Chapter 6

Solving Large Systems

6.1 Elimination with Reordering

Finite elements and finite differences produce large linear systems $KU = F$. *The matrix K is extremely sparse.* It has only a small number of nonzero entries in a typical row. In "physical space" those nonzeros are tightly clustered—they come from neighboring nodes and meshpoints. But we cannot number N^2 nodes in a plane in any way that keeps all neighbors close together! So in 2-dimensional problems, and even more in 3-dimensional problems, we meet three questions right away:

1. How best to number the nodes

2. How to use the sparseness of K (when nonzeros might be widely separated)

3. Whether to choose **direct elimination** or an **iterative method**.

That last point separates this section on elimination (where **node order** is important) from later sections on iterative methods (where **preconditioning** is crucial).

To fix ideas, we will create the n equations $KU = F$ from Laplace's difference equation in an interval, a square, and a cube. With N unknowns in each direction, K has order $n = N$ or N^2 or N^3. There are 3 or 5 or 7 nonzeros in a typical row of the matrix. Second differences in 1D, 2D, and 3D are shown in Figure 6.1.

Along an inside row of the matrix, the entries add to zero. In two dimensions this is $4 - 1 - 1 - 1 - 1 = 0$. This "zero sum" remains true for finite elements (the element shapes decide the exact numbers). It reflects the fact that $u = 1$ solves Laplace's equation and $U = \texttt{ones}(n, 1)$ has differences equal to zero. The constant vector solves $KU = 0$ *except near the boundaries.* When a neighbor is a boundary point, its known value moves onto the right side of $KU = F$. Then that row of K is *not zero sum.* Otherwise K would be singular, if $K * \texttt{ones}(n, 1) = \texttt{zeros}(n, 1)$.

Using block matrix notation, we can create the 2D matrix $\boldsymbol{K2D}$ from the familiar N by N second difference matrix K. We number the nodes of the square a row at

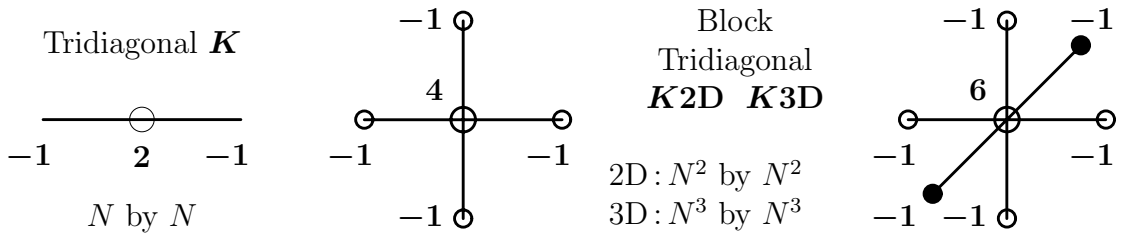


Figure 6.1: $3, 5, 7$-point difference molecules for $-u_{xx}, -u_{xx} - u_{yy}, -u_{xx} - u_{yy} - u_{zz}$.

a time (this "natural numbering" is not necessarily best). Then the -1's for the neighbors above and below are N positions away from the main diagonal of $\boldsymbol{K2D}$.

The 2D matrix in Figure 6.2a is *block tridiagonal with tridiagonal blocks*:

$$K = \begin{bmatrix} 2 & -1 & & \\ -1 & 2 & -1 & \\ & \cdot & \cdot & \cdot \\ & & -1 & 2 \end{bmatrix} \qquad \boldsymbol{K2D} = \begin{bmatrix} K+2I & -I & & \\ -I & K+2I & -I & \\ & \cdot & \cdot & \cdot \\ & & -I & K+2I \end{bmatrix} \tag{1}$$

Size N **Elimination in this order: Size $n = N^2$**

Time N **Bandwidth $w = N$, Space $nw = N^3$, Time $nw^2 = N^4$**

The matrix $K2D$ has 4's down the main diagonal. Its bandwidth $w = N$ is the distance from the diagonal to the nonzeros in $-I$. Many of the spaces in between are filled during elimination! Then the storage space required for the factors in $K2D = LU$ is of order $nw = N^3$. The time is proportional to $nw^2 = N^4$, when n rows each contain w nonzeros, and w nonzeros below the pivot require elimination.

Again, the operation count grows as nw^2. Each elimination step uses a row of length w. There can be nw nonzeros to eliminate below the diagonal. If some entries stay zero inside the band, elimination could be faster than nw^2—and this is our goal.

Those counts are not impossibly large in many practical 2D problems (and we show how they can be reduced). The horrifying N^7 count will come for elimination on $K3D$. Suppose the 3D cubic grid is numbered a plane at a time. In each plane we see a 2D square, and $K3D$ has blocks of order N^2 from those squares. With each square numbered as above, the blocks come from $\boldsymbol{K2D}$ and $\boldsymbol{I} = \boldsymbol{I2D}$:

$$\boldsymbol{K3D} = \begin{bmatrix} \boldsymbol{K2D}+\boldsymbol{2I} & -\boldsymbol{I} & & \\ -\boldsymbol{I} & \boldsymbol{K2D}+\boldsymbol{2I} & -\boldsymbol{I} & \\ & \cdot & \cdot & \cdot \\ & & -\boldsymbol{I} & \boldsymbol{K2D}+\boldsymbol{2I} \end{bmatrix}$$

3D Size $n = N^3$
Bandwidth $w = N^2$
Elimination space N^5
Elimination time N^7

The main diagonal of $K3D$ contains 6's, and "inside rows" have six -1's. Next to a face or edge or corner of the cube, we lose one or two or three of those -1's.

From any node to the node above it, we count N^2 nodes. The $-I$ blocks are far from the main diagonal and the bandwidth is $w = N^2$. Then $nw^2 = N^7$.

Fill-in Let me focus attention immediately on the key problem, when we apply elimination to sparse matrices. *The zeros "inside the band" may fill in with nonzeros.* Those nonzeros enter the triangular factors of $K = LL^{\mathrm{T}}$ (Figure 6.2a). When multiples of one pivot row are subtracted from other rows, a nonzero in that pivot row will infect all the others.

Sometimes a matrix has a full row, from an equation like $\sum U_j = 1$ (see Figure 6.3). That full row better come last!

A good way to visualize the nonzero structure (sparsity structure) of K is by a graph. The rows of K are nodes in the graphs. A nonzero entry K_{ij} produces an edge between nodes i and j. *The graph of $K2D$ is exactly the mesh of unknowns*! Filling in a nonzero adds a new edge (a diagonal in our square graph). Watch how fill-in happens in elimination:

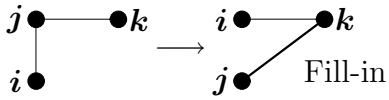
FILL-IN ***New nonzero in the matrix*** / ***New edge in the graph***

Suppose a_{ij} is eliminated. A multiple of row j later is subtracted from the row i. If a_{jk} is nonzero in row j, then a_{ik}^{new} becomes filled in row i:

Matrix nonzeros

$$\begin{matrix} a_{jj} & a_{jk} \\ a_{ij} & 0 \end{matrix} \longrightarrow \begin{matrix} a_{jj} & a_{jk} \\ 0 & a_{ik}^{\text{new}} \end{matrix}$$

Graph edges



Kronecker product One good way to create $K2D$ from K and I (N by N) is the $\texttt{kron}(A, B)$ command. This replaces each number a_{ij} by the block $a_{ij}B$. To take second differences in all columns at the same time, and all rows, use `kron` twice:

$$\boldsymbol{K2D} = \texttt{kron}(K, I) + \texttt{kron}(I, K) = \begin{bmatrix} 2I & -I & \cdot \\ -I & 2I & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} + \begin{bmatrix} K & & \\ & K & \\ & & \cdot \end{bmatrix} \tag{2}$$

This sum agrees with (1). Then a 3D box needs $K2D$ and $I2D = \texttt{kron}(I, I)$ in each plane. This easily adjusts to allow rectangles, with I's of different sizes. For a cube, take second differences inside all planes with $K2D$, plus differences in the z-direction:

$$\boldsymbol{K3D} = \texttt{kron}(K2D, I) + \texttt{kron}(I2D, K) \quad \text{has size} \quad (N^2)(N) = N^3\,. \tag{3}$$

Here K and $K2D$ and $K3D$ of size N and N^2 and N^3 are serving as *models of the type of matrices that we meet*. But we have to say that there are special ways to work with these particular matrices. The x, y, z directions are separable. We can use FFT-type methods in each direction. See Section 6.____ on *Fast Poisson Solvers*.

MATLAB should know that the matrices are sparse! If we create $I =$ `speye`(N) and K comes from `spdiags` as in Section 1.1, the `kron` command will preserve the sparse treatment. Before we describe reordering by minimum degree, allow me to display `spy`$(\boldsymbol{K2D})$ plus fill, and the reordered `spy`$(P\boldsymbol{K2D}P^{\mathrm{T}})$. *The latter will have much less fill-in.* Your eye may not see this so clearly, but the count is convincing!

Figure 6.2: (a) $K2D$ is filled in by elimination. (b) A good reordering of $K2D$.

Minimum Degree Algorithm

We now describe a useful way to reorder meshpoints and equations in $\boldsymbol{K2D}\,U = F$. The ordering achieves *approximate minimum degree* at each step—**the number of nonzeros below the pivot is almost minimized**. This is essentially the algorithm used in MATLAB's command $U = K\backslash F$, when K has been defined as a sparse matrix. We list some of the functions from the `sparfun` directory:

`speye`	(sparse identity I)	`nnz`	(number of nonzero entries)
`find`	(find indices of nonzeros)	`spy`	(visualize sparsity pattern)
`colamd` and `symamd`	(approximate minimum degree permutation of K)		

You can test and use the minimum degree algorithms without a careful analysis. The approximations are faster than the exact minimum degree permutations `colmmd` and `symmmd`. The speed (in two dimensions) and the roundoff errors are quite acceptable.

In the Laplace examples, the minimum degree ordering of nodes is irregular compared to "a row at a time." The final bandwidth is probably not decreased. But many nonzero entries are postponed as long as possible! That is the key.

The difference is shown in the **arrow matrix** of Figure 6.3. Its minimum degree ordering (one nonzero below each pivot) produces large bandwidth, but ***no fill-in***. Elimination will only change the last row and column. The triangular factors L and U have all the same zeros as A, keeping the arrow. The space for storage stays at $3n$, and elimination needs only n divisions and multiplications and subtractions. When a sparse matrix has a full row, *put it last.*

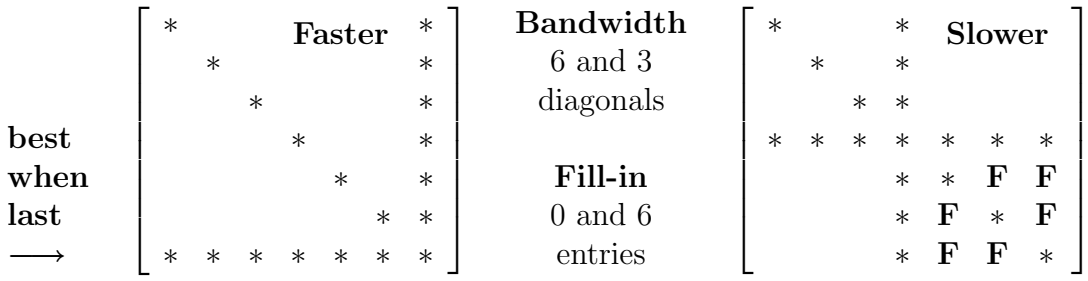


Figure 6.3: Minimum degree (arrow matrix) defeats minimum bandwidth.

The second ordering reduces the bandwidth from 6 to 3. But when row 4 is used as the pivot row, the entries indicated by **F** are *filled in*. That lower quarter of A becomes full, with $n^2/8$ nonzeros in each factor L and U. You see that the whole nonzero "profile" of the matrix decides the fill-in, not just the bandwidth.

Here is another example, from the **red-black ordering** of a square grid. Color the gridpoints like a checkerboard. Then all four neighbors or a red point are black, and vice versa. If we number the red points before the black points, the *permuted* $K2D$ has diagonal blocks of $4I$:

Red-black permuation

$$\boldsymbol{P}\,(\boldsymbol{K2D})\,\boldsymbol{P^{\mathrm{T}}} = \begin{bmatrix} 4\,I_{\mathsf{red}} & -1\text{'s} \\ -1\text{'s} & 4\,I_{\mathsf{black}} \end{bmatrix}. \tag{4}$$

This pushes the -1's and the fill-in into the last $N^2/2$ rows. Notice how P permutes the rows (the equations) and P^{T} permutes the columns (the unknowns). Together they keep 4's on the main diagonal.

Now the real thing. **Minimum degree algorithms** choose the $(k+1)$st pivot column, after the first k columns have been eliminated as usual below the diagonal. The algorithms look at the nonzeros in the lower right matrix of size $n-k$.

Symmetric case:	Choose the remaining meshpoint with the **fewest neighbors**.
Unsymmetric case:	Choose the remaining column with the **fewest nonzeros**.

The component of U corresponding to that column is renumbered $k+1$. So is the meshpoint in the finite difference grid. Of course elimination in that column will normally produce new nonzeros in the remaining columns. Some fill-in is unavoidable. The algorithm keeps track of the new positions of nonzeros, and also the actual entries. It is the ***positions*** that decide the ***ordering*** of unknowns (the permutation of columns). Then the *entries* in K decide the *numbers* in L and U.

The **degree of a node** is the number of connections to other nodes. This is the number of off-diagonal nonzeros in that column of K. In Figure 6.4 the corner nodes $1, 3, 4, 6$ all begin with degree 2. The side nodes 2 and 5 have degree 3. *The degrees change as elimination proceeds*! **Nodes connected to the pivot become connected to each other—and that entry of the matrix fills in.**

You will see how a renumbering of meshpoints preserves the symmetry of K. The rows and columns are reordered in the same way. Then $P\,K_{\mathsf{old}}P^{\mathrm{T}} = K_{\mathsf{new}} = K_{\mathsf{new}}^{\mathrm{T}}$.

Example 1 Figure 6.4 shows a small example of the minimal degree ordering, for Laplace's 5-point scheme. The problem starts with six meshpoints (they would be connected to boundary points where U is known) and a 6 by 6 matrix. K has two 3 by 3 tridiagonal blocks from horizontal links, and two 3 by 3 blocks with $-I$ from vertical links.

The first elimination step chooses row 1 as pivot row, because node 1 has minimum degree 2. (Any degree 2 node could come first, leading to different elimination orders.) The pivot is **P**, the other nonzeros in that row are boxed. When row 1 operates on rows 2

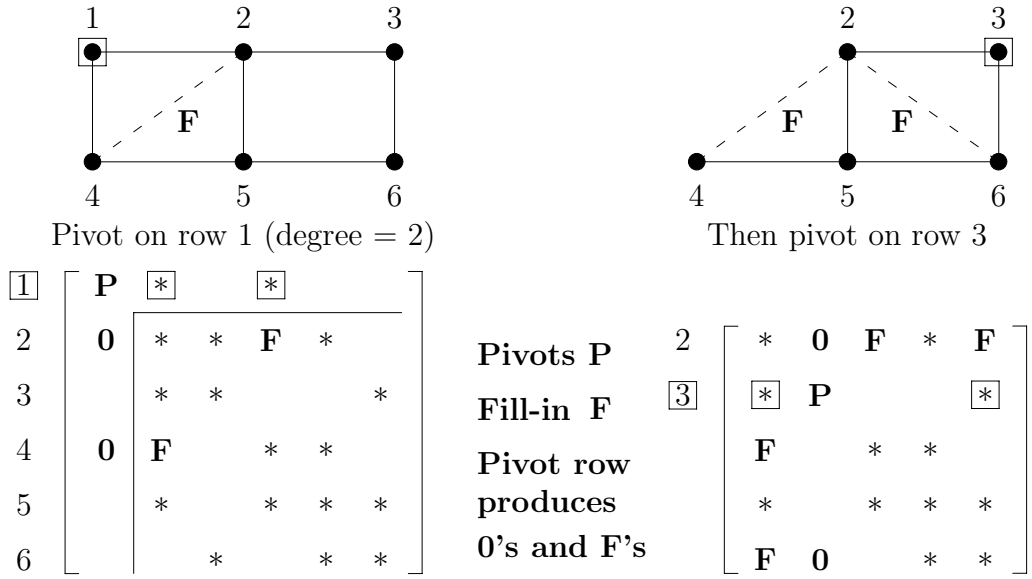


Figure 6.4: Minimum degree nodes 1 and 3 give pivots **P**. New diagonal edges 2–4 and 2–6 in the graph match the first four entries **F** that are filled in by elimination.

and 4, it changes six entries below it (left figure). *The two fill-in entries marked by* **F** *change to nonzeros*. This fill-in of the $(2,4)$ and $(4,2)$ entries corresponds to the dashed line connecting nodes 2 and 4 in the graph.

Elimination continues on the 5 by 5 matrix (and the graph with 5 nodes). Node 2 still has degree 3, so it is not eliminated next. If we break the tie by choosing node 3, elimination using the new pivot **P** will fill in the $(2,6)$ and $(6,2)$ positions. *Node 2 becomes linked to node 6 because they were both linked to the eliminated node 3.*

The problem is reduced to 4 by 4, for the unknown U's at the remaining nodes 2, 4, 5, 6. Problem ____ asks you to take the next step—choose a minimum degree node and reduce the system to 3 by 3. Figure 6.5 shows the start of a minimum degree ordering for a larger grid. Notice how fill-in (16 edges, 32 **F**'s) increases the degrees.

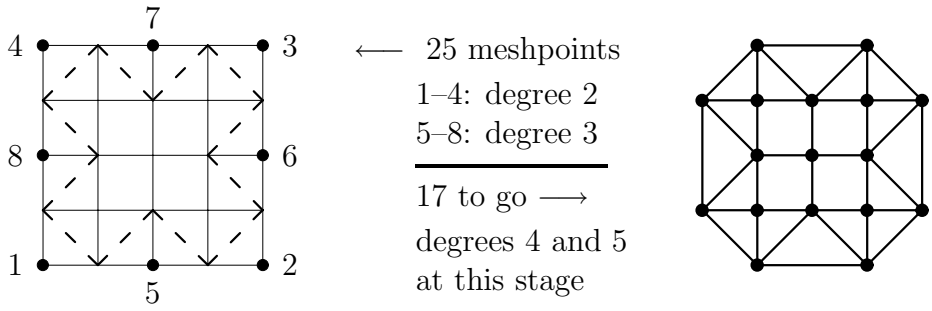


Figure 6.5: Nodes connected to an eliminated node become connected to each other.

Storing the Nonzero Structure = Sparsity Pattern

A large system $KU = F$ needs a fast and economical storage of the node connections (the positions i, j of nonzeros in the matrix). The internal list of edges and nonzero positions will change as elimination proceeds. Normally we don't see that list.

Here we create the list for $N = 4$ by $[i, j, s] = \texttt{find}(K)$. This has $\texttt{nnz}(K) = 10$:

$$\boldsymbol{i} = \mathbf{1\ \ 2\ \ 1\ \ 2\ \ 3\ \ 2\ \ 3\ \ 4\ \ 3\ \ 4} \qquad \boldsymbol{j} = \mathbf{1\ \ 1\ \ 2\ \ 2\ \ 2\ \ 3\ \ 3\ \ 3\ \ 4\ \ 4}$$

$$\boldsymbol{s} = \mathbf{2\ \ -1\ \ -1\ \ 2\ \ -1\ \ -1\ \ 2\ \ -1\ \ -1\ \ 2}$$

The 10 entries are in s, so the fifth nonzero has $i = 3$, $j = 2$, and $s = K_{32} = -1$. The positions i, j of nonzeros in K are in lexicographical order.

You can see that the list j of column indices should be compressed. All we need are *pointers* to indicate when a new column appears on the list. In reality j is replaced by this short list of pointers to the list, including a last pointer to position 11 that signals the stop:

$$\textbf{point} = \mathbf{1\ \ 3\ \ 5\ \ 8\ \ 11}$$

can be updated by $\texttt{perm}(\textbf{point})$ when columns are reordered.

The sparse backslash command $U = K\backslash F$ uses an approximate minimum degree algorithm. First it checks the nonzero pattern to see if row and column permutations P_1 and P_2 can produce a **block triangular form**. The reordered system is $\boldsymbol{P_1 K P_2^{\mathrm{T}}(P_2 U) = P_1 F}$:

$$\textbf{Block triangular matrix} \qquad \boldsymbol{P_1 K P_2^{\mathrm{T}}} = \begin{bmatrix} B_{11} & B_{12} & \cdot & \cdot \\ 0 & B_{22} & \cdot & \cdot \\ 0 & 0 & \cdot & \cdot \\ 0 & 0 & 0 & B_{mm} \end{bmatrix}.$$

The reordered unknown P_2U and the right-hand side P_1F are similarly partitioned into m blocks. ***Block back-substitution*** starts with the smaller problem $B_{mm}U_m = F_m$. Working upwards, we only deal with the blocks B_{ii} on the diagonal (the smaller the better). Surprisingly often, a block triangular form is available.

To preserve symmetry we need $P_1 = P_2$. In the positive definite case, the Cholesky command `chol` is preferred to `lu`, since useless row exchanges can be safely omitted. If $\texttt{diag}(K)$ is positive, there is a chance (not a certainty!) of positive definiteness. Backslash will try `chol` until it is forced to accept row exchanges.

It is understood that MATLAB is not tuned for high performance. Use it for tests and experiments and adjustments. Established codes often turn to C^{++}.

Graph Separators

Here is another approach to ordering, different from minimum degree. The whole graph or mesh is separated into disjoint pieces by a cut. This separator goes through

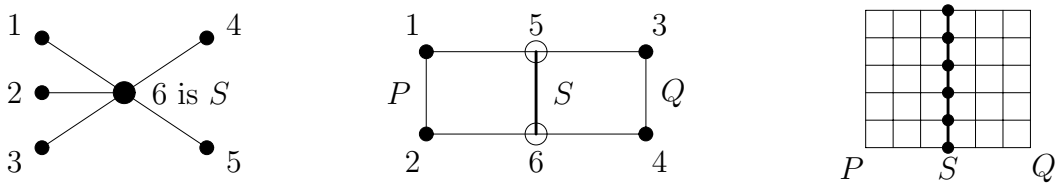
a small number of nodes or meshpoints. *It is a good idea to number the nodes in the separator last.* Elimination is relatively fast for the disjoint pieces P and Q. It only slows down at the end, for the (smaller) separator S.

The meshpoints in P have no direct connections to Q (both are connected to the separator S). Numbered in that order, the "block arrow" stiffness matrix has two blocks of zeros. Its $K = LU$ factorization looks like this:

$$\boldsymbol{K} = \begin{bmatrix} K_{\mathrm{P}} & \mathbf{0} & K_{\mathrm{PS}} \\ \mathbf{0} & K_{\mathrm{Q}} & K_{\mathrm{QS}} \\ K_{\mathrm{SP}} & K_{\mathrm{SQ}} & K_{\mathrm{S}} \end{bmatrix} \quad \boldsymbol{L} = \begin{bmatrix} L_{\mathrm{P}} & & \\ \mathbf{0} & L_{\mathrm{Q}} & \\ X & Y & Z \end{bmatrix} \quad \boldsymbol{U} = \begin{bmatrix} U_{\mathrm{P}} & \mathbf{0} & A \\ & U_{\mathrm{Q}} & B \\ & & C \end{bmatrix} \quad (5)$$

The zero blocks in K give zero blocks in L and U. The submatrix K_{P} produces L_{P} and U_{P}. Then come the factors $L_{\mathrm{Q}}U_{\mathrm{Q}}$ of K_{Q}, followed by the connections through the separator. The major cost is often that last step, the solution of a fairly dense system. When the separator S has size N, a dense system costs N^3.

A separator illustrates the key idea of **domain decomposition**: *Cut the problem into smaller pieces.* This is natural for structural analysis of an airplane: Solve separately for the wings and the fuselage. The smaller system for the separator (where the pieces meet) is like the third row of equation (5). This expresses the requirement that the unknown and its normal derivative (stress or flux) match along the separator. We apologize that a full discussion of domain decomposition is impossible (see [–]).



Arrow matrix (Figure 6.3)

Separator comes last (Figure 6.4)

Blocks P, Q Separator S

Figure 6.6: A graph separator numbered last produces a block arrow matrix K.

Figure 6.6 shows three examples, each with separators. The graph for a perfect arrow matrix has a one-point separator (very unusual). The 6-node rectangle has a two-node separator in the middle. Every N by N grid can be cut by an N-point separator (and N is much smaller than N^2).

On a rectangular grid, the best cut is down the middle in the shorter direction. Our model problem on a square is actually the hardest! A region shaped like a U (or maybe a radiator in 3D) might look difficult. But actually it allows very short separators and fast elimination. A tree needs *no* fill-in (Problem ____).

Nested Dissection

You could say that the numbering of P then Q then S is **block minimum degree**. But one cut with one separator will not come close to an optimal numbering. It is natural to extend the idea to a nested sequence of cuts. P and Q have their own separators at the next level. This **nested dissection** continues until it is not productive to cut further. It is a strategy of "divide and conquer."

Figure 6.7 illustrates three levels of nested dissection on a 7 by 7 grid. The first separator is down the middle. Then two cuts go across and four cuts go down. Numbering the separators last within each stage, the matrix K of size 49 has arrows inside arrows inside arrows. The `spy` command will display the pattern of nonzeros.

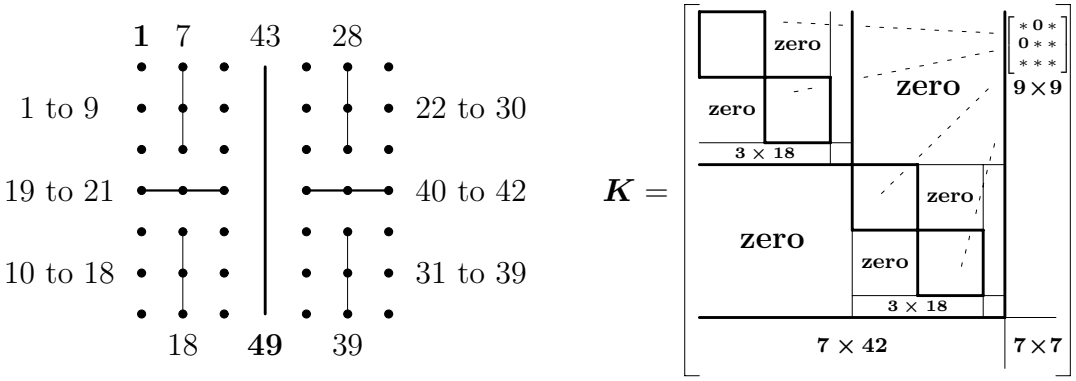


Figure 6.7: Three levels of separators. Still ____ nonzeros in K, only ____ in L.

Separators and nested dissection show how numbering strategies are based on the **graph of nodes**. Edges between nodes correspond to nonzeros in the matrix K. The fill-in created by elimination (entries **F** in L and U) corresponds to adjacent edges in the graph. In practice, there has to be a balance between simplicity and optimality in the numbering—in scientific computing simplicity is a very good thing!

Here are the complexity estimates for the Laplacian with N^2 or N^3 nodes:

Nested Separators	$n = N^2$ in 2D	$n = N^3$ in 3D
Space (nonzeros from fill-in)	$\boldsymbol{N^2 \log N}$	$\boldsymbol{N^4}$
Time (flops for elimination)	$\boldsymbol{N^3}$	$\boldsymbol{N^6}$

In the last century, nested dissection lost out (too slow) on almost all applications. Now larger problems are appearing and the asymptotics eventually give nested dissection an edge. All planar graphs have separators of size $\sqrt{n}$ into nearly equal pieces (P and Q have sizes at most $2n/3$). Of course a new idea for ordering could still win. We don't recommend the older algorithm of (reverse) Cuthill-McKee.

A reasonable compromise is the backslash command $U = K \backslash F$ that uses a nearly minimum degree ordering in Sparse MATLAB.

The text [GL] by George and Liu is the classic reference for this section on ordering of the nodes. The new book [,2007] by Davis has a superb description of his recent algorithms and how they are implemented (in MATLAB's backslash and in UMFPACK for sparse unsymmetric systems).

Problem Set 6.1

1 Create $K2D$ for a 4 by 4 square grid with $N^2 = 3^2$ interior mesh points (so $n = 9$). Print out its factors $K = LU$ (or its Cholesky factor $C = \texttt{chol}(K)$ for the symmetrized form $K = C^{\mathrm{T}}C$). How many zeros in these triangular factors? Also print out $\texttt{inv}(K)$ to see that it is full.

2 As N increases, what parts of the LU factors of $K2D$ are filled in ?

3 Can you answer the same question for $K3D$? In each case we really want an estimate cN^p of the number of nonzeros (the most important number is p).

4 Use the `tic; ...; toc` clocking command or the `cpu` command to compare the solution time for $K2Dx$ = random f in ordinary MATLAB and sparse MATLAB (where $K2D$ is defined as a sparse matrix). Above what value of N does the sparse routine $K\backslash f$ win ?

5 Compare ordinary vs. sparse solution times in the three-dimensional $K3Dx$ = random f. At which N does the sparse $K\backslash f$ begin to win ?

6 Incomplete LU

7 Draw the next step after Figure 6.4 when the matrix has become 4 by 4 and the graph has nodes 2–4–5–6. Which have minimum degree? Is there more fill-in?

8 Redraw the right side of Figure 6.4 if row number 2 is chosen as the second pivot row. Node 2 does not have minimum degree. Indicate new edges in the 5-node graph and new nonzeros **F** in the matrix.

9 Order a tree to avoid all fill-in.

10 Block triangular.

11 Suppose the unknowns U_{ij} on a square grid are stored in an N by N matrix, and not listed in a vector U of length N^2. Show that the vector result of $(K2D)U$ is now produced by the matrix $KU + UK$.

12 Elimination in minimum degree order is started for the square mesh in Figure 6.5. How large do the degrees become as elimination continues? (You could try a large graph, giving systematics instructions for ties in the minimum degree.)

13 Experiment with the red-black permutation in equation ( ), to see how much fill-in is produced by elimination. Is the red-black ordering superior to the original row-by-row ordering?