Supplementary notes for Lecture 3

Sparse matrices and graph models

Numerical Mathematics, 2026-27

1 A variational characterization of fill

For an elimination ordering π, let $\mathrm{fill}(G, \pi)$ be the number of fill edges. Let $\mathcal{C}(G)$ be the set of chordal supergraphs of G on the same vertex set.

Minimum fill as a chordal completion.

$$\min_{\pi} \mathrm{fill}(G, \pi) = \min_{H \in \mathcal{C}(G)} |E(H) \setminus E(G)|.$$

Proof. For the inequality $\leq$, take any ordering π and let $H = G^{F(\pi)}$ be its filled graph. The filled graph is chordal, contains G, and its edges outside G are exactly the fill edges. Hence

$$\min_{H \in \mathcal{C}(G)} |E(H) \setminus E(G)| \leq \mathrm{fill}(G, \pi).$$

Taking the minimum over π gives $\min_H \leq \min_\pi$.

For the reverse inequality, let H be a chordal supergraph of G and let π be a perfect elimination ordering of H. Eliminate G in the order π. At every step, the elimination graph of G is a subgraph of the elimination graph of H. Eliminating a vertex deletes it and makes its neighborhood a clique; since every neighborhood in G is contained in the corresponding neighborhood in H, the same operation preserves the subgraph relation. This is a statement about sparsity patterns; numerical cancellation is ignored. Since π is a PEO of H, eliminating H creates no fill; therefore every fill edge created while eliminating G is already an edge of H. Thus

$$\mathrm{fill}(G, \pi) \leq |E(H) \setminus E(G)|.$$

Taking the minimum over chordal H gives $\min_\pi \leq \min_H$.

The same argument gives the standard characterization of treewidth:

$$\mathrm{tw}(G) = \min_{H \in \mathcal{C}(G)} (\omega(H) - 1),$$

where $\omega(H)$ is the clique number. A PEO minimizes the largest later neighborhood, and the corresponding filled graph is a chordal completion. Minimum fill and treewidth are therefore two facets of the same chordal-completion problem; both are NP-hard to compute exactly.

Corollary: Treewidth as a chordal-completion parameter.

$$\mathrm{tw}(G) = \min_{H \in \mathcal{C}(G)} (\omega(H) - 1),$$

where $\omega(H)$ is the clique number of H.

Proof. If H is a chordal completion and π is a PEO of H, then eliminating G in the order π gives every vertex at most $\omega(H) - 1$ later neighbors: the later neighbors together with the vertex form a clique of H. Hence $\mathrm{tw}(G) \leq \omega(H) - 1$, and minimizing over H gives $\mathrm{tw}(G) \leq \min_H(\omega(H) - 1)$.

Conversely, take an ordering π that attains $\operatorname{tw}(G) = k$ and let $H = G^{F(\pi)}$. The filled graph is chordal, and every clique created during the elimination has at most $k+1$ vertices. Every clique of H is contained in one of these elimination cliques, so $\omega(H) \le k+1$. Thus $\min_H(\omega(H) - 1) \le k = \operatorname{tw}(G)$.

2 Bounded treewidth and fill

This section gives the full counting argument behind the bounded-treewidth estimate in the deck.

Treewidth. For a graph G,

$$\operatorname{tw}(G) = \min_{\text{orderings}} \max_i \bigl|N(v_i) \cap \{v_{i+1}, \dots, v_n\}\bigr|.$$

Bounded treewidth. Let $\operatorname{tw}(G) = k$. Then there is an elimination ordering with at most k later neighbors at every step. For that ordering, the Cholesky factor has $O(nk)$ nonzeros, the number of fill edges is $O(nk^2)$, and the factorization costs $O(nk^2)$ flops.

Proof. Choose an ordering $v_1, \dots, v_n$ that attains the minimum in the definition of treewidth, and put $N_i = N(v_i) \cap \{v_{i+1}, \dots, v_n\}$. Then $|N_i| \le k$ for every i.

In the elimination, the entry (v_i, v_i) is the diagonal entry of column i of L, and the off-diagonal entries of column i can occur only in rows indexed by the later neighbors N_i. The reason is that the only new pattern entries incident to v_i are created when v_i is eliminated, and those entries join v_i to a later neighbor. Hence column i has at most $k+1$ nonzeros, including the diagonal, and

$$\operatorname{nnz}(L) \le \sum_{i=1}^{n} (k+1) = n(k+1) = O(nk).$$

When v_i is eliminated, the elimination lemma makes N_i a clique. Since $|N_i| \le k$, at most

$$\binom{k}{2} = \frac{k(k-1)}{2}$$

new edges can be created at this step. These edges are charged once, when they first appear, so the total number of fill edges is at most

$$n \frac{k(k-1)}{2} = O(nk^2).$$

For the flop count, the Schur-complement update at step i changes only the trailing block indexed by N_i. That block has at most k rows and k columns, so the rank-one update costs $O(k^2)$ operations. Summing over the n steps gives $O(nk^2)$ flops. The same count bounds the work needed to form the factor, while the storage bound is $O(nk)$ from the column count above.

The estimate $O(nk^2)$ for fill edges is a safe upper bound. The important conclusion is that, for fixed treewidth, both storage and work are linear in n. This is the quantitative reason why bounded-treewidth graphs are the favourable case for sparse elimination.

3 Cuthill–McKee and reverse Cuthill–McKee

The bandwidth of a symmetric matrix is

$$b = \max\bigl\{|i - j| : a_{ij} \neq 0\bigr\},$$

and its profile is $\sum_i (i - \mathrm{env}(i))$, where $\mathrm{env}(i)$ is the first nonzero column of row i. Cuthill–McKee (CM) is a graph heuristic for reducing these quantities.

Breadth-first search (BFS) from a starting vertex partitions the vertices by graph distance: the BFS layer L_j is the set of vertices at distance j from the start. The CM ordering is built from these layers. Choose a starting vertex (a minimum-degree vertex is a common choice). Put it first, then list its neighbors in increasing order of degree, then the unvisited neighbors of those vertices in increasing degree, and continue until the graph is exhausted. Within each layer, the vertices are sorted by their degree in the original graph.

Cuthill–McKee algorithm

1 Choose a start vertex v_0 (a peripheral or minimum-degree vertex).
2 Set $L_0 = \{v_0\}$ and mark v_0 visited.
3 For $j = 0, 1, \ldots$, form L_{j+1} from the unvisited neighbors of L_j, ordered by increasing degree.
4 Append the layers in order; the resulting list is the CM permutation.

The reverse Cuthill–McKee (RCM) ordering is obtained by reversing the CM list. Reversal often reduces the profile and the bandwidth further, because it changes which side of the matrix the envelope is measured from. Both variants cost $O(n + m)$ with bucket sorting by degree, or $O((n + m) \log n)$ with a comparison sort.

> CM and RCM target bandwidth and profile, not fill directly. They are useful for banded or skyline solvers and as cheap global orderings, but they do not replace minimum-degree or nested-dissection orderings when fill is the primary cost. The notebook compares RCM with natural and minimum-degree-type orderings on a sparse matrix.

> **A path.** For a path $1 - 2 - \ldots - n$ started at an endpoint, the BFS layers are singletons and CM returns the natural order; RCM returns the reverse order. Both have bandwidth 1, so the example shows the mechanism but not the advantage. On a two-dimensional grid the layers are diagonals, and the reordering can reduce the bandwidth substantially.

4 A worked nested-dissection example

Consider the 5×5 five-point grid with vertices numbered row by row:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 & 25 \end{pmatrix}.$$

Each vertex is connected to its horizontal and vertical neighbors. The graph has 40 edges, and the lower triangle of A has $25 + 40 = 65$ nonzeros.

A nested-dissection ordering uses the cross

$$S = \{3, 8, 11, 12, 13, 14, 15, 18, 23\},$$

the middle row together with the middle column. Removing S leaves four 2×2 blocks:

$$Q_1 = \{1, 2, 6, 7\}, \quad Q_2 = \{4, 5, 9, 10\}, \quad Q_3 = \{16, 17, 21, 22\}, \quad Q_4 = \{19, 20, 24, 25\}.$$

The ordering is Q_1, Q_2, Q_3, Q_4, S. The four blocks are independent, so they can be eliminated first; the separator is eliminated last.

Fill counts. For the natural ordering $1, 2, \ldots, 25$, the elimination creates 64 fill edges. The lower-triangular factor therefore has $65 + 64 = 129$ stored entries. For the nested-dissection ordering above, the number of fill edges is 51, so the factor has $65 + 51 = 116$ stored entries. On this tiny grid the gain is modest; for an $m \times m$ grid the asymptotic counts are much more favorable.

The natural ordering has bandwidth m and fill $O(m^3) = O\big(n^{3/2}\big)$ for an $m \times m$ grid. Nested dissection has fill $O(n \log n)$ and factorization cost $O\big(n^{3/2}\big)$. The separator recurrence is

$$F(n) \le 4F(n/4) + O(n), \quad T(n) \le 4T(n/4) + O\big(n^{3/2}\big),$$

which gives $F(n) = O(n \log n)$ and $T(n) = O\big(n^{3/2}\big)$.

For comparison, the following table gives the fill counts for $m \times m$ grids with $m = 5, 7, 9$, computed by running the same symbolic elimination for the natural ordering and for the recursive cross ordering. The nested-dissection counts grow more slowly, although the asymptotic advantage is not dramatic on these small instances.

m	$n = m^2$	Natural fill	Nested fill
5	25	64	51
7	49	216	155
9	81	512	383

The cross is a separator of size $O(m) = O(\sqrt{n})$, not the optimal $O(\sqrt{n})$ separator of the planar separator theorem. For a grid it already gives the right exponent. The planar separator theorem of Lipton and Tarjan guarantees a separator of size $O(\sqrt{n})$ for every planar graph, which yields the same asymptotic bounds.

4.1 Symbolic factorization on a 3×3 grid

Consider the 3×3 grid

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix},$$

with the nested-dissection-style ordering

$$1, 3, 7, 9, 2, 4, 6, 8, 5 :$$

the four corners, then the four edge midpoints, then the center. The following table is the output of symbolic factorization: for each column j it lists the set $\mathcal{N}_j$ of rows $i > j$ in the lower-triangular factor L (the diagonal is implicit), and the parent in the elimination tree.

Column	Later neighbors	Parent
1	2, 4	2
3	2, 6	2
7	4, 8	4
9	6, 8	6
2	4, 6, 5	4
4	6, 8, 5	6
6	8, 5	8
8	5	5
5	—	—

The elimination tree is therefore

$$5 \leftarrow 8 \leftarrow 6 \leftarrow \{4, 9\}, \quad 4 \leftarrow \{2, 7\}, \quad 2 \leftarrow \{1, 3\}.$$

Equivalently, the parent map is

$$1 \to 2, \quad 3 \to 2, \quad 7 \to 4, \quad 9 \to 6, \quad 2 \to 4, \quad 4 \to 6, \quad 6 \to 8, \quad 8 \to 5.$$

The factor has

$$\operatorname{nnz}(L) = 9 + \sum_j |\mathcal{N}_j| = 9 + (2 + 2 + 2 + 2 + 3 + 3 + 2 + 1 + 0) = 26$$

entries. The original lower triangle has $9 + 12 = 21$ entries, so the number of fill entries is 5.

> Symbolic factorization computes this pattern from the graph alone, before any floating-point operation. The elimination tree records the dependency between columns: column j must be processed before $\operatorname{parent}(j)$. The tree in this example has two branches at the center, which is typical for a nested-dissection ordering.

5 Nested dissection: the planar recurrence

The deck states the planar separator theorem and gives the grid recurrence as a sketch. This section supplies the full nested-dissection accounting. The only external ingredient is the Lipton–Tarjan separator theorem itself; its proof is a separate, longer argument.

> **Planar separator, Lipton–Tarjan.** Every planar graph with n vertices has a vertex separator S of size at most $C\sqrt{n}$ whose removal leaves components of size at most $2n/3$.

Let $F(n)$ be the number of fill edges produced by nested dissection on a planar graph with n vertices, and let $T(n)$ be the factorization cost.

> **Fill recurrence.**
>
> $$F(n) = O(n \log n).$$

> ***Proof.*** Consider one recursive node whose subgraph has n vertices. Its separator has size $s \le C\sqrt{n}$. After the components are eliminated, the separator vertices are eliminated last; by the elimination lemma they become a clique, so this step creates at most
>
> $$\binom{s}{2} \le C^2 n/2 = O(n)$$
>
> fill edges. No other fill is created at this node: all remaining work belongs to the components.
>
> If the components have sizes $n_1, \dots, n_r$, then $n_j \le 2\frac{n}{3}$ and $\sum_j n_j \le n$. Therefore
>
> $$F(n) \le \sum_{j=1}^{r} F\big(n_j\big) + cn.$$
>
> At depth ℓ of the recursion, the subgraphs are disjoint and have total size at most n. The separator-fill contribution of their parent nodes is at most c times that total, hence at most cn at every level. Every subgraph at depth ℓ has size at most $(2/3)^\ell n$, so the recursion stops after $O(\log n)$ levels. Consequently
>
> $$F(n) \le cn\Big(1 + \log_{3/2} n\Big) = O(n \log n).$$

Factorization recurrence.

$$T(n) = O\big(n^{3/2}\big).$$

Proof. Factor the components recursively. After they are finished, the only remaining block at this node is the separator Schur complement, which has size $s \times s$ and is dense; factoring it costs

$$O(s^3) = O\big(n^{3/2}\big).$$

Thus, for some constant c,

$$T(n) \le \sum_{j=1}^{r} T\big(n_j\big) + cn^{3/2}, \quad n_j \le 2\frac{n}{3}, \quad \sum_j n_j \le n.$$

To bound the sum at one level, note that the function $x^{3/2}$ is convex. With $n_j \le 2\frac{n}{3}$ and $\sum_j n_j \le n$, the sum $\sum_j n_j^{3/2}$ is largest when one component has size $2\frac{n}{3}$ and the remaining mass $\frac{n}{3}$ is itself one component. Its factor is

$$\rho = (2/3)^{3/2} + (1/3)^{3/2} = \frac{2\sqrt{2}+1}{3\sqrt{3}} < 1.$$

Hence the total cost at depth ℓ is at most $c\rho^\ell n^{3/2}$, again by induction over the levels. The geometric series gives

$$T(n) \le cn^{3/2} \sum_{\ell \ge 0} \rho^\ell = \frac{c}{1-\rho} n^{3/2} = O\big(n^{3/2}\big).$$

Nested dissection on planar graphs. Nested dissection gives

$$\text{fill} = O(n \log n), \quad \text{factorization} = O\big(n^{3/2}\big), \quad \text{solve} = O(n \log n).$$

Proof. The fill and factorization bounds are the two lemmas. A planar graph has $O(n)$ edges. The factor has

$$\operatorname{nnz}(L) = \operatorname{nnz}(A) + \text{fill} = O(n) + O(n \log n) = O(n \log n)$$

nonzeros, and a triangular solve touches each nonzero once. Hence one solve costs $O(n \log n)$.

5.1 The grid case in full

For an $m \times m$ grid with $n = m^2$, take the middle row and middle column as the separator S. Then $|S| = 2m - 1 = O(\sqrt{n})$, and removing S leaves four square subgrids of size at most $n/4$ (up to boundary effects). The recurrences are

$$F(n) \le 4F(n/4) + O(n), \quad T(n) \le 4T(n/4) + O\big(n^{3/2}\big).$$

For the fill recurrence, at depth ℓ there are 4^ℓ subgrids of size about $n/4^\ell$, so the total separator fill at that level is

$$4^\ell O\big(n/4^\ell\big) = O(n).$$

The depth is $\log_4 n$, hence

$$F(n) = O(n \log_4 n) = O(n \log n).$$

For the factorization recurrence, a level contributes

$$4^{\ell} O\Big(\big(n/4^{\ell}\big)^{3/2}\Big) = O\big(n^{3/2}/2^{\ell}\big).$$

This is a geometric series:

$$T(n) = O\big(n^{3/2}\big) \sum_{\ell \geq 0} 2^{-\ell} = O\big(n^{3/2}\big).$$

The factor has $O(n \log n)$ nonzeros by the fill bound, so the solve bound is again $O(n \log n)$.

> The grid calculation also shows why the exponents are what they are: the separator has size $\Theta(\sqrt{n})$, so the dense separator block costs $\Theta\big(n^{3/2}\big)$ at the root. The four-way split makes the level costs decrease geometrically.

5.2 Constructing separators in practice

The Lipton–Tarjan proof is constructive: BFS layers of a spanning tree yield a separator of size $O(\sqrt{n})$ in polynomial time. The theorem is nevertheless an existence result with unspecified constants; exact separator optimization is not the practical route. Production nested dissection therefore uses different separator generators in practice:

- geometric separators for meshes with coordinates, such as recursive bisection by a median coordinate;
- multilevel graph partitioners (METIS, SCOTCH) for general graphs;
- minimum-degree or approximate-minimum-degree orderings for local fill, combined with a global nested-dissection partition.

The planar separator theorem supplies the asymptotic guarantee behind nested dissection; the practical separator is a heuristic chosen for good constants and speed.

6 A complete proof of the Cauchy determinant formula

Cauchy determinant. For distinct $x_1, \dots, x_n$ and $y_1, \dots, y_n$ with $x_i \neq y_j$,

$$\det\left(\frac{1}{x_i - y_j}\right) = (-1)^{n(n-1)/2} \frac{\prod_{i<j}(x_j - x_i)(y_j - y_i)}{\prod_{i,j}(x_i - y_j)}.$$

Proof. Let D_n be the determinant. For $n = 2$,

$$\det\left(\begin{pmatrix} \frac{1}{x_1 - y_1} & \frac{1}{x_1 - y_2} \\ \frac{1}{x_2 - y_1} & \frac{1}{x_2 - y_2} \end{pmatrix}\right) = -\frac{(x_2 - x_1)(y_2 - y_1)}{(x_1 - y_1)(x_1 - y_2)(x_2 - y_1)(x_2 - y_2)},$$

which is the formula with sign $(-1)^{2 \cdot 1/2} = -1$.

For the induction step, subtract the last row from every other row. The (i, j) entry for $i < n$ becomes

$$\frac{1}{x_i - y_j} - \frac{1}{x_n - y_j} = \frac{x_n - x_i}{(x_i - y_j)(x_n - y_j)}.$$

Factor $x_n - x_i$ from row i and $\frac{1}{x_n - y_j}$ from column j. The resulting matrix has the first $n - 1$ rows $\frac{1}{x_i - y_j}$ and last row all ones. Subtract the last column from columns $j < n$:

$$\frac{1}{x_i - y_j} - \frac{1}{x_i - y_n} = \frac{y_j - y_n}{(x_i - y_j)(x_i - y_n)}.$$

Factor $y_j - y_n$ from column $j < n$, then $\frac{1}{x_i - y_n}$ from rows $i < n$. Expanding along the last row leaves exactly the Cauchy determinant D_{n-1} in the first $n-1$ variables. The new factors add $\prod_{j<n}(y_j - y_n)$, which contributes the remaining sign $(-1)^{n-1}$, so the induction gives the stated sign.

The same induction is the standard proof of Cauchy's double alternant in Krattenthaler, **Advanced Determinant Calculus**, Section 2.1, equation (2.7). The formula is also the starting point for Cauchy matrices, displacement rank, and the fast Cauchy solver used in structured linear algebra.

7 Fast algebra from displacement structure

The displacement equation

$$D_x C - C D_y = 11^T, \quad D_x = \operatorname{diag}(x_1, \dots, x_n), \quad D_y = \operatorname{diag}(y_1, \dots, y_n),$$

is a compact description of the Cauchy matrix $C_{ij} = \frac{1}{x_i - y_j}$. The right-hand side has rank one, so the equation is a certificate that the n^2 entries of C are generated by the two short vectors x, y and by the rank-one term 11^T. Structured algorithms store this **generator**, not all entries.

7.1 Fast matrix–vector multiplication

For a vector b, define

$$q(z) = \prod_{k=1}^{n} (z - y_k), \quad p(z) = q(z) \sum_{j=1}^{n} \frac{b_j}{z - y_j} = \sum_{j=1}^{n} b_j \prod_{k \neq j} (z - y_k).$$

The numerator is a polynomial of degree at most $n-1$, and

$$\frac{p(z)}{q(z)} = \sum_{j=1}^{n} \frac{b_j}{z - y_j},$$

so

$$(Cb)_i = \frac{p(x_i)}{q(x_i)}, \quad i = 1, \dots, n.$$

This identity is the basis of the fast multiplication. The polynomial q is built by a **product tree**: pair the linear factors $z - y_j$, multiply the pairs, then multiply the resulting quadratics, and continue. With FFT-based polynomial multiplication, the total cost of building the tree and q is $O(M(n) \log n)$, where $M(n) = O(n \log n)$ is the cost of multiplying two degree-n polynomials. The values $p(y_j) = b_j q'(y_j)$ determine p; fast interpolation from these values uses the same product tree. Finally, p is evaluated at all x_i by **fast multipoint evaluation**: recursively divide p by the node polynomials of the tree and keep the remainders. This costs another $O(M(n) \log n)$. Combining the steps gives

$$O(M(n) \log n) = O(n \log^2 n)$$

operations.

The displacement equation is used at the structural level: it guarantees that the Cauchy matrix is represented by the short generator $(x, y, 1, 1)$. The actual arithmetic is polynomial manipulation. Without this structure, the entries are unrelated and the product Cb costs $O(n^2)$.

7.2 Fast inversion

The inverse is again displacement-structured. Multiplying the displacement equation on the left and right by C^{-1} gives

$$C^{-1}D_x - D_yC^{-1} = C^{-1}(11^T)C^{-1} = (C^{-1}1)(1^TC^{-1}).$$

The right-hand side is the outer product of two vectors, so C^{-1} has displacement rank one with the same diagonal matrices (the sign and order of the displacement operator are reversed).

A matrix with a displacement generator of rank r can be inverted by a **generalized Schur algorithm**. At each elimination step, the algorithm updates the generator using the displacement equation and forms the next structured Schur complement; the Schur complements remain displacement rank r. Each generator update costs $O(rn)$, and there are n steps, so the inverse costs $O(rn^2)$. For a Cauchy matrix $r = 1$, giving the $O(n^2)$ inversion stated in the deck. The inverse is obtained as a structured factorization; one does not run the $O(n^2)$ matvec algorithm n times.

> The inverse displacement equation and the closure of displacement rank under Schur complementation are the two facts behind the structured solver. The same mechanism applies to Toeplitz, Vandermonde, and Cauchy-like matrices; only the displacement operator and the size of the generator change.

8 Hierarchical matrices

8.1 What an H-matrix is

An H-matrix is a data-sparse representation of a dense matrix by recursive block partitioning. Blocks that are far apart in an appropriate sense are stored as low-rank products.

> **Cluster tree.** Let $I = \{1, \ldots, m\}$. A **cluster tree** T_I is a rooted tree whose nodes are subsets of I. The root is I, and the children of an internal node t are disjoint sets whose union is t. The nodes are called **clusters**; the leaves are the leaf clusters.

> **Block cluster tree.** Let $I = \{1, \ldots, m\}$ and $J = \{1, \ldots, n\}$, with cluster trees T_I and T_J. A **block cluster tree** $T_{I\times J}$ is a rooted tree whose nodes are pairs (t, s) with $t \subseteq I$ and $s \subseteq J$. The root is (I, J). An internal node is refined by replacing t by its children, s by its children, or both, so that the children partition the Cartesian product $t \times s$. The recursion stops at a block that is admissible or cannot be split further; its leaves are the **blocks** of the partition.

> For a given matrix $A \in \mathbb{R}^{m\times n}$, the block cluster tree is a partition of the index rectangle $I \times J$ built from the row and column index sets (or from their geometry); it is not uniquely determined by the numerical entries. One chooses it so that admissible blocks have low numerical rank.

> **Cluster tree for four indices.** Let $I = J = \{1, 2, 3, 4\}$, split each root into $I_1 = \{1, 2\}$, $I_2 = \{3, 4\}$ and the corresponding column clusters, and split those into singletons. If the block cluster tree splits both sides at the root, its four children are $I_1 \times J_1$, $I_1 \times J_2$, $I_2 \times J_1$, and $I_2 \times J_2$: they are the four 2×2 submatrices of the partition. Each child is refined until its blocks are admissible or reach singleton clusters.

> **H-matrix.** Fix an **admissibility predicate** $\mathrm{adm}(t, s)$ on pairs of row and column clusters and a rank bound r. Let $A_{t,s}$ denote the submatrix of A with rows in t and columns in s. An **H-matrix** $A \in \mathbb{R}^{m\times n}$ is represented by leaf blocks stored as follows:
> - if $\mathrm{adm}(t, s)$ holds, $A_{t,s}$ is approximated by UV^* with $U \in \mathbb{R}^{|t|\times r}$, $V \in \mathbb{R}^{|s|\times r}$;
> - otherwise, $A_{t,s}$ is stored as a dense $|t| \times |s|$ block.

The number r is the **block rank** (it may be allowed to vary from block to block). The definition allows $m \neq n$; the square case is $I = J$.

A common geometric choice is the following. If the row indices are points x_i and the column indices are points y_j, put

$$\operatorname{diam}(t) = \max_{i,j \in t} \|x_i - x_j\|, \quad \operatorname{dist}(t,s) = \min_{i \in t, j \in s} \|x_i - y_j\|,$$

and call t, s admissible when $\max(\operatorname{diam}(t), \operatorname{diam}(s)) \leq \eta \operatorname{dist}(t,s)$. This is only one possible admissibility rule; the H-matrix definition itself is independent of geometry.

A two-block H-matrix. Partition an 8×8 matrix into four 4×4 blocks:

$$A = \begin{pmatrix} A_{11} & U_1 V_1^* \\ U_2 V_2^* & A_{22} \end{pmatrix},$$

where A_{11}, A_{22} are dense and the off-diagonal blocks have rank one, with $U_i, V_i \in \mathbb{R}^{4 \times 1}$. Storage is $16 + 16 + 8 + 8 = 48$ numbers instead of 64. The matrix–vector product is

$$y_1 = A_{11} x_1 + U_1 (V_1^* x_2), \quad y_2 = A_{22} x_2 + U_2 (V_2^* x_1),$$

which uses two dense 4×4 products and four rank-one products.

In a typical H-matrix the partition is recursive: diagonal blocks are subdivided further, while sufficiently separated off-diagonal blocks are approximated at a coarser level. For a one-dimensional integral operator, the storage is $O(n \log n)$; for a two-dimensional problem with a suitable geometric admissibility condition, it is also $O(n \log n)$ up to logarithmic factors, depending on the block rank.

H-matrices support approximate arithmetic on the block structure. Addition requires a common refinement of the two block partitions; low-rank blocks are added and the ranks can double, so the result is truncated back to the prescribed rank. Matrix–vector multiplication costs $O(n \log n)$ for a fixed block rank and a balanced partition. Matrix–matrix multiplication and approximate factorization use the same block recursion and truncate ranks after each combination; their cost is near-linear up to logarithmic factors, depending on the dimension and admissibility. Linear systems are solved by H-LU factorization or by iterative methods preconditioned with an H-matrix.

H-matrices are useful whenever a dense operator has numerically low-rank blocks away from the diagonal. Typical applications include boundary integral equations and boundary-element methods, elliptic PDEs, Gaussian-process covariance matrices and kernel methods, Schur complements in sparse direct solvers, optimal control, and model reduction. The common feature is not a particular application but a hierarchical partition with compressible far-field blocks.

8.2 Fast multipole method

The fast multipole method (FMM) evaluates sums

$$u_i = \sum_{j=1}^{n} K(x_i, x_j) q_j, \quad i = 1, \ldots, n,$$

for a smooth kernel K in $O(n)$ or $O(n \log n)$ operations. The index set is organized in a tree. For two well-separated clusters, the kernel is expanded in a multipole expansion around the source

cluster and a local expansion around the target cluster; translations move the expansion up and down the tree. The far-field interaction is thus computed once per cluster pair instead of once per pair of points.

One-dimensional multipole expansion. Consider the kernel $K(x,y) = \frac{1}{x-y}$ and two clusters: sources $y_j \in [a,b]$ and targets x_i with $|x_i| > \max_j |y_j|$. Then

$$\frac{1}{x_i - y_j} = \frac{1}{x_i} \sum_{k \geq 0} \left(y_j / x_i\right)^k.$$

Truncating after p terms gives

$$\frac{1}{x_i - y_j} \approx \sum_{k=0}^{p-1} x_i^{-k-1} y_j^k.$$

The sums $\sum_j q_j y_j^k$ are the multipole moments of the source cluster; evaluating the truncated expansion at all targets costs $O(p)$ per target instead of $O(n)$. The truncation error is the geometric-series remainder

$$\left| \frac{1}{x_i - y_j} - \sum_{k=0}^{p-1} x_i^{-k-1} y_j^k \right| = \left(|y_j / x_i|\right)^p / |x_i - y_j|,$$

so it decays exponentially in p whenever the clusters are separated by a fixed relative gap. This is the one-dimensional prototype of the FMM.

The low-rank blocks of an H-matrix are the algebraic counterpart of the FMM expansions. An H-matrix stores the block factors explicitly; the FMM applies the expansions on the fly and avoids storing the factors. Both rely on the same analytic fact: for a smooth kernel and well-separated clusters, the interaction matrix has rapidly decaying singular values.

Standard references are W. Hackbusch, **Hierarchical Matrices: Algorithms and Analysis** (Springer, 2015), and S. Börm, L. Grasedyck, and W. Hackbusch, "Introduction to hierarchical matrices with applications," **Eng. Anal. Bound. Elem.** 27 (2003), 405–422. For the FMM, see L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," **J. Comput. Phys.** 73 (1987), 325–348.