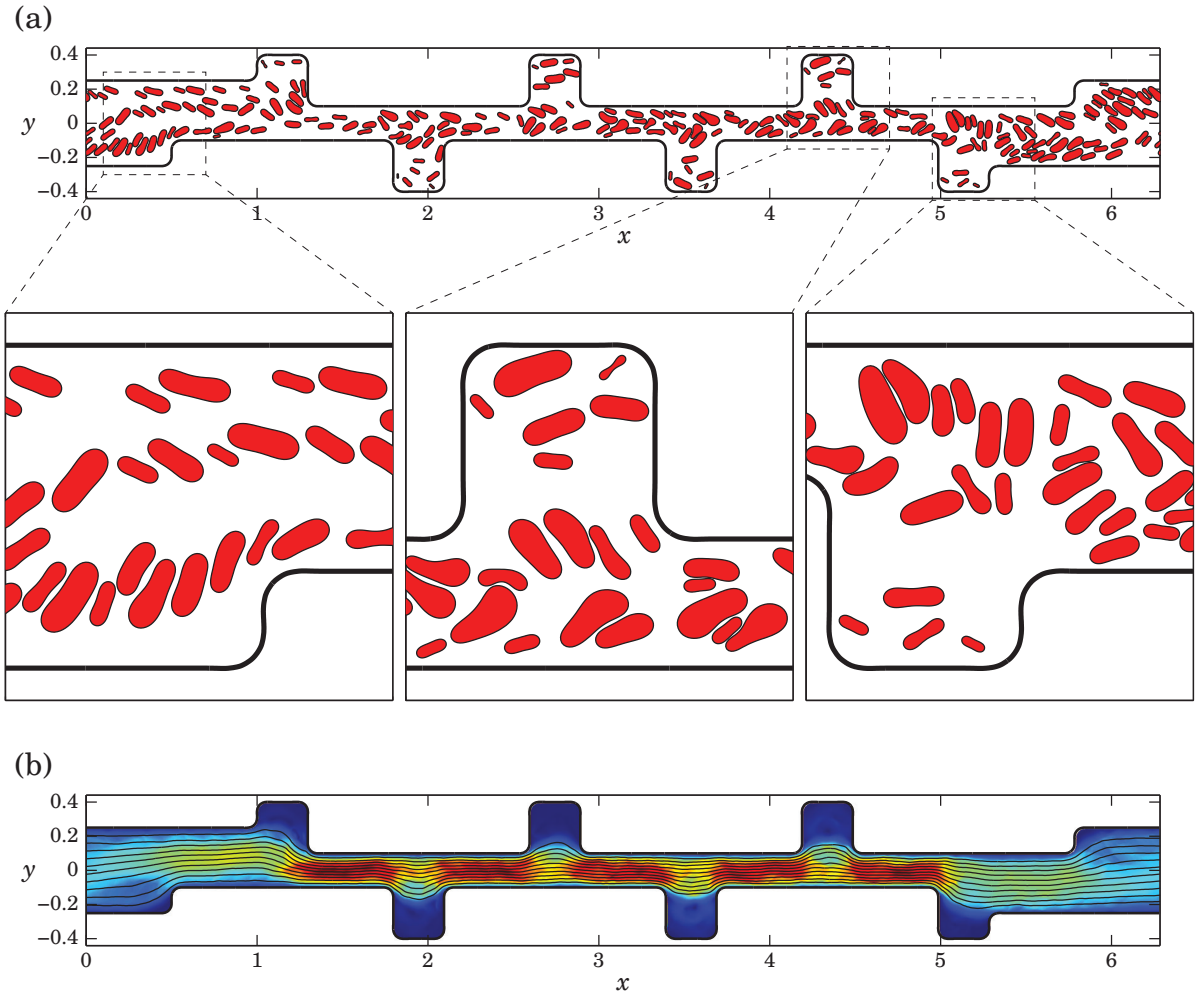
Lots of solves



N_{total}	Precomp	Solve
121136	4759.7	432.2 (1.5 per incident angle)

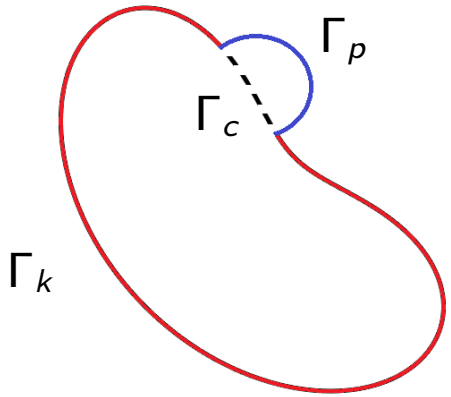
Example from "Part II: A fast direct solution technique for quasi-periodic scattering in multi-layered medium," with Y. Zhang.

Stokes flow



Example from "A fast algorithm for simulating multiphase flows through periodic geometries of arbitrary shape," with G. Marple, A. Barnett, and S. Veerapaneni.

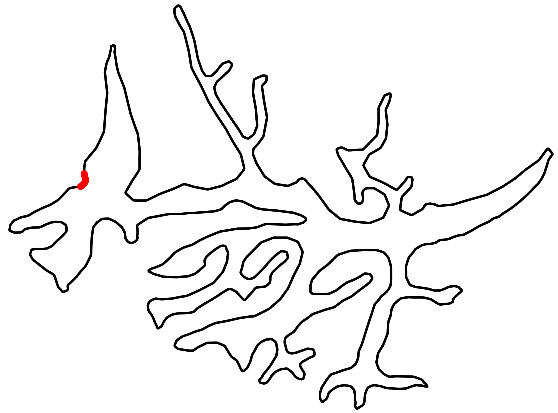
Local peturbation problem



$$-\Delta u(\mathbf{x}) = 0, \qquad \mathbf{x} \in \Omega,$$
$$u(\mathbf{x}) = g(\mathbf{x}), \quad \mathbf{x} \in \Gamma.$$

$$\left(\underbrace{\begin{bmatrix} \mathbf{A}_{oo} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{pp} \end{bmatrix}}_{\tilde{\mathbf{A}}} + \underbrace{\begin{bmatrix} \mathbf{0} & -\mathbf{A}_{kc} & \mathbf{A}_{kp} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{A}_{pk} & \mathbf{0} & \mathbf{0} \end{bmatrix}}_{\mathbf{Q}_{\text{new}}} \right) \underbrace{\begin{bmatrix} \boldsymbol{\sigma}_k \\ \boldsymbol{\sigma}_c^{dum} \\ \boldsymbol{\sigma}_p \end{bmatrix}}_{\boldsymbol{\sigma}_{ext}} = \begin{bmatrix} \mathbf{g}_k \\ \mathbf{0} \\ \mathbf{g}_p \end{bmatrix}$$

Applied to local refinement: Stokes



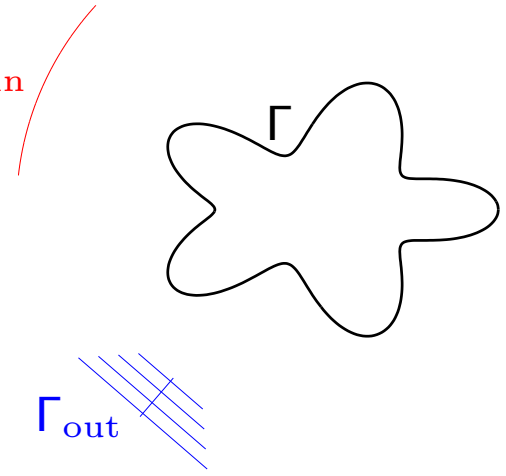
Original	$T_{\text{HBS, comp}}$	$T_{\text{HBS, inv}}$	$T_{\text{HBS, Dsol}}$
	4.09e+2	2.76e+1	7.87e-2
Changed	$\tilde{T}_{\text{HBS, comp}}$	$\tilde{T}_{\text{HBS, inv}}$	$\tilde{T}_{\text{HBS, Dsol}}$
	4.09e+2	2.76e+1	7.87e-2
LR solver	$\tilde{T}_{\text{ELS, comp}}$	$\tilde{T}_{\text{ELS, inv}}$	$\tilde{T}_{\text{ELS, Dsol}}$
	5.33e+0	2.01e+0	9.62e-2

From *A fast direct solver for integral equations on locally refined boundary discretizations and its application to multiphase flow simulations* with Y. Zhang and S. Veerapaneni

The scattering operator

Consider the problem

$$\begin{cases} \Delta u + \omega^2 u = 0 & \text{in } \Omega^c, \\ u = g & \text{on } \Gamma, \\ u \quad \text{'radiative' far from} \quad \Omega \end{cases}$$



To solve this problem, we seek to evaluate

$$u_{\text{out}} = T_{\text{out}} \quad T_{\Gamma}^{-1} \quad T_{\text{in}} \quad s_{\text{in}},$$

where $T_{\text{in}} : L^2(\Gamma_{\text{in}}) \rightarrow L^2(\Gamma)$ is the operator

$$g(x) = [T_{\text{in}} s_{\text{in}}](x) = \int_{\Gamma_{\text{in}}} G_\omega(x,y) s_{\text{in}}(y) ds(y), \qquad x \in \Gamma,$$

$T_\Gamma : L^2(\Gamma) \rightarrow L^2(\Gamma)$ is the operator

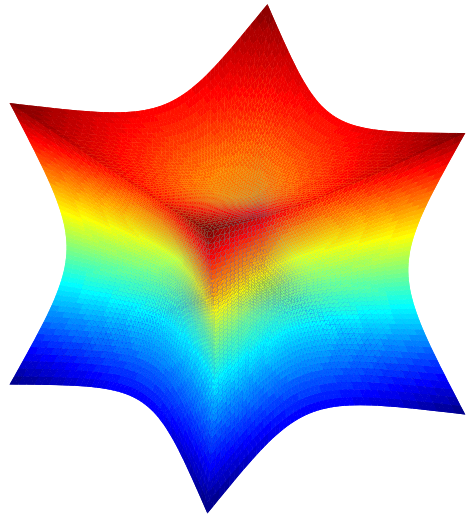
$$[T_\Gamma \sigma](x) = -\frac{1}{2}\sigma(x) + \int_\Gamma \partial_{\nu_y} G_\omega(x,y)\sigma(y) ds(y), \qquad x \in \Gamma,$$

and $T_{\text{out}} : L^2(\Gamma) \rightarrow L^2(\Gamma_{\text{out}})$ is the operator

$$[T_{\text{out}} \sigma](x) = \int_\Gamma \partial_{\nu_y} G_\omega(x,y)\sigma(y) ds(y), \qquad x \in \Gamma_{\text{out}}.$$

Scattering Matrix: Corners and edges

Consider the Neumann boundary value problem on the deformed cube Ω with a fixed wavenumber $\omega = \pi/2$ making the domain approximately 3.46 wavelengths diameter.



N_{tris}	N	E	T	$N_{\text{out}} \times N_{\text{in}}$
192	21 504	2.60×10^{-08}	$6.11 \times 10^{+02}$	617×712
432	48 384	2.13×10^{-09}	$1.65 \times 10^{+03}$	620×694
768	86 016	3.13×10^{-10}	$3.58 \times 10^{+03}$	612×685

$\varepsilon = 1.0 \times 10^{-10}$ 12th order quadrature

Examples are from "A high-order accelerated direct solver for non-oscillatory integral equations on curved surfaces," with J. Bremer, and P.G. Martinsson.

Outline

1. Integral equations
2. Fast direct solver classification
3. Illustration of an inversion technique
4. Techniques for creating low rank factorizations
5. Results for an integral equation
6. Extensions
7. Summary
8. Some available software

Summary

- Fast direct solvers for dense matrices are useful especially when you need to apply the same inverse many times and when the system is ill-conditioned.
- What solver is best for you will depend on your computational resources and what you want to do.
- In some applications, you can create a highly compressed representation of the operator you need without sacraficing memory.
- Recasting your problem so that you can take advantage of a precomputed solver is highly advantegeous.
- While understanding how to use low rank factorizations is very helpful in recasting a problem, there is software to eliminate the need to build full direct solvers.

Outline

1. Integral equations
2. Fast direct solver classification
3. Illustration of an inversion technique
4. Techniques for creating low rank factorizations
5. Results for an integral equation
6. Extensions
7. Summary
8. Some available software

Some available software

- chunkIE - Flatiron Institute
  Two dimensional integral equation package that has FMM and Fast direct solver capabilities.

- BEM++ - Betcke's Group
  Integral equation software that has FMM and MPI capabilities.

- Mumps
  Sparse direct solver has options for acceleration based on Block Low Rank factorizations.

- Berkeley High Performance Hierarchical Matrix Software Suite
  A collection of solver technology for both sparse and dense matrices.
  OpenMP and MPI options are avialable.

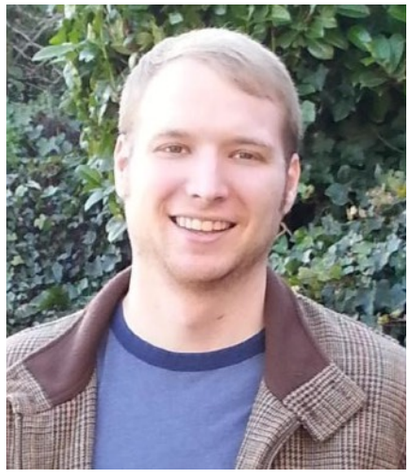
NOTE: This is not a complete list of what is available.

FLATIRON INSTITUTE
Center for Computational Mathematics

chunkIE: a MATLAB integral equation toolbox

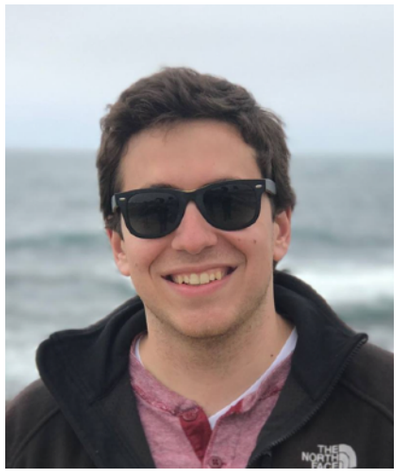
Manas Rachh
mrachh@flatironinstitute.org

The chunkers

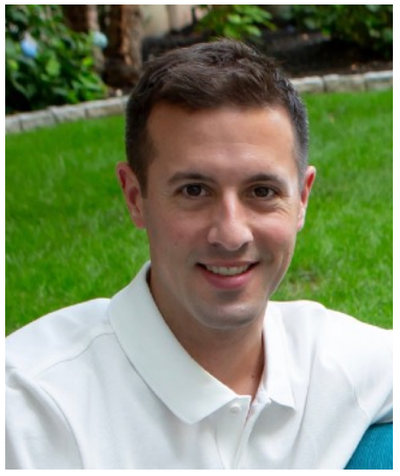


Travis Askham

Jeremy Hoskins



Dan Fortunato



Mike O’Neil

Shidong Jiang

Tristan Goodwill

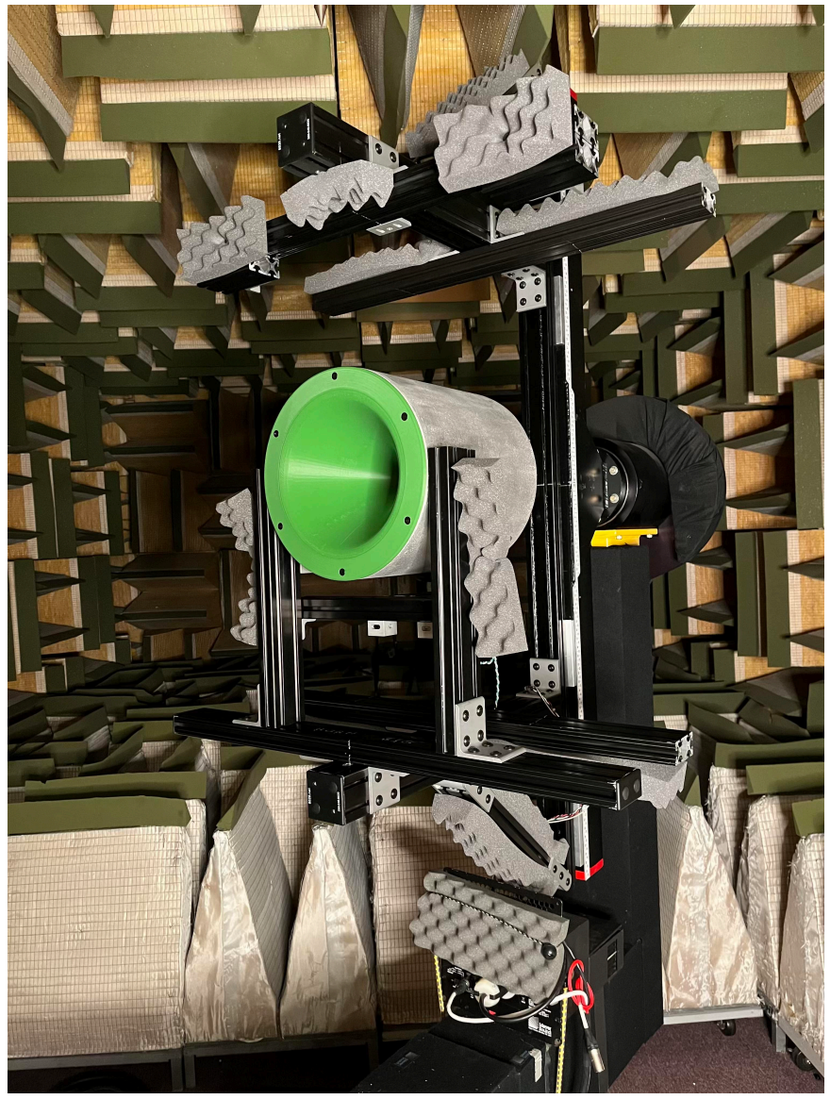
Fredrik Fryklund

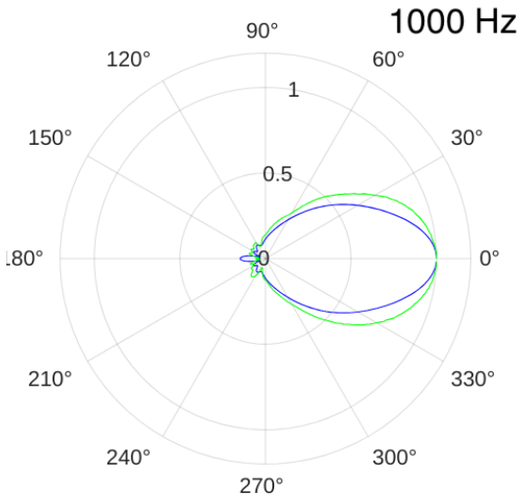
Hai Zhu

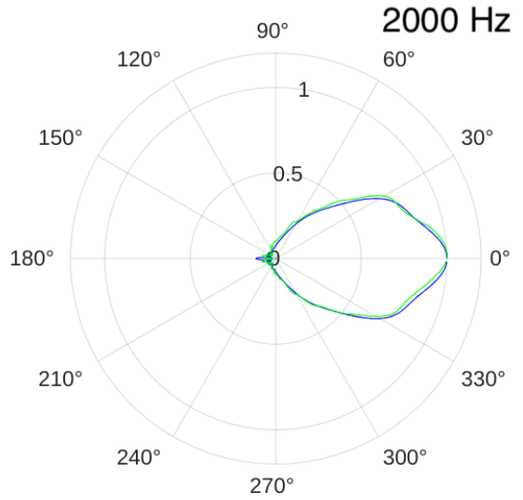
Solomon Quinn

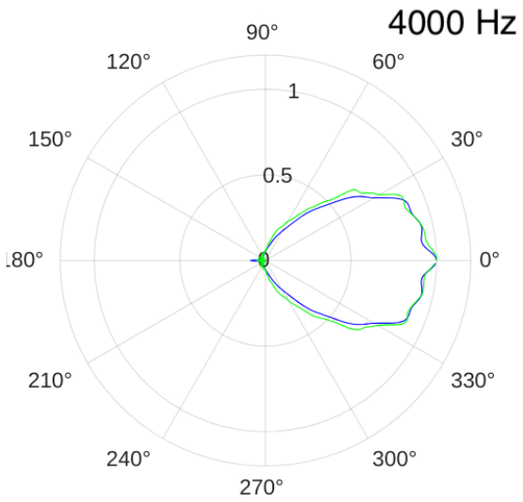
Paul Beckman

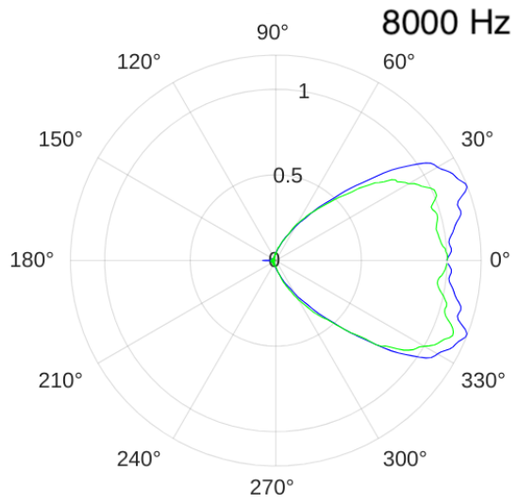
Speaker design with chunkIE











Thanks to Perrin Meyer, Meyer Sound

What is chunkIE?

A package for testing/prototyping new integral representations in two dimensions, with easy coupling to fast multipole methods and fast direct solvers for solving large scale problems.

https://chunkie.readthedocs.io

Key features:

- Flexible geometry specification: function handles, ordered set of points, graphs of curves
- Object oriented modules for easy specification of standard boundary value problems
- Coupling to fast algorithms for both direct and iterative solvers

Integral equations without the usual hassles:

- Quadrature for log singular, cauchy, hypersingular, and absolute value singular kernels
- Corner singularities/multiple junctions handled using Johan Helsing's recursively compressed inverse preconditioner
- Dense matrices upon discretization no problem: fmm2d accelerated iterative solvers, and recursive skeletonization using FLAM

Email us at mrachh@flatironinstitute.org if you'd like to join us for the next chunkIE hackathon or raise an issue on git for a feature request/bug report

Programming in chunkIE

Step 1: Define the geometry (Chunker/chunkgraph)

```
n = 10; modes = zeros(2*n + 1,1);
modes(1) = 1; modes(2*n+1) = 0.3; % set params for function handle
chnkr = chunkerfunc(@(t) chnk.curves.bymode(t, modes)); % build chunker
plot(chnkr, 'k.');
```

Step 2: Find a good integral representation for the problem

When in doubt, consult cheat sheet (https://github.com/flatironinstitute/comptools24/blob/main/IE_cheat_sheet.pdf)

Step 3: Specify kernels for integral representation

```
K = kernel('laplace', 'd');
```

Step 4: Discretize

```
A = chunkermat(chnkr, K);
A = A - 0.5*eye(size(A,1));
```

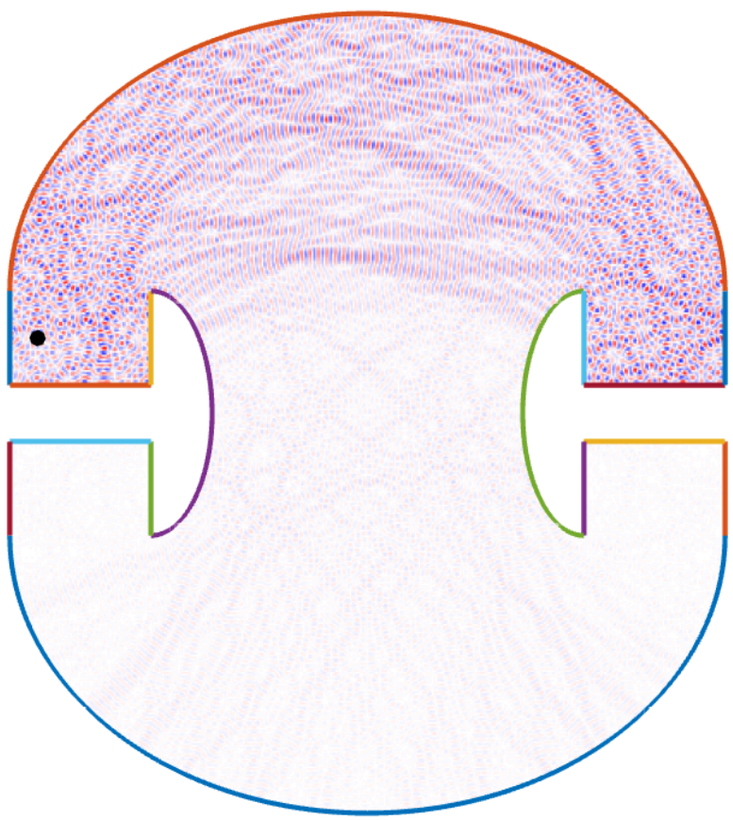
Step 5: Solve

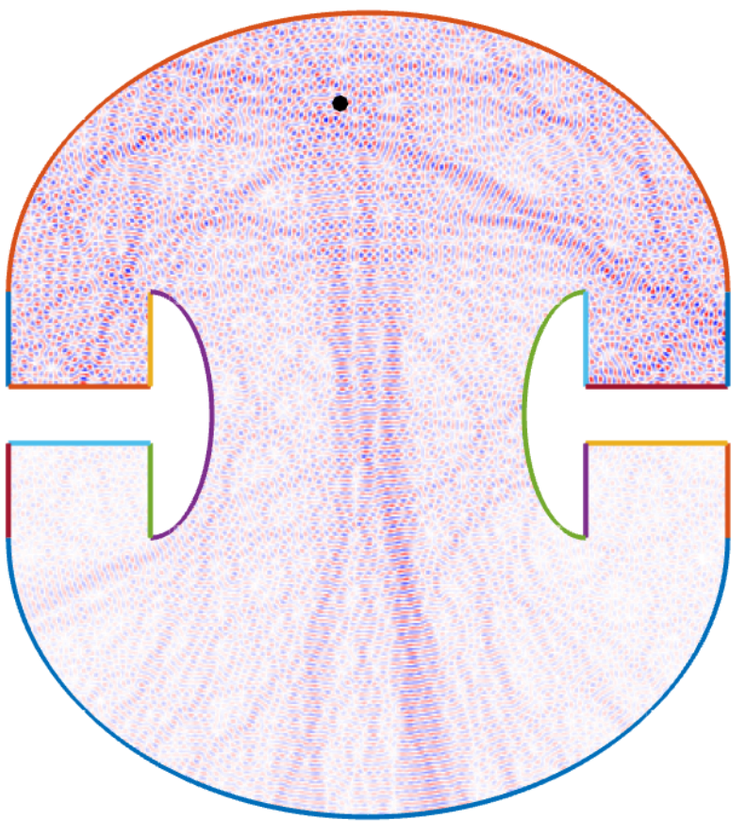
```
soln = A \ rhs;
```

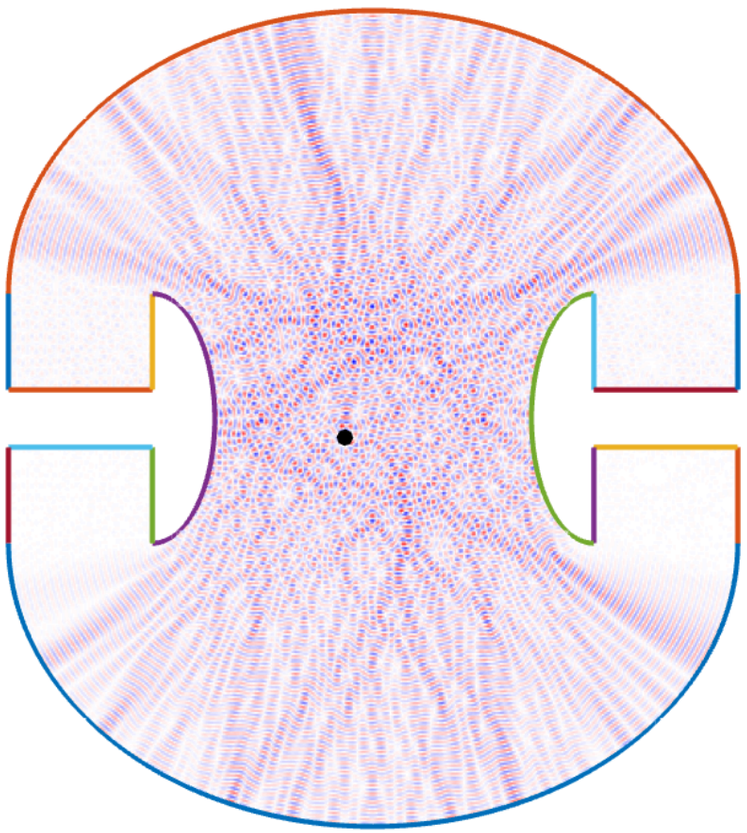
Step 6: Postprocess

```
upot = chunkerkerneval(chnkr, K, soln, targs(:,in));
```

Programming in chunkIE: the final result







Direct inversion of Integral Equations in 3D

Ignacia Fierro-Piccardo [1] Timo Betcke [1, 2] Matthew Scroggs [2]
Srhinath Kailasa [3]

[1] Department of Mathematics, University College London UK

[2]Advanced Research Computing Centre, University College London, UK

[3]Department of Engineering, University of Cambridge, UK

March 6, 2025

Our motivation

When discretizing Boundary Integral Operators we, arrive at expressions of the shape

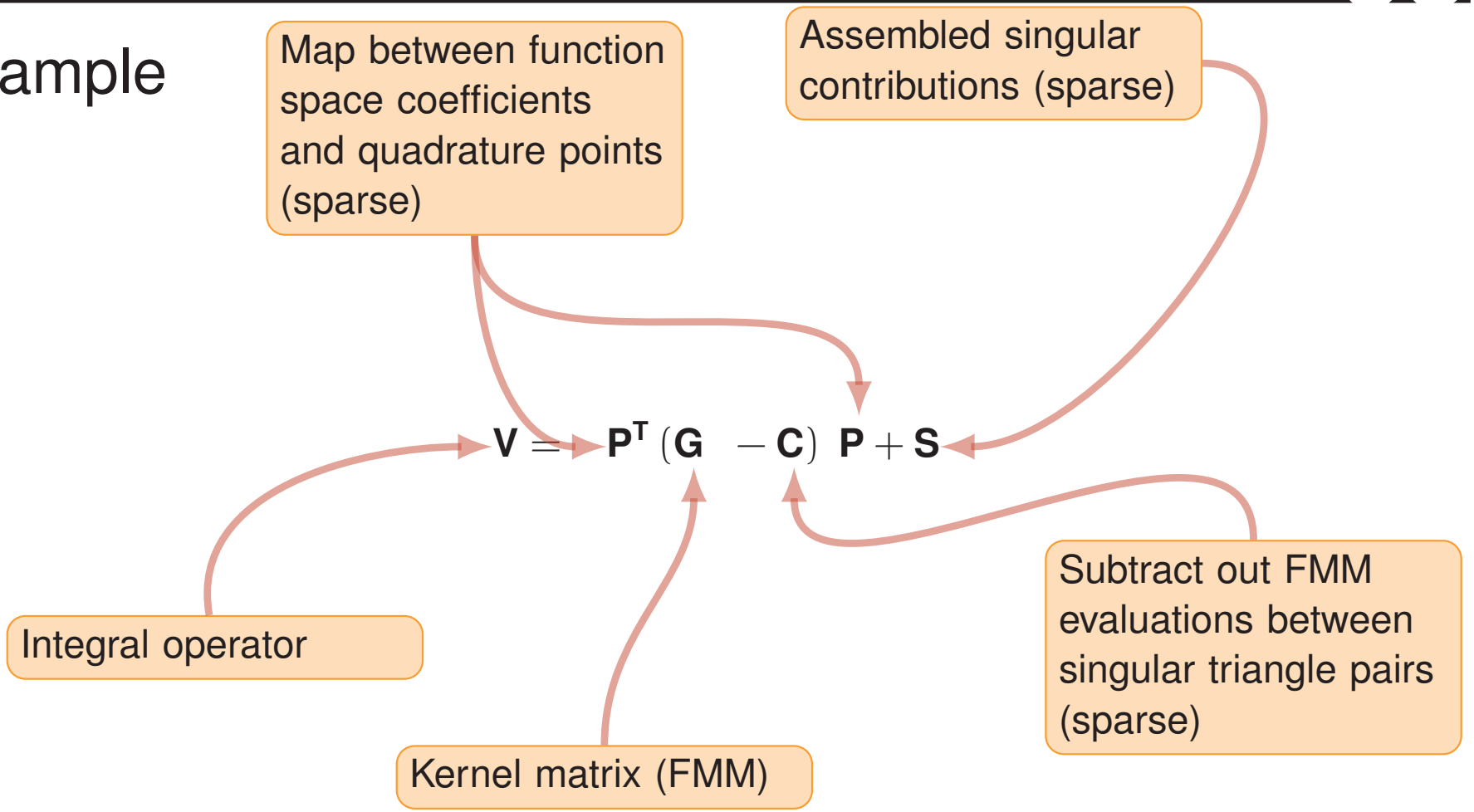
$$\mathbf{V}_\kappa[i,j] = \int_\Gamma \int_\Gamma \mathbf{G}_\kappa(\mathbf{x},\mathbf{y})\psi_j(\mathbf{y})\varphi_i(\mathbf{y})d\Gamma(\mathbf{y}), \tag{1}$$

$$\mathbf{K}_\kappa[i,j] = \int_\Gamma \int_\Gamma \nabla_{\mathbf{x}}\mathbf{G}_\kappa(\mathbf{x},\mathbf{y})\psi_j(\mathbf{y})\varphi_i(\mathbf{y})d\Gamma(\mathbf{y}), \tag{2}$$

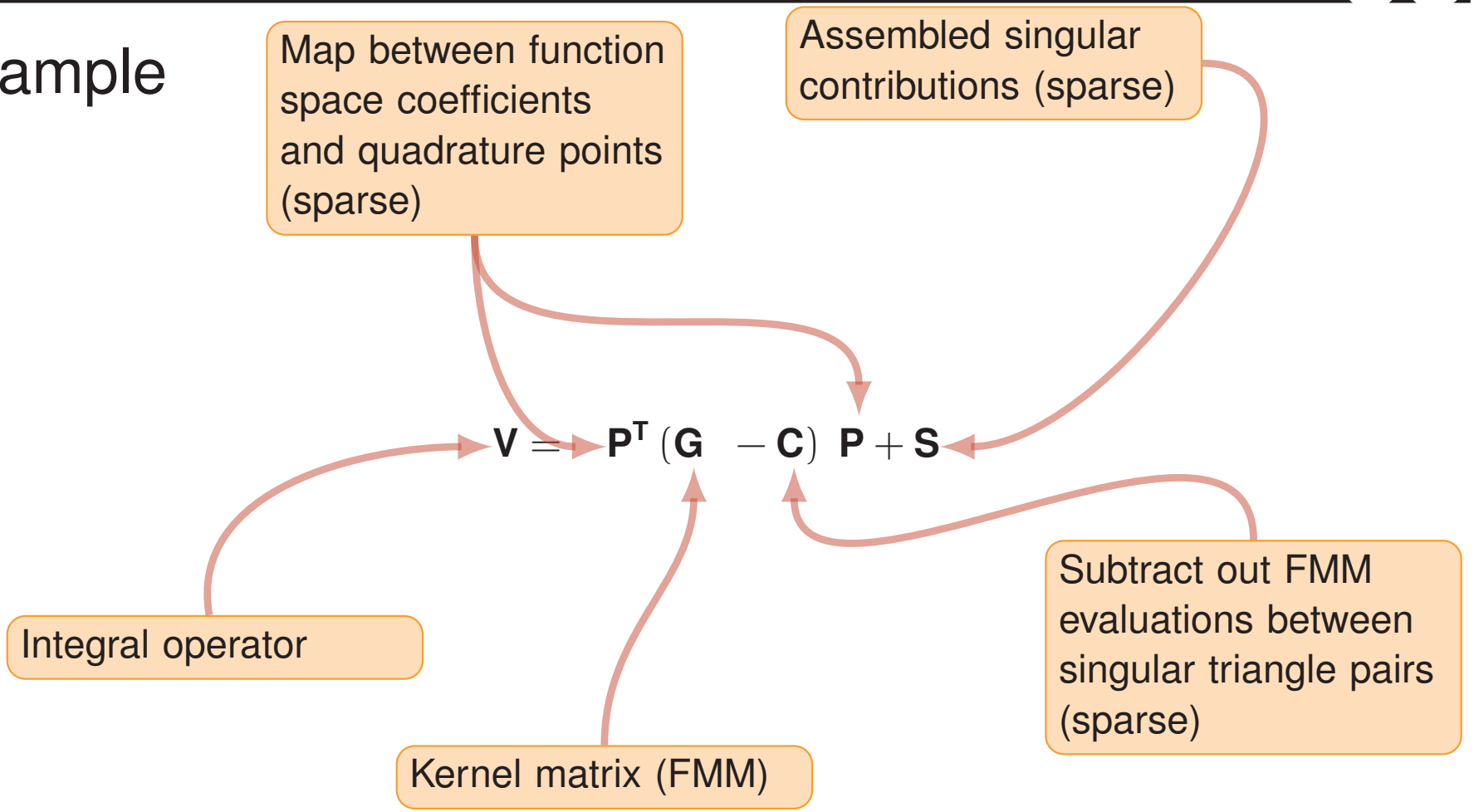
$$\text{with } \mathbf{G}_\kappa(\mathbf{x},\mathbf{y}) = \frac{e^{i\kappa\|\mathbf{x}-\mathbf{y}\|}}{4\pi\|\mathbf{x}-\mathbf{y}\|}.$$

- Dense matrices (high order of discretization and $\mathcal{O}(N^2)$ matrix-vector products).
- For large scale problems, we use FMM or H-Matrix techniques to compress interactions and reduce the complexity of matrix-vector operations to $\mathcal{O}(N\log^\alpha N)$, where $\alpha = 0, 1$.

Example

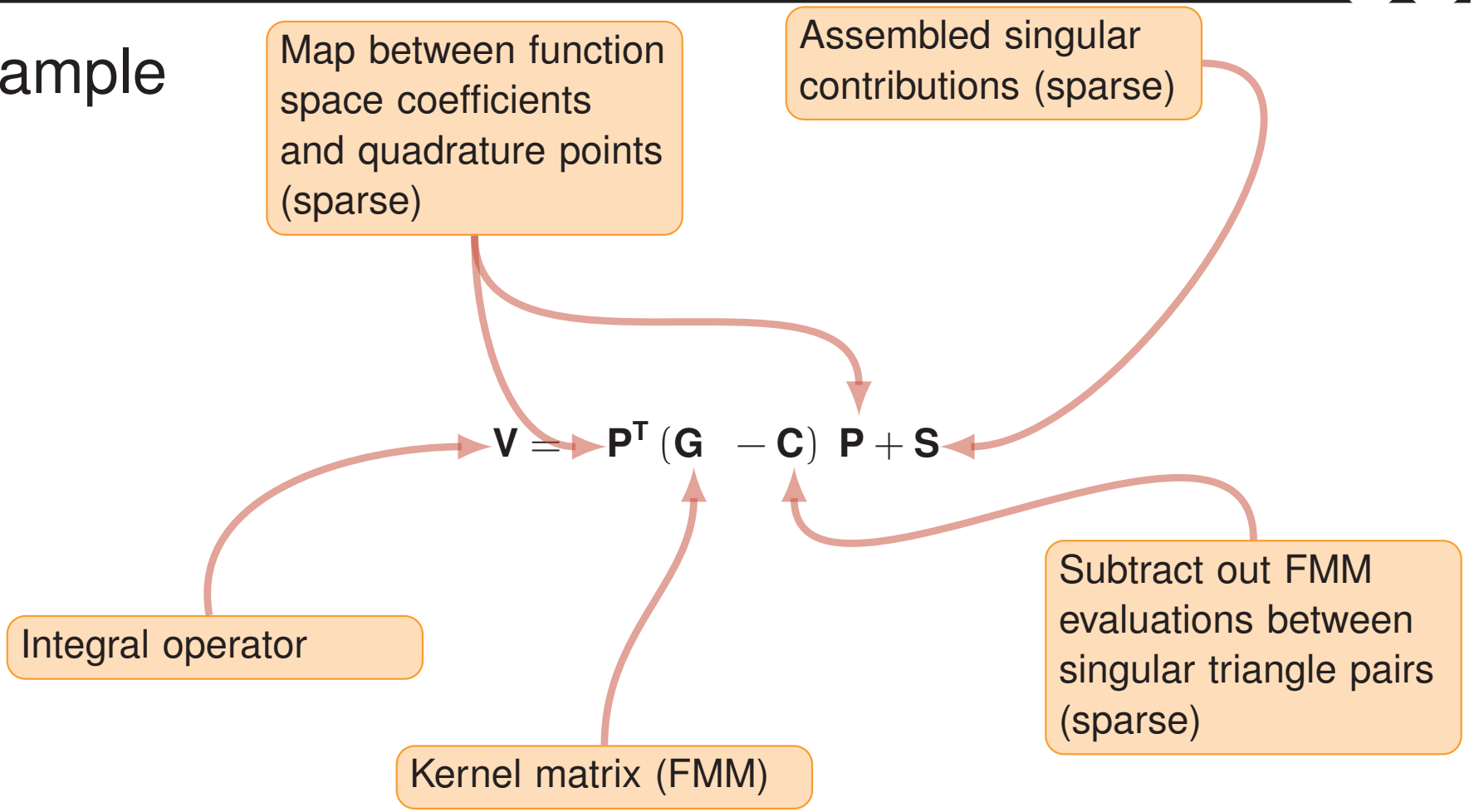

Map between function space coefficients and quadrature points (sparse)
Assembled singular contributions (sparse)
$\mathbf{V} = \mathbf{P}^T (\mathbf{G} - \mathbf{C}) \mathbf{P} + \mathbf{S}$
Integral operator
Kernel matrix (FMM)
Subtract out FMM evaluations between singular triangle pairs (sparse)

Example



- Initial results on Archer2 (Laplace kernel, **2 million** dofs on a sphere mesh): **50s pre-computation time**, **4s for each matvec on 128 Core Archer 2 node**.

Example



- Initial results on Archer2 (Laplace kernel, **2 million** dofs on a sphere mesh): **50s pre-computation time**, **4s for each matvec on 128 Core Archer 2 node**.
- What about **inverses**?

Fast assembly and fast inverses with RSRS

- To compute **V**, we have created multiple independent Rust crates:
  - `https://github.com/linalg-rs/rlst` (rust linear solver toolbox).
  - `https://github.com/bempp/kifmm` (kernel-independent fmm).
  - `/bempp/distributed-tools` (low-level data distribution and ghost communicator routines on top of MPI).
  - `/bempp/bempp-rs`, (integral operators assembler)
  - `/bempp/ndelement` (high-order basis functions and finite elements).
  - `/bempp/ndgrid` (serial and parallel grids supporting arbitrary topology and geometry dimensions).
  - `/bempp/green-kernels` (very fast SIMD accelerated Green's functions on Intel and ARM Architectures).

[1] Their publication is titled Randomized Strong Recursive Skeletonization: Simultaneous compression and factorization of $\mathcal{H}$-matrices in the Black-Box Setting.

Fast assembly and fast inverses with RSRS

- To compute **V**, we have created multiple independent Rust crates:
  - `https://github.com/linalg-rs/rlst` (rust linear solver toolbox).
  - `https://github.com/bempp/kifmm` (kernel-independent fmm).
  - `/bempp/distributed-tools` (low-level data distribution and ghost communicator routines on top of MPI).
  - `/bempp/bempp-rs`, (integral operators assembler)
  - `/bempp/ndelement` (high-order basis functions and finite elements).
  - `/bempp/ndgrid` (serial and parallel grids supporting arbitrary topology and geometry dimensions).
  - `/bempp/green-kernels` (very fast SIMD accelerated Green's functions on Intel and ARM Architectures).
- These have also allowed us to build a **preliminary implementation** of the RSRS in Rust based on the work of Anna Yesypenko and Gunnar Martinson (2023)[1].

[1] Their publication is titled Randomized Strong Recursive Skeletonization: Simultaneous compression and factorization of $\mathcal{H}$-matrices in the Black-Box Setting.

Fast assembly and fast inverses with RSRS

- To compute **V**, we have created multiple independent Rust crates:
  - `https://github.com/linalg-rs/rlst` (rust linear solver toolbox).
  - `https://github.com/bempp/kifmm` (kernel-independent fmm).
  - `/bempp/distributed-tools` (low-level data distribution and ghost communicator routines on top of MPI).
  - `/bempp/bempp-rs`, (integral operators assembler)
  - `/bempp/ndelement` (high-order basis functions and finite elements).
  - `/bempp/ndgrid` (serial and parallel grids supporting arbitrary topology and geometry dimensions).
  - `/bempp/green-kernels` (very fast SIMD accelerated Green's functions on Intel and ARM Architectures).
- These have also allowed us to build a **preliminary implementation** of the RSRS in Rust based on the work of Anna Yesypenko and Gunnar Martinson (2023)[1].
  - RSRS performs an **approximate factorisation** of an operator **A** (**K**), at the time it constructs a factorisation of an **approximate inverse** to **A** ($\mathbf{K}^{-1}$).

[1] Their publication is titled Randomized Strong Recursive Skeletonization: Simultaneous compression and factorization of $\mathcal{H}$-matrices in the Black-Box Setting.

Fast assembly and fast inverses with RSRS

- To compute **V**, we have created multiple independent Rust crates:
  - `https://github.com/linalg-rs/rlst` (rust linear solver toolbox).
  - `https://github.com/bempp/kifmm` (kernel-independent fmm).
  - `/bempp/distributed-tools` (low-level data distribution and ghost communicator routines on top of MPI).
  - `/bempp/bempp-rs`, (integral operators assembler)
  - `/bempp/ndelement` (high-order basis functions and finite elements).
  - `/bempp/ndgrid` (serial and parallel grids supporting arbitrary topology and geometry dimensions).
  - `/bempp/green-kernels` (very fast SIMD accelerated Green's functions on Intel and ARM Architectures).
- These have also allowed us to build a **preliminary implementation** of the RSRS in Rust based on the work of Anna Yesypenko and Gunnar Martinson (2023)[1].
  - RSRS performs an **approximate factorisation** of an operator **A** (**K**), at the time it constructs a factorisation of an **approximate inverse** to **A** ($\mathbf{K}^{-1}$).
  - Designed as a **black box** that needs a **matrix-vector** operation and an **octree**.

[1] Their publication is titled Randomized Strong Recursive Skeletonization: Simultaneous compression and factorization of $\mathcal{H}$-matrices in the Black-Box Setting.

Preliminary results on a unit sphere

Kernel: $\mathbf{I} + \frac{\exp(-\|\mathbf{x}-\mathbf{y}\|^2)}{\#\text{dofs}}$ (50,000 dofs)

tolerance	$\|\mathbf{K}^{-1}\mathbf{A} - \mathbf{I}\|_1$	samples
1E-3	0.04	4,031
1E-4	0.02	5,232

Kernel: $\mathbf{I} + \frac{1}{4\pi\|\mathbf{x}-\mathbf{y}\|\,\#\text{dofs}}$ (50,000 dofs)

tolerance	$\|\mathbf{K}^{-1}\mathbf{A} - \mathbf{I}\|_1$	samples
1E-3	0.25	8,452
1E-4	0.08	11,786

Preliminary results on a unit sphere

Kernel: $\mathbf{I} + \frac{\exp(-\|\mathbf{x}-\mathbf{y}\|^2)}{\#\text{dofs}}$ (50,000 dofs)

tolerance	$\|\mathbf{K}^{-1}\mathbf{A} - \mathbf{I}\|_1$	samples
1E-3	0.04	4,031
1E-4	0.02	5,232

Kernel: $\mathbf{I} + \frac{1}{4\pi\|\mathbf{x}-\mathbf{y}\|\,\#\text{dofs}}$ (50,000 dofs)

tolerance	$\|\mathbf{K}^{-1}\mathbf{A} - \mathbf{I}\|_1$	samples
1E-3	0.25	8,452
1E-4	0.08	11,786

Challenges

- Curse of dimensionality for 3D problems: large number of neighbours $\rightarrow$ large number of samples.
- The number of samples grows for larger ranks.

Preliminary results on a unit sphere

Kernel: $\mathbf{I} + \frac{\exp(-\|\mathbf{x}-\mathbf{y}\|^2)}{\#\text{dofs}}$ (50,000 dofs)

tolerance	$\|\mathbf{K}^{-1}\mathbf{A} - \mathbf{I}\|_1$	samples
1E-3	0.04	4,031
1E-4	0.02	5,232

Kernel: $\mathbf{I} + \frac{1}{4\pi\|\mathbf{x}-\mathbf{y}\|\,\#\text{dofs}}$ (50,000 dofs)

tolerance	$\|\mathbf{K}^{-1}\mathbf{A} - \mathbf{I}\|_1$	samples
1E-3	0.25	8,452
1E-4	0.08	11,786

Challenges

- Curse of dimensionality for 3D problems: large number of neighbours $\rightarrow$ large number of samples.
- The number of samples grows for larger ranks.

Advantages

- Applicable to black box operators.
- Can reach good accuracy with a number of samples smaller than the dimension of the matrix.
- Once RSRS is built, it can be applied to multiple right-hand sides.
- Parallelization is possible.

Thank you!

MUMPS: a MUltifrontal Massively Parallel Solver

A robust package using a direct method for solving

$$\mathbf{A}\ \mathbf{X} = \mathbf{B},$$

where **A** is a large sparse matrix, and **B** is dense or sparse

A free software distributed under CeCILL-C license (LGPL like)

supported by industrials and academics and co-developed by

Bordeaux Univ., CERFACS, CNRS, ENS Lyon, INPT, Inria, Mumps Tech, and Sorbonne Univ.

Latest release: MUMPS 5.7.3 July 2024

Work on fast direct solvers based on Block Low-Rank format since 2010

$\Longrightarrow$ See SIAM CSE talk of Antoine Jego, Minisymposium MS223 on Mixed and reduced precision,

"Efficient mixed-precision memory bound BLAS based on memory accessors with appl. to sparse direct solvers"

Block Low-Rank (BLR) main features and properties

- BLR is based on a flat 2D block partitioning, compatible with features of a general solver

  P. Amestoy, C. Ashcraft, O. Boiteau, A. Buttari, J.-Y. L'Excellent, and C. Weisbecker. *"Improving Multifrontal Methods by Means of Block Low-Rank Representations"*. SIAM SISC (2015).

- BLR reduces asymptotic complexity:

  Complexity reduction (3D Poisson, $n = N \times N \times N$ mesh, BLR rank bound in $O(1)$):

  $O(n^2) \rightarrow O(n^{4/3})$ flops
  $O(n^{4/3}) \rightarrow O(n \log\ n)$ memory

  P. Amestoy, A. Buttari, J.-Y. L'Excellent, and T. Mary. *"On the Complexity of the Block Low-Rank Multifrontal Factorization"*. SIAM SISC (2017).

- Multilevel BLR (MBLR) to finely control desired complexity between BLR and $\mathcal{H}$:

  2 levels: $O(n^{10/9})$ flops $\rightarrow$ 4 levels: $O(n)$ flops

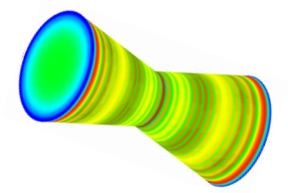
  P. Amestoy, A. Buttari, J.-Y. L'Excellent, and T. Mary. *"Bridging the gap between flat and hierarchical low-rank matrix formats: the multilevel BLR format"*. SIAM SISC (2019).

- BLR is backward stable

  N. Higham and T. Mary. *"Solving Block Low-Rank Linear Systems by LU Factorization is Numerically Stable"*. IMA J. Numer. Anal.(2021).

Block Low-Rank: flops reduction versus time performance[1]

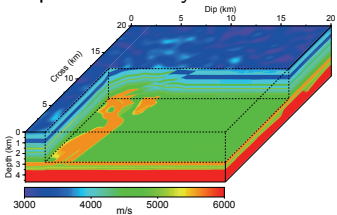
Required accuracy: 10^{-9}



Structural mechanics, $n = 8M$
Flops Ratio=17

→ **Time Ratio= 6**

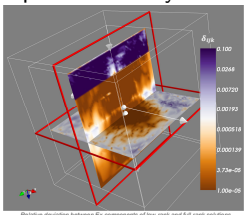
Required accuracy: 10^{-3}



Seismic imaging, $n = 17M$
Flops Ratio=27

→ **Time Ratio= 7**

Required accuracy: 10^{-7}



Electromagnetism, $n = 21M$
Flops Ratio=65

→ **Time Ratio=19**

Converting flops reduction into performance gains in not straightforward!

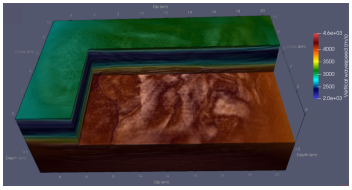
P. Amestoy, A. Buttari, J.-Y. L'Excellent, and T. Mary. *"Performance and Scalability of the Block Low-Rank Multifrontal Factorization on Multicore Architectures"*. ACM TOMS (2019).

[1] *where Ratio corresponds to Full Rank / BLR*

BLR with Mixed Precision on Full-Waveform Inversion (WIND team)

- Adastra computer (French national computing center)
- Application: Gorgon Model, reservoir 23km x 11km x 6.5km, Helmholtz equation $\Rightarrow$ Complex matrix, 531 Million dofs
- Full Rank cost: flops for one LU factorization$= 2.6 \times 10^{18}$
- Use of BLR with Mixed Precision:

Amestoy, Boiteau, Buttari, Gerest, Jézéquel, L'Excellent, Mary. *"Mixed precision low-rank approximations and their application to BLR LU factorization"*. IMA J. Numer. Anal.(2023).



(25-Hz Gorgon FWI velocity model)

FR (Full Rank); BLR with $\varepsilon_{blr} = 10^{-5}$; 48 000 cores (500 MPI $\times$ 96 threads/MPI)

FR: fp32; Mixed precision BLR: 3 precisions (32bits, 24bits, 16bits) for storage

LU size (TBytes)			Flops		Time BLR + Mixed (sec)			Scaled Resid.
FR	BLR	+mixed	FR	BLR+mixed	Analysis	Facto	Solve	BLR+mixed
73	34	26	2.6×10^{18}	0.5×10^{18}	446	5500	27	7×10^{-4}

in practice: hundreds to thousands of Solve steps (sparse right-hand sides (sources))

Operto et al. *"Pushing the limits of 3D frequency-domain FWI with the 2015/2016 OBN Gorgon dataset"*. EAGE 2024 conference.

Berkeley High Performance Hierarchical Matrix Software Suite

Lead developers: Yang Liu, Pieter Ghysels, Sherry Li

ButterflyPACK

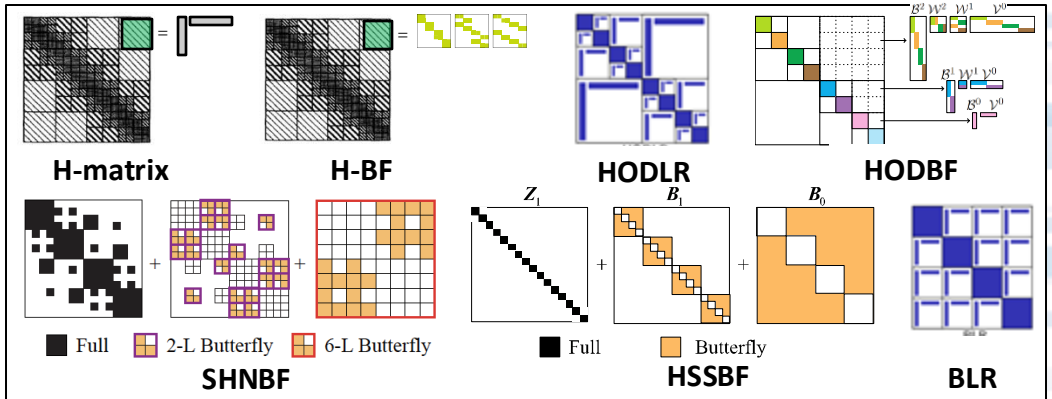
Fast PDE solvers and transforms

Hierarchical matrix and tensor algorithms for compressing integral equation operators, kernel matrices, and integral transforms.

- **Core Functionality:** fast preconditioners and direct solvers for dense linear systems, and fast oscillatory integral transforms

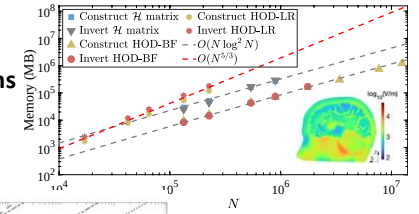
- **Dense Matrix Solvers using Hierarchical Approximations**
  - Hierarchical partitioning, low-rank approximations
  - Hierarchical-LU (H-matrix), H LU with butterfly (H-BF), Hierarchically Off-Diagonal Low-Rank (HODLR), Hierarchically Off-Diagonal Butterfly (HODBF), Block Low-Rank (BLR), Strongly Admissible and Hierarchically Nested Butterfly (SHNBF), Hierarchically Semi-Separable Matrix with Butterfly (HSS-BF).
  - Fortran2008, with C++ interface, MPI (optional)+OpenMP
  - Applications: BEM, Volume IE, kernel & covariance matrices, integral transforms...

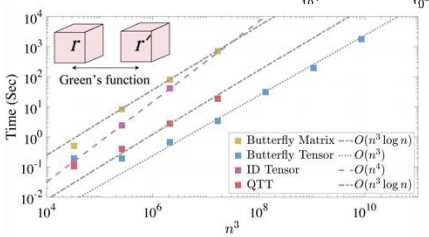
- **Dense Tensor Compression Algorithms**
  - Tucker-like interpolative decomposition, tensor butterfly, and quantized tensor train (QTT) (in progress)
  - Fortran2008, with C++ interface, MPI (optional)+OpenMP
  - Applications: High-dimensional integral transforms



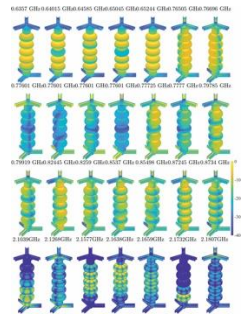
Hierarchical matrix algorithms in ButterflyPACK

Memory scaling of volume IE for biomedical applications using ButterflyPACK





Tensor algorithms in ButterflyPACK: Tucker, tensor butterfly and QTT (in progress)



Resonance modes of cavity solved by surface IEs using ButterflyPACK

https://portal.nersc.gov/project/sparse/butterflypack

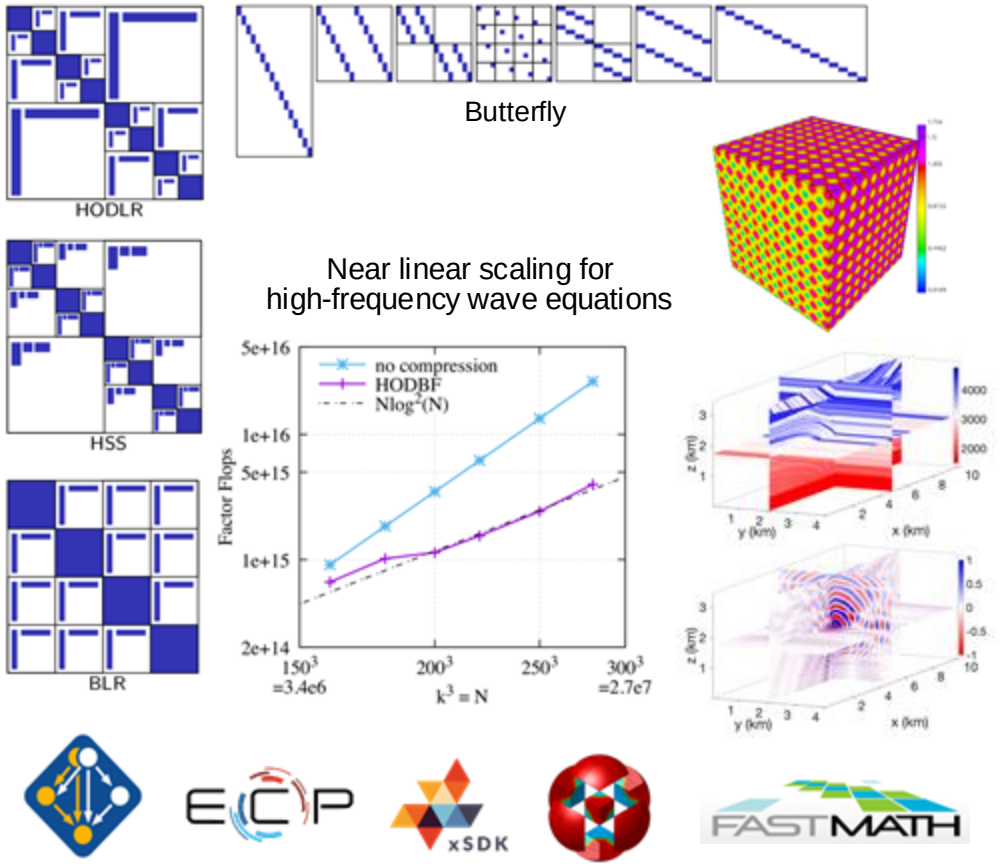
STRUMPACK
Structured Matrix Package

STRUMPACK

Hierarchical solvers for dense rank-structured matrices and fast algebraic sparse solver and robust and scalable preconditioners.

BERKELEY LAB

- **Dense Matrix Solvers using Hierarchical Approximations**
  - Hierarchical partitioning, low-rank approximations
  - Hierarchically Semi-Separable (HSS), Hierarchically Off-Diagonal Low-Rank (HODLR), Hierarchically Off-Diagonal Butterfly (HODBF), Block Low-Rank (BLR), Butterfly
  - C++ Interface to ButterflyPACK (Fortran)
  - Applications: BEM, Cauchy, Toeplitz, kernel & covariance matrices, ...
  - Asymptotic complexity much lower than LAPACK/ScaLAPACK routines
- **Sparse Direct Solver**
  - Algebraic sparse direct solver
  - GPU: CUDA, HIP/ROCm, DPC++ (in progress)
  - Orderings: (Par)METIS, (PT)Scotch, RCM
- **Preconditioners**
  - Approximate sparse factorization, using hierarchical matrix approximations
  - Scalable and robust, aimed at PDE discretizations, indefinite systems, …
  - Iterative solvers: GMRES, BiCGStab, iterative refinement
- **Software**
  - BSD license
  - Interfaces from PETSc, MFEM, Trilinos, available in Spack



github.com/pghysels/STRUMPACK

Hands-on exercises

- Pre-installed software at Frank system at University of Oregon
- Instructions for login and run the example programs
  - Follow the instructions at http://bit.ly/415imnR
  - Choose a username and password from the "**Accounts Assignment**" table
  - Connect to the login node, then connect to one of the available compute nodes (saturn is recommended as it supports all the tests)
  - Go to the directory ACES2024_Tutorial_Direct_Solvers/[compute node name]/manual_install
  - Use run.sh following the documentation to run the ButterflyPACK and STRUMPACK tests.

- Contacts
  - Yang Liu, liuyangzhuan@lbl.gov
  - Sherry Li, xsli@lbl.gov