Randomized sampling of structured matrices

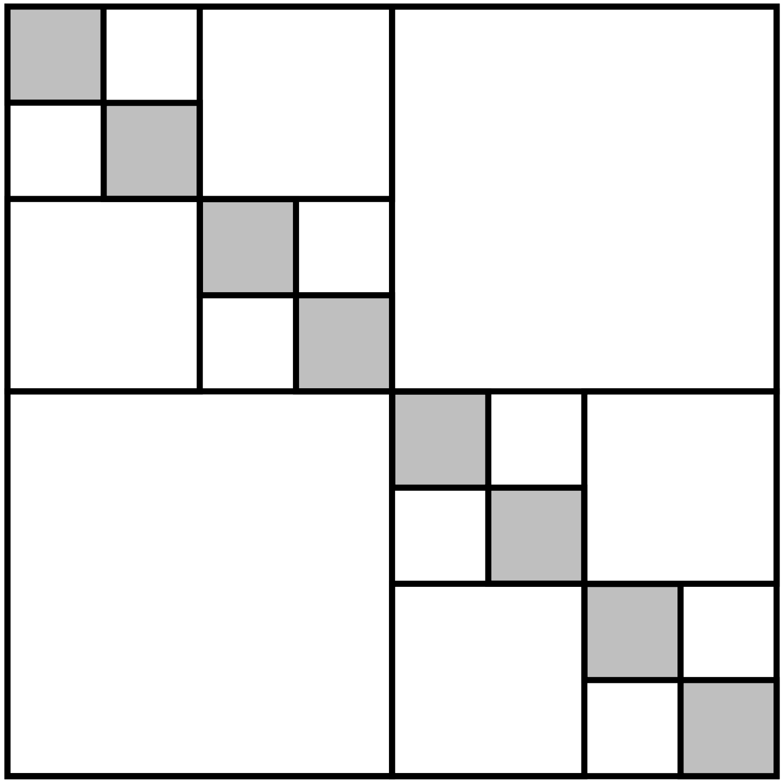
Gunnar Martinsson, The University of Colorado at Boulder

What is a "structured matrix"?

The answer depends on whom you ask . . .

For our purposes, we will say that a matrix is structured if it can be tessellated into blocks in such a way that each block is of numerically low rank.

The following figure shows one of the most common tessellations:



When a matrix has this kind of "structure", we can exploit it to accelerate a range of linear algebraic operations:

- Matrix-vector multiplies. (As in FMM, Barnes-Hut, panel clustering, *etc.*)
- Matrix-matrix multiplies.
- Matrix factorizations (QR, LU, Cholesky, *etc*).
- Matrix inversions.
- Eigenvalue decompositions, SVDs — in particular partial ones.

Structured matrices are ubiquitous in scientific computing:

- Numerical methods for (more or less) elliptic PDEs.
- Signal processing.
- Inverses of discrete Laplacians?

Example 1 — Integral equation methods for elliptic PDEs:

Essentially all integral operators (single layers, double layers, etc) of classical potential theory turn into structured matrices upon discretization.

Discretizations of Green's functions on bounded domains are structured matrices.

Approximations of Dirichlet-to-Neumann operators.

Etc, etc, . . .

Structured matrices abound in this environment.

Example 2 — Toeplitz matrices:

Let $\ldots,\ a_{-2},\ a_{-1},\ a_0,\ a_1,\ a_2,\ \ldots$ be complex numbers.

For a positive integer N, let A denote the $N \times N$ matrix with entries

$$A_{ij} = a_{j-i}.$$

For instance, for $N = 4$, we get

$$A = \begin{bmatrix} a_0 & a_1 & a_2 & a_3 \\ a_{-1} & a_0 & a_1 & a_2 \\ a_{-2} & a_{-1} & a_0 & a_1 \\ a_{-3} & a_{-2} & a_{-1} & a_0 \end{bmatrix}.$$

Then the Fourier transform of A,

$$\hat{A} = F_N\, A\, F_N^*$$

is a structured matrix. (A itself is not necessarily "structured".)

Example 3 — Sampling of “sinc” matrix:

Let $\{x_j\}_{j=1}^{N}$ denote real numbers such that $-\infty < x_1 < x_2 < \cdots < x_{N-1} < x_N < \infty$.

Define for a real positive number c the $N \times N$ matrix A by

$$A_{ij} = \begin{cases} c & i = j, \\ \dfrac{\sin\bigl(c\,(x_i - x_j)\bigr)}{x_i - x_j}, & i \neq j. \end{cases}$$

Then A is a structured matrix.

Recall: Prolate spheroidals are eigenfunctions of the sinc operator — Hong’s talk.

Example 4 — Inverse of discrete Laplacian on a square uniform grid:

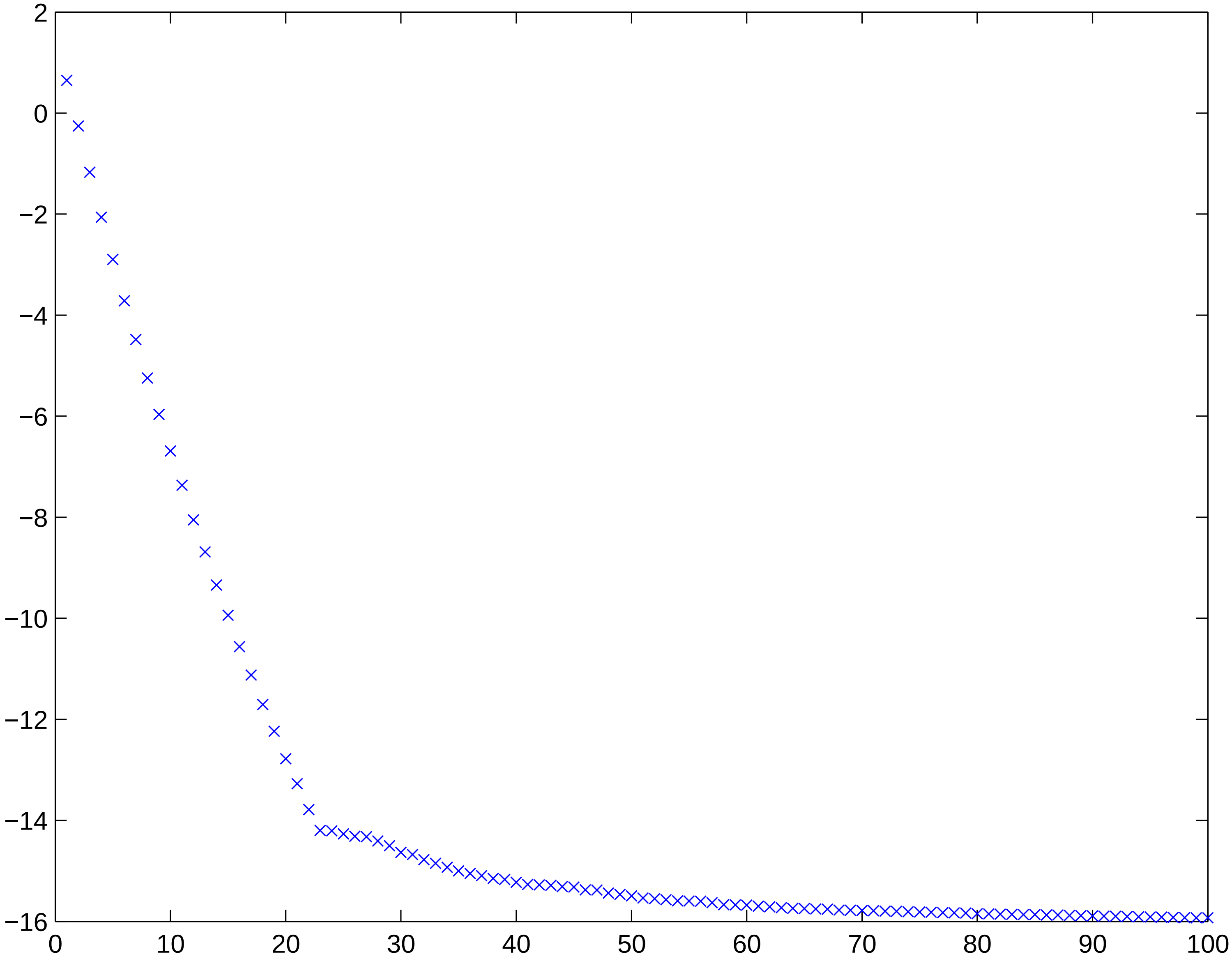
Let L be the standard five-point stencil (discrete Laplacian) on a 50×50 grid:

$$L = \begin{bmatrix} C & -I & 0 & 0 & \cdots \\ -I & C & -I & 0 & \cdots \\ 0 & -I & C & -I & \cdots \\ \vdots & \vdots & \vdots & \vdots & \end{bmatrix} \qquad C = \begin{bmatrix} 4 & -1 & 0 & 0 & \cdots \\ -1 & 4 & -1 & 0 & \cdots \\ 0 & -1 & 4 & -1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \end{bmatrix}.$$

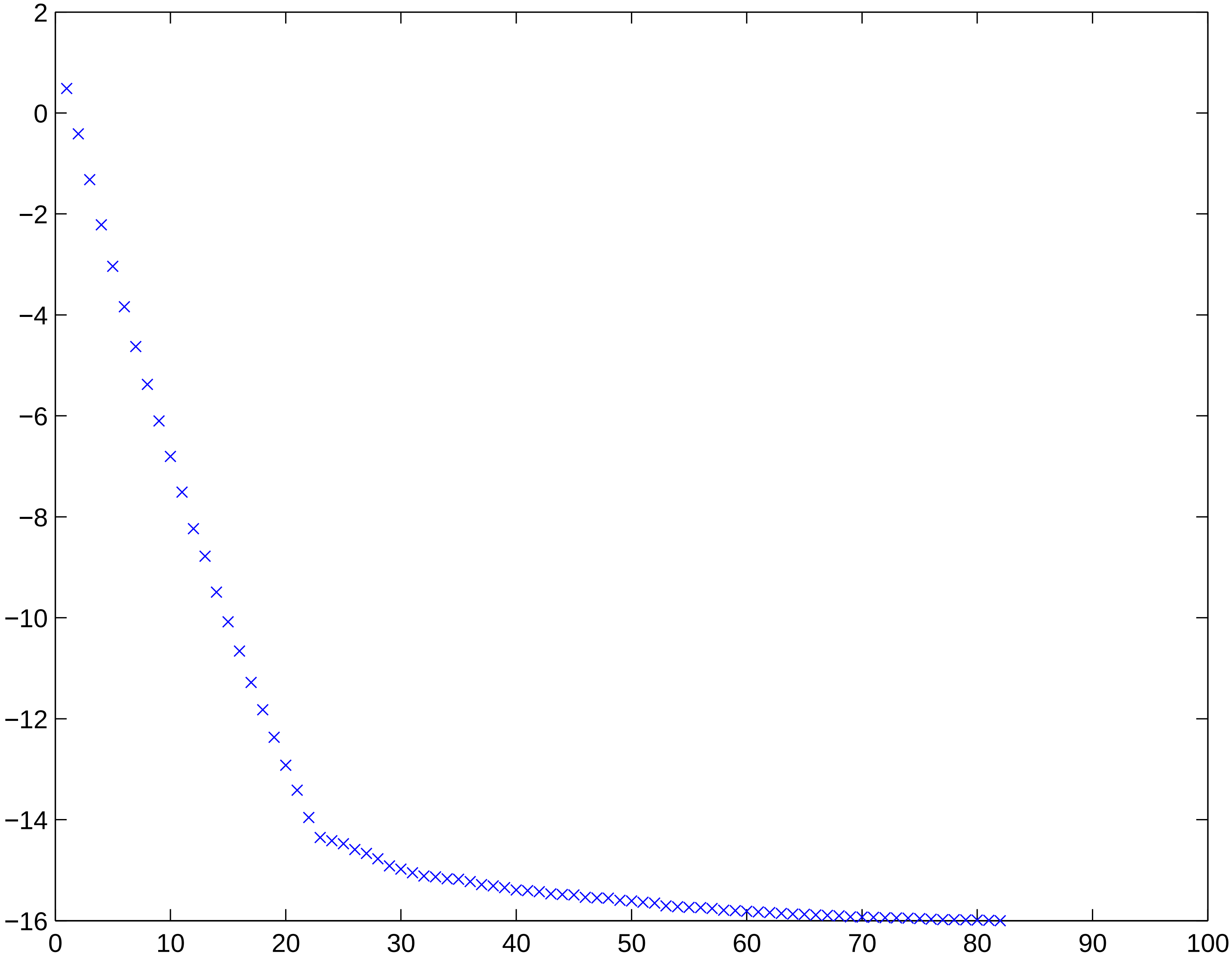
Let A be the inverse of L, and partition it:

$$A = L^{-1} = \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix}.$$

We consider the 625×625 submatrix A_{14} of the $2\,500 \times 2\,500$ matrix A.



The 10-logarithm of the singular values of A_{14}.



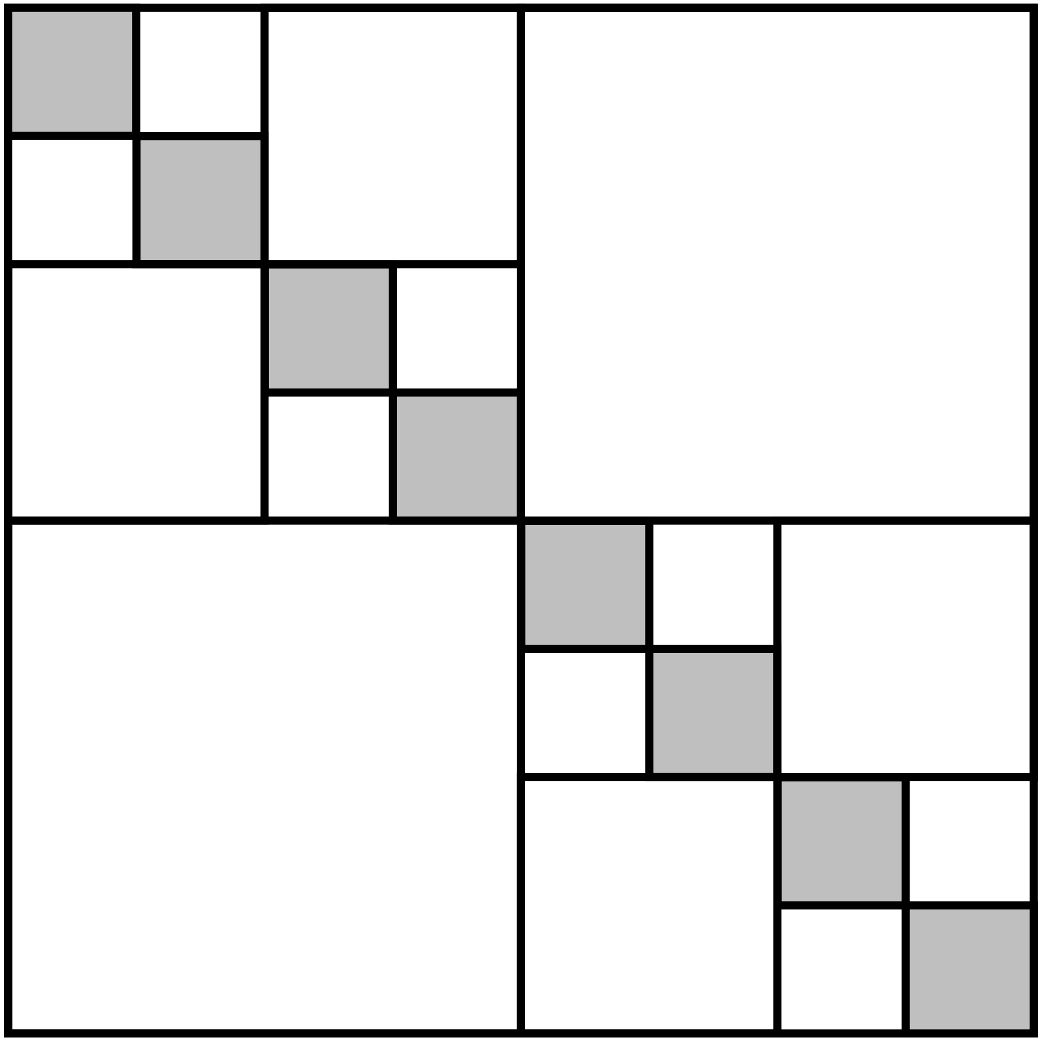
The 10-logarithm of the singular values of A_{14} — now with *random coefficients*.

Example 5 — More general incarnations of discrete Laplacians . . . ?

Important: Many of the structured matrices that arise in applications have complicated tessellation structures.

[Shiv's example]

For simplicity, we will be concerned only with the most basic tessellation:



(In honesty, there are reasons for this beyond just notational simplicity. . .)

Fast matrix-vector multiply for a low-rank matrix:

Let A be an $N \times N$ matrix with a rank-k approximate factorization,

$$A = U\,V^{\rm t}.$$

and let x be an $N \times 1$ vector.

Computing $A\,x$ directly requires N^2 flops.

Instead, use that $A\,x = U\,(V^{\rm t}\,x)$.
Evaluating $y = V^{\rm t}\,x$ costs $N\,k$ flops.
Evaluating $x = U\,y$ costs $N\,k$ flops.
Total cost is $2\,N\,k$ flops.

Fast matrix-vector multiply for a structured matrix:

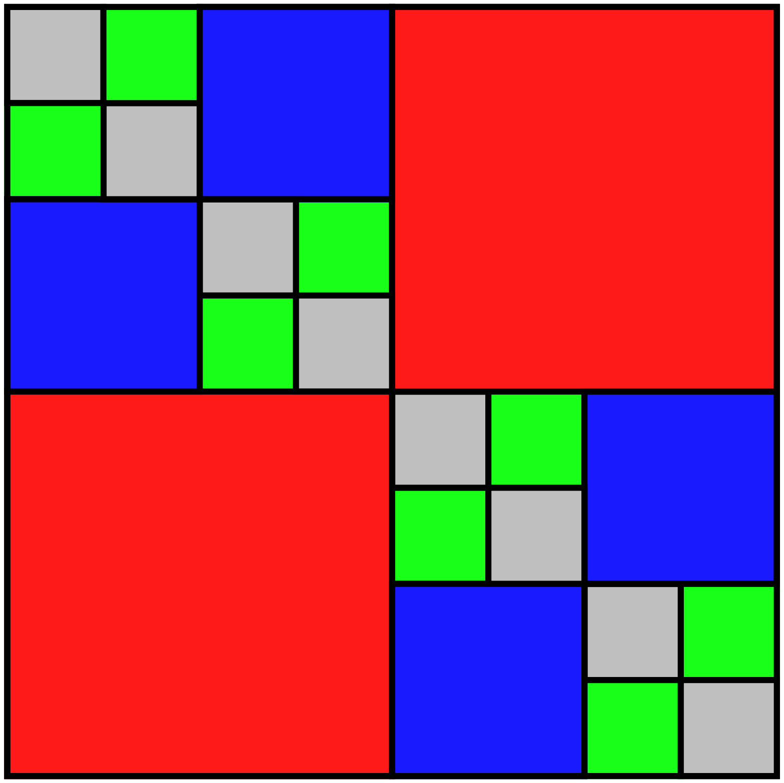
Suppose that each block has rank at most k.

At level p, there are 2^p blocks of size $N/2^p$ so the cost is

$$T_p = 2^p \; 2 \; \frac{N}{2^p} \; k = 2\,N\,k.$$

There are $\log(N)$ levels, so the total cost is

$$T_{\text{total}} = 2\,k\,N\;\log(N).$$



Level 1 blocks are red.

Level 2 blocks are blue.

Level 3 blocks are green.

Fast matrix-matrix multiply for a structured matrix:

Suppose that A and B are two structured matrices with the same block structure.

$$A\,B = \left[\begin{array}{cc} A_{11} & A_{12} \\ A_{21} & A_{22} \end{array}\right] \left[\begin{array}{cc} B_{11} & B_{12} \\ B_{21} & B_{22} \end{array}\right] = \left[\begin{array}{cc} A_{11}\,B_{11} + A_{12}\,B_{21} & A_{11}\,B_{12} + A_{12}\,B_{22} \\ A_{21}\,B_{11} + A_{22}\,B_{21} & A_{21}\,B_{12} + A_{22}\,B_{22} \end{array}\right].$$

The matrices A_{11}, A_{22}, B_{11}, B_{22} are themselves "structured".

The green products involve two rank-k matrices.

The blue products involve one rank-k matrix and one structured matrix.

The red products involve two structured matrices.

Recurse!

Let T_N denote the cost for multiplying two $N \times N$ structured matrices. Then

$$T_N = 2\,T_{N/2} + \alpha\,k\,(N/2)\,\log(N/2).$$

It follows that

$$T_N \sim k\,N\,\log(N)^2.$$

Fast matrix inversion for a structured matrix:

Suppose that A is a structured matrix.

Recall that

$$A^{-1} = \left[\begin{array}{cc} A_{11} & A_{12} \\ A_{21} & A_{22} \end{array}\right]^{-1} = \left[\begin{array}{cc} X_{11} + X_{11}\,A_{12}\,X_{22}\,A_{21}\,X_{11} & -X_{11}\,A_{12}\,X_{22} \\ -X_{22}\,A_{21}\,X_{11} & X_{22} \end{array}\right],$$

where

$$\begin{aligned} X_{11} =& A_{11}^{-1}, \\ X_{22} =& \big(A_{22} - A_{21}\,A_{11}^{-1}\,A_{12}\big)^{-1} = \big(A_{22} - A_{21}\,X_{11}\,A_{12}\big)^{-1}. \end{aligned}$$

Again, if T_N is the cost to invert an $N \times N$ structured matrix, then

$$T_N = 2\,T_{N/2} + \alpha\,k\,(N/2)\,\log(N/2),$$

so

$$T_N \sim k\,N\,log(N)^2.$$

Fast (partial) spectral decomposition of a structured matrix:

Recall shifted inverse iteration for a matrix A:

- Let $\{\lambda, v\}$ be an estimate for an eigenpair for A.
- Construct a new estimate for the eigenvector via

$$v' = \frac{(A - \lambda I)^{-k}\, v}{||(A - \lambda I)^{-k}\, v||}.$$

- Construct a new estimate for the eigenvalue via

$$\lambda' = (v')^{\mathrm{t}}\, A\, v'.$$

- Repeat!

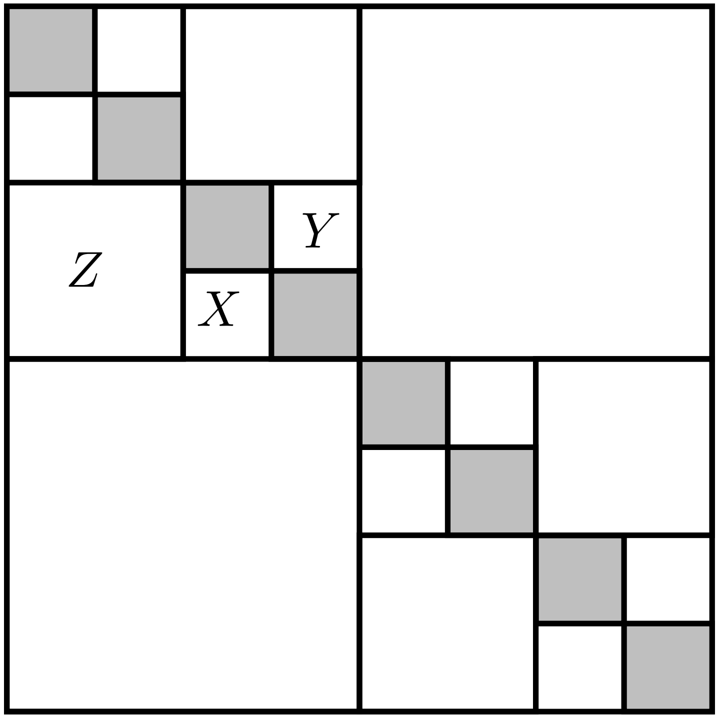
Now suppose that:

1. You can (fairly) quickly compute an estimate for $(A - \lambda I)^{-1}$.
2. You can (very) quickly update $(A - \lambda I)^{-1}$ to construct $(A - \lambda' I)^{-1}$.
3. You have a fast matrix-vector multiplier for A.

You could then perform power iteration pretty darn fast . . .

Getting rid of the "$\log(N)$" factors.

Use "nested" basis functions.



Let X, Y, and Z be the blocks marked in the figure, and suppose that

$X = U_X\,\hat{X}$,

$Y = U_Y\,\hat{Y}$,

$Z = U_Z\,\hat{Z}$,

for some matrices U_X, U_Y, and U_Z that each have k orthonormal columns. We say that the basis matrices are "nested" if

$$U_Z = \begin{bmatrix} U_X & 0 \\ 0 & U_Y \end{bmatrix} \hat{U}_Z,$$

for some $2k \times k$ matrix $\hat{U}_X$. Then

$$U_Z^{\rm t}\, f = \hat{U}_Z^{\rm t} \begin{bmatrix} U_X^{\rm t} & 0 \\ 0 & U_Y^{\rm t} \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = \hat{U}_Z^{\rm t} \begin{bmatrix} U_X^{\rm t}\, f_1 \\ U_Y^{\rm t}\, f_2 \end{bmatrix}.$$

So $U_Z^{\rm t}\, f$ can cheaply be computed from $U_X^{\rm t}\, f_1$ and $U_Y^{\rm t}\, f_2$.

When "nested" bases are used, we get "wavelet-like" algorithms.

```
Construct the expansion of a function $f$ in all bases at the finest level:
loop over all boxes $\tau$ at the finest level
    $f_\tau = U_\tau^{\rm t}\, f(I_\tau)$
end loop

Recursively construct the expansion of $f$ in the bases at the higher levels:
loop over levels, finer to coarser, $p = P-1,\, P-2,\, \dots,\, 1$
    loop over all boxes $\tau$ on level $p$
        Let $\sigma_1$ and $\sigma_2$ denote the sons of $\tau$.
        $f_\tau = \hat{U}_\tau^{\rm t} \begin{bmatrix} f_{\sigma_1} \\ f_{\sigma_2} \end{bmatrix}$
    end loop
end loop
```

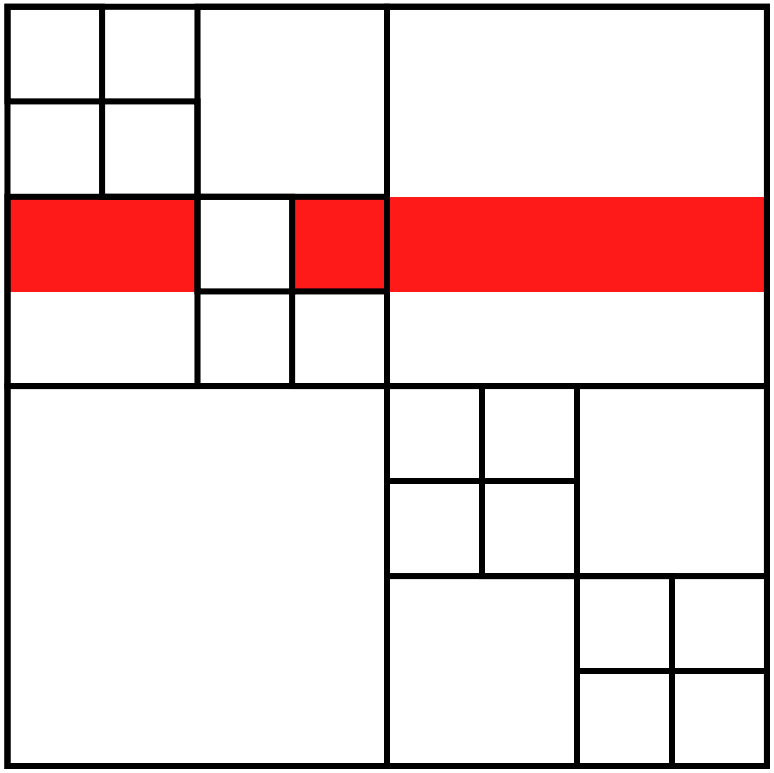
The total cost is $O(k\,N)$ instead of $O(k\,N\,\log(N))$.

OK, so nested bases are great, but how do you construct them?

Remark 1: Not all structured matrices have nested bases.

Remark 2: Simply compressing the small matrices first will not work.

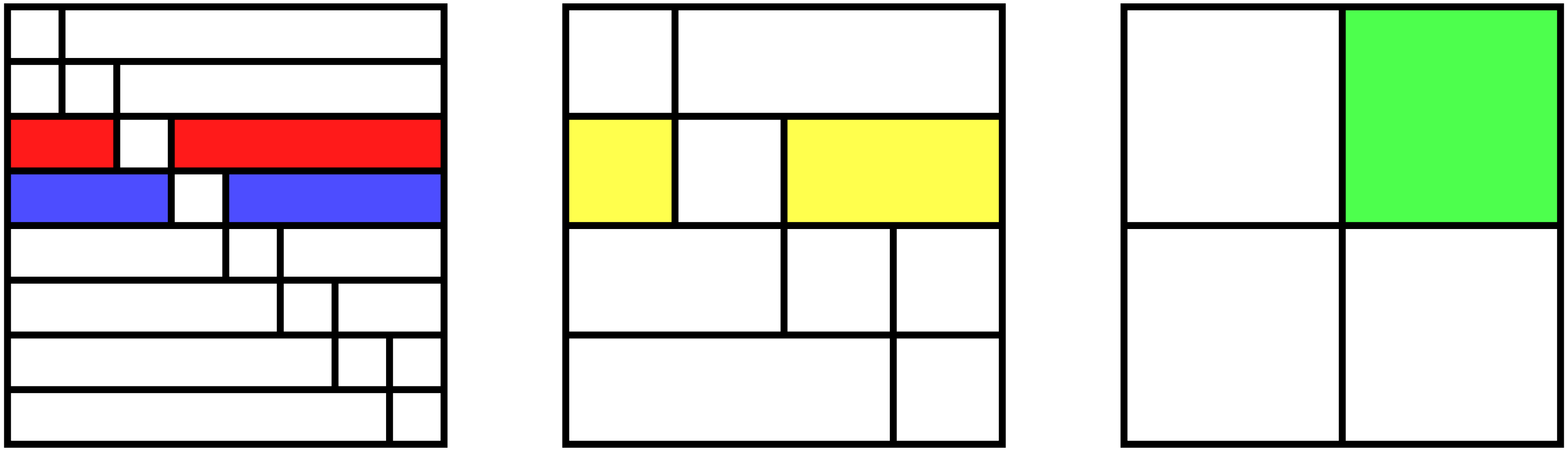
What you (at least in principle) need to do is to compress blocks like this one:



This leads us to what Gu and Chandrasekaran call “Hierarchically Semi-Separable” (HSS) matrices.

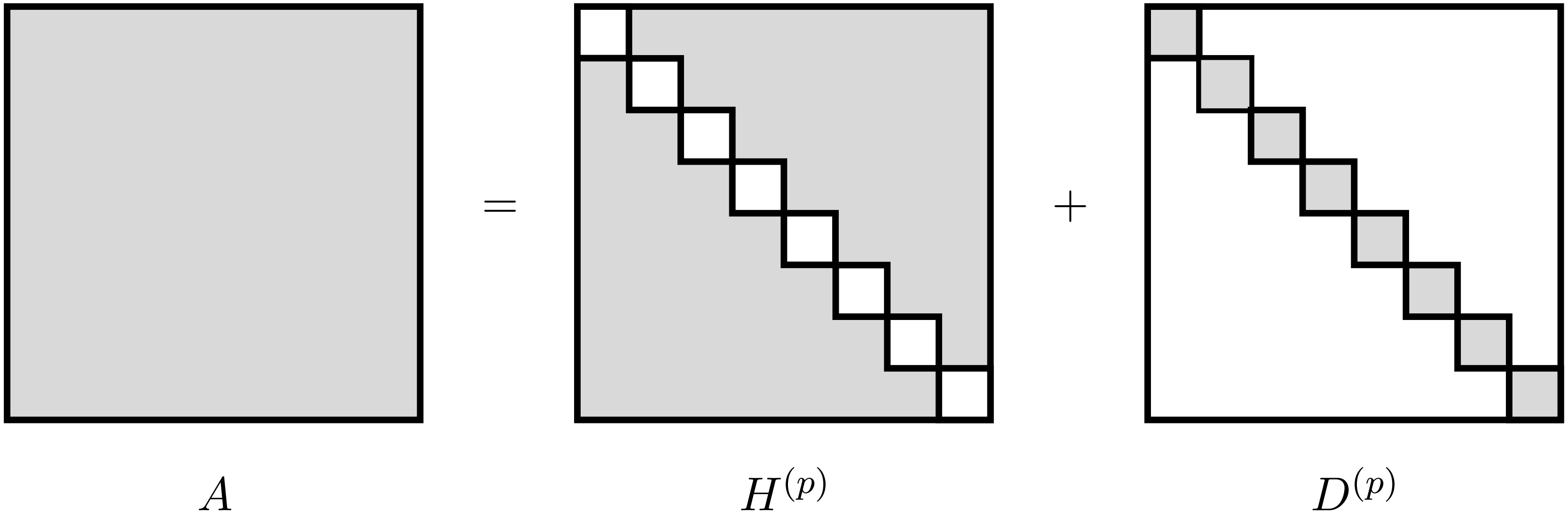
An HSS matrix is one for which all “HSS blocks” are rank deficient. (For simplicity, we assume that they all have the same ε-rank k.)

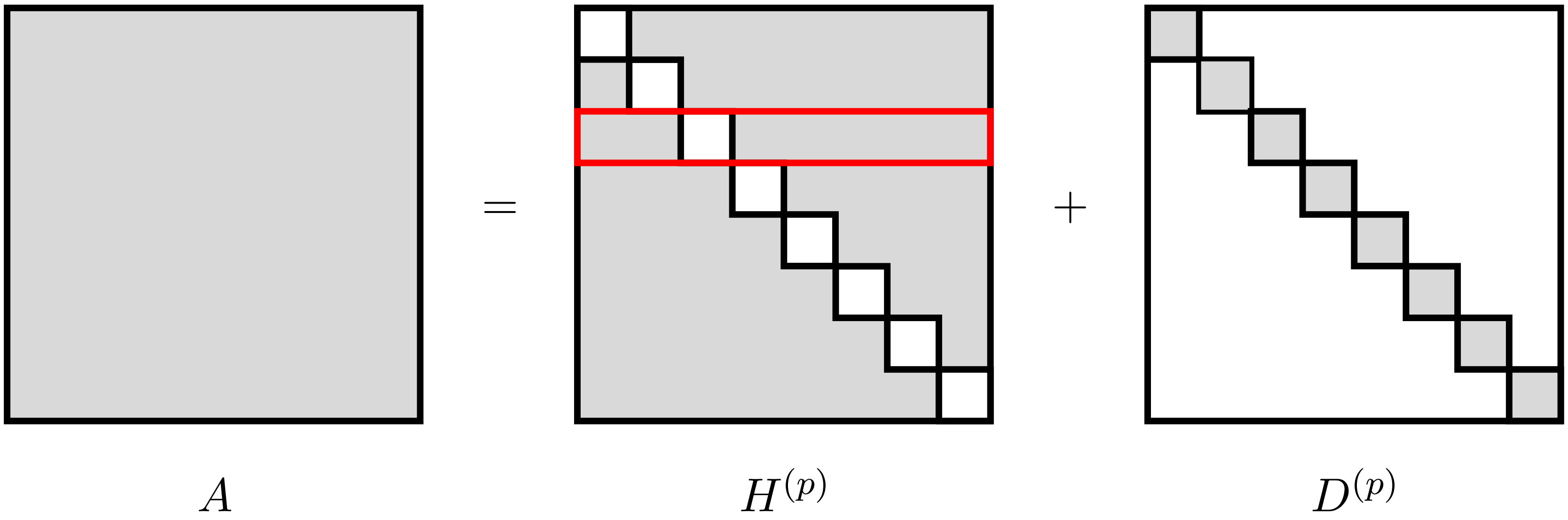
Illustration of “HSS blocks” at various levels:



Level 3 Level 2 Level 1

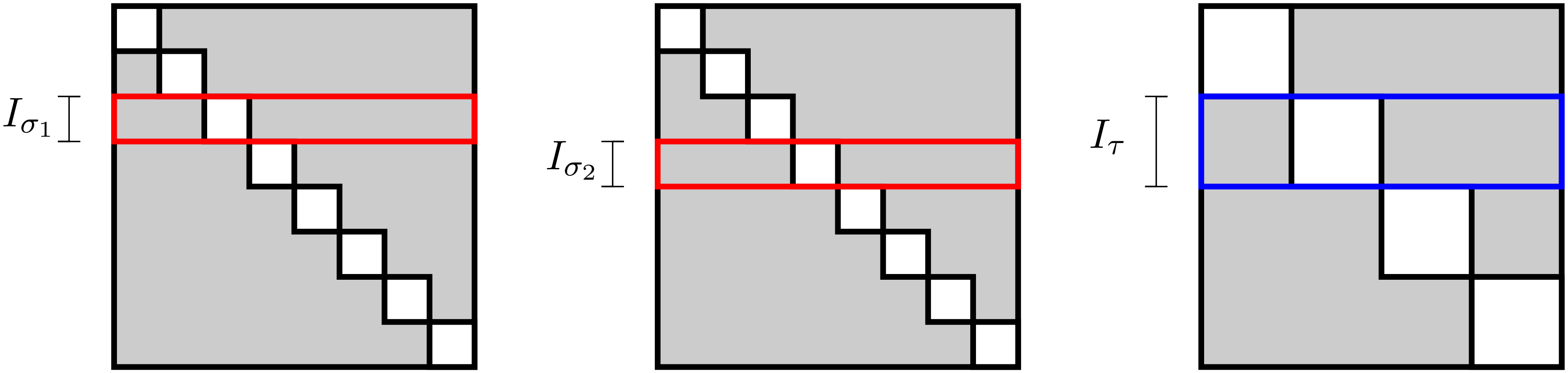
We consider the case of symmetric matrices only.


A
=
$H^{(p)}$
+
$D^{(p)}$



For a box τ at level p, corresponding to index set I_τ, set

$$R_\tau = H(I_\tau, :).$$



R_{σ_1} (in red) R_{σ_2} (in red) R_τ (in blue)

Factor R_{σ_1} and R_{σ_2}:

$$R_{\sigma_1} = U_{\sigma_1}\,\hat{R}_{\sigma_1}, \qquad \text{and} \qquad R_{\sigma_2} = U_{\sigma_2}\,\hat{R}_{\sigma_2}.$$

Then

$$R_\tau(:, I_\tau^{\mathrm{c}}) = \begin{bmatrix} R_{\sigma_1}(:, I_\tau^{\mathrm{c}}) \\ R_{\sigma_2}(:, I_\tau^{\mathrm{c}}) \end{bmatrix} = \begin{bmatrix} U_{\sigma_1} & 0 \\ 0 & U_{\sigma_2} \end{bmatrix} \begin{bmatrix} \hat{R}_{\sigma_1}(:, I_\tau^{\mathrm{c}}) \\ \hat{R}_{\sigma_2}(:, I_\tau^{\mathrm{c}}) \end{bmatrix} = \begin{bmatrix} U_{\sigma_1} & 0 \\ 0 & U_{\sigma_2} \end{bmatrix} \hat{U}_\tau\,\tilde{R}_\tau(:, I_\tau^{\mathrm{c}}).$$

The straight-forward implementation of this idea leads to an $O(k\,N^2)$ scheme.

How to reduce the cost to $O(k\,N)$?

Suppose that we have a fast matrix-vector multiplier.

(And that we can afford to evaluate a small number of actual entries of A.)

The randomized sampling machinery tells us that if Ω is an $N \times (k+10)$ matrix whose entries are i.i.d. Gaussian random variables, then we can construct U_τ from

$$\Psi_\tau = R_\tau\,\Omega.$$

All the matrices $\{\Psi_\tau\}_\tau$ can be generated from the global product $A\,\Omega$.

First compute

$$\Psi = A\,\Omega.$$

Then on the finest level, level P, compute

$$\Psi^{(P)} = H^{(P)}\,\Omega = (A - D^{(P)})\,\Omega = \Psi - D^{(P)}\,\Omega.$$

From $\Psi^{(P)}$ we directly extract for any box τ on the finest level

$$\Psi_\tau = \Psi^{(P)}(I_\tau, :).$$

Since $\Psi_\tau = R_\tau\,\Omega$, we can construct U_τ from Ψ_τ.

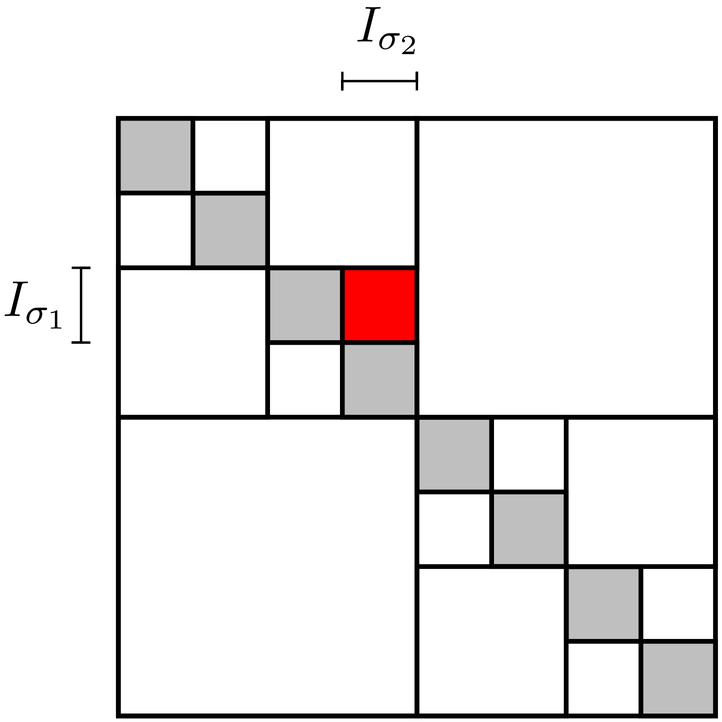
On the next coarser level, level $P-1$, we find that

$$\Psi^{(P-1)} = H^{(P-1)}\,\Omega = (A - D^{(P)})\,\Omega - (D^{(P-1)} - D^{(P)})\,\Omega = \Psi^{(P)} - (D^{(P-1)} - D^{(P)})\,\Omega.$$

From $\Psi^{(P-1)}$, we construct U_τ for any cell τ on level $P-1$.

Then proceed recursively,

$$\Psi^{(p-1)} = \Psi^{(p)} - (D^{(p-1)} - D^{(p)})\,\Omega.$$



The method as described so far only constructs bases for the off-diagonal blocks, not full factorizations.

Consider the block $B_{\sigma_1\sigma_2}$ marked in red in the figure above. We know that

$$B_{\sigma_1\sigma_2} = U_{\sigma_1}\,\hat{B}_{\sigma_1\sigma_2}\,U_{\sigma_2}^{\rm t},$$

for some as yet unknown matrix $\hat{B}_{\sigma_1\sigma_2}$.

Note: It is of course the case that $\hat{B}_{\sigma_1\sigma_2} = U_{\sigma_1}^{\rm t}\,B_{\sigma_1\sigma_2}\,U_{\sigma_2}$, but it's too expensive to actually compute these matrix-matrix products.

Let us extract the core question:

Suppose that we are given:

- An $N \times N$ matrix B of rank $k \ll N$.
- An $N \times k$ matrix U_1 such that $B = U_1\,U_1^{\rm t}\,B$.
- An $N \times k$ matrix U_2 such that $B = B\,U_2\,U_2^{\rm t}$.

We do **not** have a fast matrix-vector multiplier.
Can you still compute a factorization for B in less than $O(N^2)$ time?

Yes, by using interpolative decompositions! Pick k rows of U_1 that span its row-space, and k rows of U_2 that span its row-space,

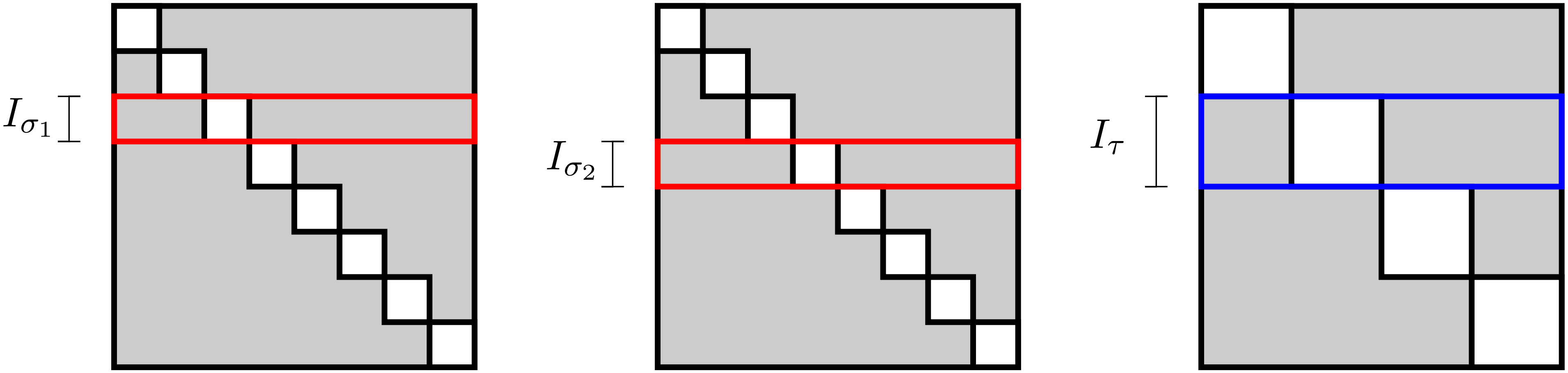
$$U_1 = X_1\,U_1(J_1,:), \qquad \text{and} \qquad U_2 = X_2\,U_2(J_2,:).$$

Then, since $B = U_1\;(U_1^{\rm t}\,B\,U_2)\;U_2^{\rm t}$,

$$B = X_1\,B(J_1,J_2)\,X_2^{\rm t}. \tag{1}$$

Post-processing easily converts (1) to an SVD using $O(N\,k^2)$ flops.

Returning to the context of computing bases for the “HSS-blocks” R_τ ...



R_{σ_1} (in red)

R_{σ_2} (in red)

R_τ (in blue)

Recall that σ_1 and σ_2 are the two children of box τ.

Pick k rows of R_{σ_1} that span the row space of R_{σ_1} and k rows of R_{σ_2} that span the row space of R_{σ_2}. Then pick k rows of R_τ out of the $2k$ rows that span R_{σ_1} and R_{σ_2} to span the row space of R_τ.

Added bonus: We actually only need to carry the rows of the $\Psi^{(p)}$’s that correspond to the spanning rows! This is how the algorithm described gets $O(N)$ rather than $O(N \log N)$ complexity.

Key points:

- Structured matrices are common in scientific computing.
- There exist fast algorithms for performing linear algebra operations on structured matrices once they have been compressed.
- Randomized sampling can in many environments be used to simultaneously compress all low-rank blocks in a structured matrix.

Open question:

Can this machinery be extended to Laplacians on “general” network matrices?