3.6 Solving Large Linear Systems

Finite elements and finite differences produce large linear systems $KU = F$. *The matrices K are extremely sparse.* They have only a small number of nonzero entries in a typical row. In "physical space" those nonzeros are clustered tightly together—they come from neighboring nodes and meshpoints. But we cannot number N^2 nodes in a plane in any way that keeps 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 can be widely separated)
3. Whether to choose **direct elimination** or an **iterative method**.

That last point will split this section into two parts—elimination methods in 2D (where node order is important) and iterative methods in 3D (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 3.17.

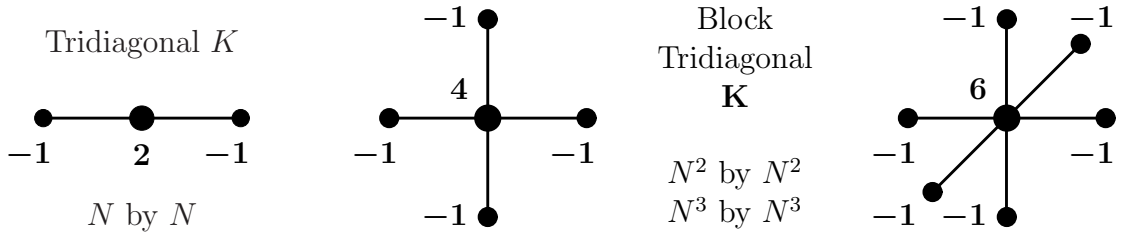


Figure 3.17: $3, 5, 7$ point difference molecules for $-u_{xx}, -u_{xx} - u_{yy}, -u_{xx} - u_{yy} - u_{zz}$.

Along a typical 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 numerical entries). It reflects the fact that $u = 1$ solves Laplace's equation and $U_i = 1$ has differences equal to zero. The constant vector solves $KU = 0$ *except near the boundaries.* When a neighbor is a boundary point where U_i is known, its value moves onto the right side of $KU = F$. Then that row of K is *not zero sum.* Otherwise K would be singular, if $K * \text{ones}(n, 1) = \text{zeros}(n, 1)$.

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

row at a time (this "natural numbering" is not necessarily best). Then the -1's for the neighbor above and the neighbor below are N **positions away from the main diagonal of $\mathbf{K_{2D}}$**. The 2D matrix is block tridiagonal with tridiagonal blocks:

$$K = \begin{bmatrix} 2 & -1 & & \\ -1 & 2 & -1 & \\ & \cdot & \cdot & \cdot \\ & & -1 & 2 \end{bmatrix} \qquad \mathbf{K_{2D}} = \begin{bmatrix} K+2I & -I & & \\ -I & K+2I & -I & \\ & \cdot & \cdot & \cdot \\ & & -I & K+2I \end{bmatrix} \tag{1}$$

Size N Elimination in this order: $\mathbf{K_{2D}}$ has size $n = N^2$

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

The matrix $\mathbf{K_{2D}}$ 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 $\mathbf{K = LU}$ 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.

Those counts are not impossibly large in many practical 2D problems (and we show how they can be reduced). The horrifying large counts come for $\mathbf{K_{3D}}$ in three dimensions. Suppose the 3D grid is numbered by square cross-sections in the natural order $1, \dots, N$. Then $\mathbf{K_{3D}}$ has blocks of order N^2 from those squares. Each square is numbered as above to produce blocks coming from $\mathbf{K_{2D}}$ and $\mathbf{I = I_{2D}}$:

$$\mathbf{K_{3D}} = \begin{bmatrix} \mathbf{K_{2D}}+2\mathbf{I} & -\mathbf{I} & & \\ -\mathbf{I} & \mathbf{K_{2D}}+2\mathbf{I} & -\mathbf{I} & \\ & \cdot & \cdot & \cdot \\ & & -\mathbf{I} & \mathbf{K_{2D}}+2\mathbf{I} \end{bmatrix}$$

Size $n = N^3$
Bandwidth $w = N^2$
Elimination space $nw = N^5$
Elimination time $\approx nw^2 = N^7$

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

The good way to create $\mathbf{K_{2D}}$ from K and I (N by N) is to use the $\mathsf{kron}(A, B)$ command. This *Kronecker product* replaces each entry a_{ij} by the block $a_{ij}B$. To take second differences in all rows at the same time, and then all columns, use kron:

$$\mathbf{K_{2D}} = \mathsf{kron}(K, I) + \mathsf{kron}(I, K)\,. \tag{2}$$

The identity matrix in two dimensions is $\mathbf{I_{2D}} = \mathsf{kron}(I, I)$. This adjusts to allow rectangles, with I's of different sizes, and in three dimensions to allow boxes. For a cube we take second differences inside all planes and also in the z-direction:

$$\mathbf{K_{3D}} = \mathsf{kron}(\mathbf{K_{2D}}, I) + \mathsf{kron}(\mathbf{I_{2D}}, K)\,.$$

Having set up these special matrices $\mathbf{K_{2D}}$ and $\mathbf{K_{3D}}$, we have to say that there are special ways to work with them. The x, y, z directions are separable. The geometry (a box) is also separable. See Section 7.2 on *Fast Poisson Solvers*. Here the matrices K and $\mathbf{K_{2D}}$ and $\mathbf{K_{3D}}$ are serving as models of *the type of matrices that we meet*.

Minimum Degree Algorithm

We now describe (a little roughly) a useful reordering of the nodes and the equations in $\mathbf{K_{2D}U} = \mathbf{F}$. The ordering achieves *minimum degree* at each step—**the number of nonzeros below the pivot row is 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 reasonable.

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 the nonzero entries are postponed as long as possible! That is the key.

The difference is shown in the **arrow matrix** of Figure 3.18. On the left, minimum degree (one nonzero off the diagonal) leads to large bandwidth. But there is *no fill-in*. Elimination will only change its last row and column. The triangular factors L and U have all the same zeros as A. The space for storage stays at $3n$, and elimination needs only n divisions and multiplications and subtractions.

$$\begin{bmatrix} * & & & & & & * \\ & * & & & & & * \\ & & * & & & & * \\ & & & * & & & * \\ & & & & * & & * \\ & & & & & * & * \\ * & * & * & * & * & * & * \end{bmatrix}$$

Bandwidth 6 and 3

Fill-in 0 and 6

$$\begin{bmatrix} * & & & * & & & \\ & * & & * & & & \\ & & * & * & & & \\ * & * & * & * & * & * & * \\ & & & * & * & \mathbf{F} & \mathbf{F} \\ & & & * & \mathbf{F} & * & \mathbf{F} \\ & & & * & \mathbf{F} & \mathbf{F} & * \end{bmatrix}$$

Figure 3.18: Arrow matrix: Minimum degree (no $\mathbf{F}$) against minimum bandwidth.

The second ordering reduces the bandwidth from 6 to 3. But when row 4 is reached as the pivot row, the entries indicated by $\mathbf{F}$ are *filled in*. That full lower quarter of A gives $\frac{1}{8}n^2$ nonzeros to both factors L and U. You see that the whole "profile" of the matrix decides the fill-in, not just the bandwidth.

The **minimum degree algorithm** chooses the $(k+1)$st pivot column, after k columns have been eliminated as usual below the diagonal, by the following rule:

In the remaining matrix of size $n-k$, select the column with the fewest nonzeros.

The component of U corresponding to that column is renumbered $k+1$. So is the node 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. So the algorithm must keep track of the new positions of nonzeros, and also the actual entries. It is the *positions* that decide the ordering of unknowns. Then the *entries* decide the numbers in L and U.

Example Figure 3.19 shows a small example of the minimal degree ordering, for Laplace's 5-point scheme. The node connections produce nonzero entries (indicated by $*$) in $\mathbf{K}$. The problem has six unknowns. $\mathbf{K}$ has two 3 by 3 tridiagonal blocks from horizontal links, and two 3 by 3 blocks with $-I$ from vertical links.

The **degree of a node** is the number of connections to other nodes. This is the number of nonzeros in that column of $\mathbf{K}$. The corner nodes $1, 3, 4, 6$ all have degree 2. Nodes 2 and 5 have degree 3. A larger region has inside nodes of degree 4, which will not be eliminated first. *The degrees change as elimination proceeds, because of fill-in.*

The first elimination step chooses row 1 as pivot row, because node 1 has minimum degree 2. (We had to break a tie! Any degree 2 node could come first, leading to different elimination orders.) The pivot is $\mathbf{P}$, the other nonzeros in that row are boxed. When row 1 operates on rows 2 and 4, it changes six entries below it. In particular, *the two fill-in entries marked by* $\mathbf{F}$ *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.

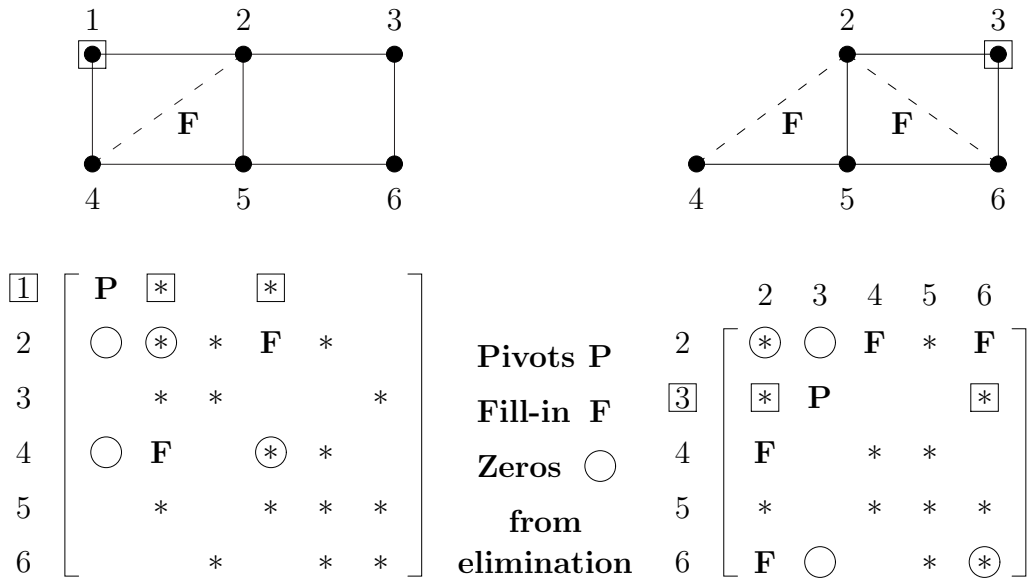


Figure 3.19: Minimum degree nodes 1 and 3. The pivots $\mathbf{P}$ are in rows 1 and 3; new edges 2–4 and 2–6 in the graph match the matrix entries $\mathbf{F}$ filled in by elimination.

Nodes that were connected to the eliminated node are now connected to each other. 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 $\mathbf{P}$ 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.

Storing the Nonzero Structure = Sparsity Pattern

A large system $\mathbf{KU} = \mathbf{F}$ needs a fast and economical storage of the node connections (which match the positions of nonzeros in $\mathbf{K}$). The connections and nonzeros change as elimination proceeds. The list of edges and nonzero positions corresponds to the "*adjacency matrix*" of the graph of nodes. The adjacency matrix has 1 or 0 to indicate nonzero or zero in $\mathbf{K}$.

For each node i, we have a list adj(i) of the nodes connected to i. How to combine these into one master list NZ for the whole graph and the whole matrix $\mathbf{K}$? A simple way is to store the lists adj(i) sequentially in NZ (the nonzeros for $i = 1$ up to $i = n$). An index array IND of pointers tells the starting position of the sublist adj(i) within the master list NZ. It is useful to give IND an $(n+1)$st entry to point to the final entry in NZ (or to the blank that follows, in Figure 3.20). MATLAB will store one more array (the same length nnz($\mathbf{K}$) as NZ) to give the actual nonzero entries.

NZ	|	2	4	|	1	3	5	|	2	6	|	1	5	|	2	4	6	|	3	5	|
		↑			↑				↑			↑			↑				↑			↑
IND		1			3				6			8			10				13			15
Node		1			2				3			4			5				6			

Figure 3.20: Master list **NZ** of nonzeros (neighbors in Figure 3.19). Positions in **IND**.

The indices i are the "original numbering" of the nodes. If there is renumbering, the new ordering can be stored as a permutation PERM. Then PERM(i) $= k$ when the new number i is assigned to the node with original number k. The text [GL] by George and Liu is the classic reference for this entire section on ordering of the nodes.

Graph Separators

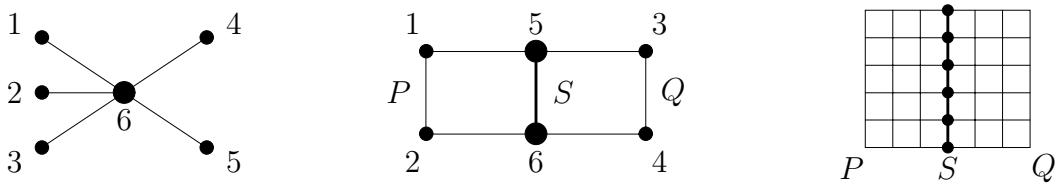
Here is another good ordering, different from minimum degree. Graphs or meshes are often separated into disjoint pieces by a cut. The cut goes through a small number of nodes or meshpoints (a **separator**). *It is a good idea to number the nodes in the*

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 three groups P, Q, S of meshpoints have no direct connections between P and Q (they are both connected to the separator S). Numbered in that order, the "block arrow" stiffness matrix and its $\mathbf{K} = \mathbf{LU}$ factorization look like this:

$$\mathbf{K} = \begin{bmatrix} K_P & \mathbf{0} & K_{PS} \\ \mathbf{0} & K_Q & K_{QS} \\ K_{SP} & K_{SQ} & K_S \end{bmatrix} \quad \mathbf{L} = \begin{bmatrix} L_P & & \\ \mathbf{0} & L_Q & \\ X & Y & Z \end{bmatrix} \quad \mathbf{U} = \begin{bmatrix} U_P & \mathbf{0} & A \\ & U_Q & B \\ & & C \end{bmatrix} \tag{3}$$

The zero blocks in $\mathbf{K}$ give zero blocks in $\mathbf{L}$ and $\mathbf{U}$. The submatrix K_P comes first in elimination, to produce L_P and U_P. Then come the factors $L_Q U_Q$ 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 of the size of the separator.



Arrow matrix (Figure 3.18)

Separator comes last (Figure 3.19)

Blocks P, Q **Separator** S

Figure 3.21: A graph separator numbered last produces a block arrow matrix $\mathbf{K}$.

Figure 3.21 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). If the meshpoints form a rectangle, the best cut is down the middle in the shorter direction.

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

Separators and nested dissection show how numbering strategies are based on the graph of nodes and edges in the mesh. Those edges correspond to nonzeros in the matrix $\mathbf{K}$. The nonzeros created by elimination (filled entries in $\mathbf{L}$ and $\mathbf{U}$) correspond to paths 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!

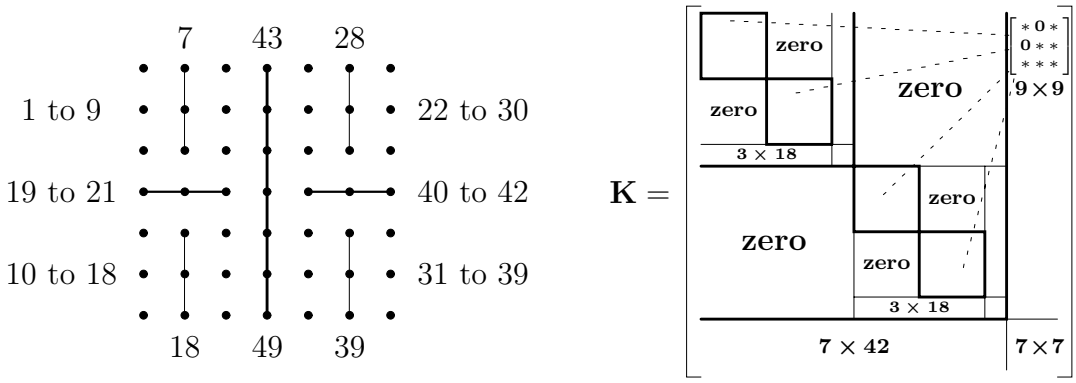


Figure 3.22: Three levels of separators. Still ____ nonzeros in **K**, only ____ in **L**.

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

Operation Counts (page K)

Here are the complexity estimates for the 5-point Laplacian with N^2 or N^3 nodes:

Minimum Degree	$n = N^2$ in 2D	$n = N^3$ in 3D
Space (nonzeros from fill-in)	X	X
Time (flops for elimination)	X	X
Nested Dissection		
Space (nonzeros from fill-in)	X	X
Time (flops for elimination)	X	X

In the last century, nested dissection lost out—it was slower on almost all applications. Now larger problems are appearing and the asymptotics eventually give nested dissection an edge. Algorithms for cutting graphs can produce short cuts into nearly equal pieces. Of course a new idea for ordering could still win.

Iterative versus Direct Methods

This section is a guide to solution methods for problems $Ax = b$ that are too large and expensive for ordinary elimination. We are thinking of *sparse matrices* A, when a multiplication Ax is relatively cheap. If A has at most p nonzeros in every row, then Ax needs at most pn multiplications. Typical applications are to large finite difference equations or finite element problems on unstructured grids. In the special case of a square grid for Laplace's equation, a Fast Poisson Solver (Section 7.2) is available.

We turn away from elimination to **iterative methods** and **Krylov subspaces**. Pure iterative methods are easier to analyze, but the Krylov subspace methods are

more powerful. So the older iterations of Jacobi and Gauss-Seidel and overrelaxation are less favored in scientific computing, compared to **conjugate gradients** and **GMRES**. When the growing Krylov subspaces reach the whole space $\mathbf{R}^n$, these methods (in exact arithmetic) give the exact solution $A^{-1}b$. But in reality we stop much earlier, long before n steps are complete. The conjugate gradient method (*for positive definite A, and with a good preconditioner*) has become truly important.

The next ten pages will introduce you to **numerical linear algebra**. This has become a central part of scientific computing, with a clear goal: *Find a fast stable algorithm that uses the special properties of the matrices.* We meet matrices that are sparse or symmetric or triangular or orthogonal or tridiagonal or Hessenberg or Givens or Householder. Those matrices are at the core of so many computational problems. The algorithm doesn't need details of the entries (which come from the specific application). By using only their structure, numerical linear algebra offers major help.

Overall, elimination with good numbering is the first choice until storage and CPU time become excessive. This high cost often arises first in three dimensions. At that point we turn to iterative methods, which require more expertise. You must choose the method and the preconditioner. The next pages aim to help the reader at this frontier of scientific computing.

Pure Iterations

We begin with old-style pure iteration (not obsolete). The letter K will be reserved for "Krylov" so we leave behind the notation $KU = F$. The linear system becomes $Ax = b$ with a large sparse matrix A, not necessarily symmetric or positive definite:

Linear system $Ax = b$ **Residual** $r_k = b - Ax_k$ **Preconditioner** $P \approx A$

The preconditioner P attempts to be "close to A" and at the same time much easier to work with. A diagonal P is one extreme (not very close). $P = A$ is the other extreme (too close). Splitting the matrix A gives an equivalent form of $Ax = b$:

Splitting $$Px = (P - A)x + b\,. \tag{4}$$

This suggests an iteration, in which every vector x_k leads to the next x_{k+1}:

Iteration $$Px_{k+1} = (P - A)x_k + b\,. \tag{5}$$

Starting from any x_0, the first step finds x_1 from $Px_1 = (P-A)x_0 + b$. The iteration continues to x_2 with the same matrix P, so it often helps to know its triangular factors L and U. Sometimes P itself is triangular, or its factors L and U are approximations to the triangular factors of A. Two conditions on P make the iteration successful:

1. The new x_{k+1} must be quickly computable. Equation (5) must be fast to solve.
2. The errors $e_k = x - x_k$ must converge quickly to zero.

Subtract equation (5) from (4) to find the **error equation**. It connects e_k to e_{k+1}:

$$\textbf{Error} \quad Pe_{k+1} = (P - A)e_k \quad \text{which means} \quad e_{k+1} = (I - P^{-1}A)e_k = Me_k\,. \tag{6}$$

The right side b disappears in this error equation. Each step multiplies the error vector by $M = I - P^{-1}A$. The speed of convergence of x_k to x (and of e_k to zero) depends entirely on M. *The test for convergence is given by the eigenvalues of* M:

Convergence test Every eigenvalue of M must have $|\lambda(M)| < 1$.

The largest eigenvalue (in absolute value) is the **spectral radius** $\rho(M) = \max|\lambda(M)|$. Convergence requires $\rho(M) < 1$. The **convergence rate** is set by the largest eigenvalue. For a large problem, we are happy with $\rho(M) = .9$ and even $\rho(M) = .99$.

Suppose that the initial error e_0 happens to be an eigenvector of M. Then the next error is $e_1 = Me_0 = \lambda e_0$. At every step the error is multiplied by λ, so we must have $|\lambda| < 1$. Normally e_0 is a combination of all the eigenvectors. When the iteration multiplies by M, each eigenvector is multiplied by its own eigenvalue. After k steps those multipliers are λ^k. We have convergence if all $|\lambda| < 1$.

For preconditioner we first propose two simple choices:

Jacobi iteration $P =$ diagonal part of A
Gauss-Seidel iteration $P =$ lower triangular part of A

Typical examples have spectral radius $\rho(M) = 1 - cN^{-1}$. This comes closer and closer to 1 as the mesh is refined and the matrix grows. An improved preconditioner P can give $\rho(M) = 1 - cN^{-1/2}$. Then ρ is smaller and convergence is faster, as in "overrelaxation." But a different approach has given more flexibility in constructing a good P, from a *quick incomplete LU factorization of the true matrix* A:

I*ncomplete *LU $P =$ (approximation to L)(approximation to U)$\,.$

The exact $A = LU$ has fill-in, so zero entries in A become nonzero in L and U. The approximate L and U could ignore this fill-in (fairly dangerous). Or $P = L_{\mathbf{approx}}U_{\mathbf{approx}}$ can keep only the fill-in entries $\mathbf{F}$ above a fixed threshold. The variety of options, and the fact that the computer can decide automatically which entries to keep, has made the ***ILU*** idea (incomplete LU) a very popular starting point.

Example The $-1, 2, -1$ matrix $A = K$ provides an excellent example. We choose the preconditioner $P = T$, the same matrix with $T_{11} = 1$ instead of $K_{11} = 2$. The LU factors of T are perfect first differences, with diagonals of $+1$ and -1. (Remember that all pivots of T equal 1, while the pivots of K are $2/1, 3/2, 4/3, \ldots$) We can compute the right side of $T^{-1}Kx = T^{-1}b$ with only $2N$ additions and no multiplications (just back substitution using L and U). *Idea*: This L and U are approximately correct for K.

The matrix $P^{-1}A = T^{-1}K$ on the left side is triangular. More than that, T is a rank 1 change from K (the $1,1$ entry changes from 2 to 1). It follows that $T^{-1}K$ and $K^{-1}T$

are rank 1 changes from the identity matrix I. A calculation in Problem ____ shows that *only the first column of I is changed*, by the "linear vector" $\ell = (N, N-1, \ldots, 1)$:

$$P^{-1}A = T^{-1}K = I + \ell e_1^{\mathrm{T}} \qquad \text{and} \qquad K^{-1}T = I - (\ell e_1^{\mathrm{T}})/(N+1)\,. \tag{7}$$

Here $e_1^{\mathrm{T}} = \begin{bmatrix} 1 & 0 & \ldots & 0 \end{bmatrix}$ so ℓe_1^{T} has first column ℓ. This example finds $x = K^{-1}b$ by a quick exact formula $(K^{-1}T)T^{-1}b$, needing only $2N$ additions for T^{-1} and N additions and multiplications for $K^{-1}T$. In practice we wouldn't precondition this K (just solve).

The usual purpose of preconditioning is to speed up convergence for iterative methods, and that depends on the eigenvalues of $P^{-1}A$. Here the eigenvalues of $T^{-1}K$ are its diagonal entries $N+1, 1, \ldots, 1$. This example will illustrate a special property of conjugate gradients, that with only two different eigenvalues it reaches the true solution x in two steps.

The iteration $Px_{k+1} = (P-A)x_k + b$ is too simple! It is choosing one particular vector in a "Krylov subspace." With relatively little work we can make a much better choice of x_k. Krylov projections are the state of the art in today's iterative methods.

Krylov Subspaces

Our original equation is $Ax = b$. *The preconditioned equation is* $P^{-1}Ax = P^{-1}b$. When we write P^{-1}, we never intend that an inverse would be explicitly computed (except in our example). The ordinary iteration is a correction to x_k by the vector $P^{-1}r_k$:

$$Px_{k+1} = (P-A)x_k + b \quad \text{or} \quad Px_{k+1} = Px_k + r_k \quad \text{or} \quad x_{k+1} = x_k + P^{-1}r_k\,. \tag{8}$$

Here $r_k = b - Ax_k$ is the **residual**. It is the error in $Ax = b$, not the error e_k in x. The symbol $P^{-1}r_k$ represents the change from x_k to x_{k+1}, but that step is not computed by multiplying P^{-1} times r_k. We might use incomplete LU, or a few steps of a "multigrid" iteration, or "domain decomposition." Or an entirely new preconditioner.

In describing Krylov subspaces, I should work with $P^{-1}A$. **For simplicity I will only write** A. I am assuming that P has been chosen and used, and the preconditioned equation $P^{-1}Ax = P^{-1}b$ is given the notation $Ax = b$. The preconditioner is now $P = I$. Our new matrix A is probably better than the original matrix with that name.

The Krylov subspace $\mathbf{K^k}(\boldsymbol{A}, \boldsymbol{b})$ contains all combinations of $\boldsymbol{b}, \boldsymbol{Ab}, \ldots, \boldsymbol{A^{k-1}b}$.

These are the vectors that we can compute quickly, multiplying by a sparse A. We look in this space $\mathbf{K}^k$ for the approximation x_k to the true solution of $Ax = b$. Notice that the pure iteration $x_k = (I-A)x_{k-1} + b$ does produce a vector in $\mathbf{K}^k$ when x_{k-1} is in $\mathbf{K}^{k-1}$. *The Krylov subspace methods make other choices of* x_k. Here are four different approaches to choosing a good x_k in $\mathbf{K}^k$—this is the important decision:

1. The residual $r_k = b - Ax_k$ is orthogonal to $\mathbf{K}^k$ (**Conjugate Gradients**, ...)

2. The residual r_k has minimum norm for x_k in $\mathbf{K}^k$ (**GMRES**, **MINRES**, ...)

3. r_k is orthogonal to a different space like $\mathbf{K}^k(A^{\mathrm{T}})$ (**BiConjugate Gradients**, ...)

4. e_k has minimum norm (**SYMMLQ**; for **BiCGStab** x_k is in $A^{\mathrm{T}}\mathbf{K}^k(A^{\mathrm{T}})$; ...)

In every case we hope to compute the new x_k quickly and stably from the earlier x's. If that recursion only involves x_{k-1} and x_{k-2} (short recurrence) it is especially fast. We will see this happen for conjugate gradients and symmetric positive definite A. The BiCG method in **3** is a natural extension of short recurrences to unsymmetric A—but stability and other questions open the door to the whole range of methods.

To compute x_k we need a basis for $\mathbf{K}^k$. The best basis $q_1, \ldots, q_k$ is *orthonormal*. Each new q_k comes from orthogonalizing $t = Aq_{k-1}$ to the basis vectors $q_1, \ldots, q_{k-1}$ that are already chosen. This is the Gram-Schmidt idea (called *modified* Gram-Schmidt when we subtract projections of t onto the q's *one at a time*, for numerical stability). The iteration to compute the orthonormal q's is known as **Arnoldi's method**:

```
1  q_1 = b/||b||_2;              % Normalize to ||q_1|| = 1
   for j = 1, ..., k-1
2      t = A q_j;                % t is in the Krylov space K^{j+1}(A, b)
   for i = 1, ..., j
3          h_ij = q_i^T t;       % h_ij q_i = projection of t onto q_i
4          t = t - h_ij q_i;     % Subtract component of t along q_i
       end;
5      h_{j+1,j} = ||t||_2;      % t is now orthogonal to q_1, ..., q_j
6      q_{j+1} = t/h_{j+1,j};    % Normalize t to ||q_{j+1}|| = 1
   end                           % q_1, ..., q_k are orthonormal in K^k
```

Put the column vectors $q_1, \ldots, q_k$ into an n by k matrix Q_k. Multiplying rows of Q_k^{T} by columns of Q_k produces all the inner products $q_i^{\mathrm{T}} q_j$, which are the 0's and 1's in the identity matrix. **The orthonormal property means that** $Q_k^{\mathrm{T}} Q_k = I_k$.

Arnoldi constructs each q_{j+1} from Aq_j by subtracting projections $h_{ij}q_i$. If we express the steps up to $j = k-1$ in matrix notation, they become $AQ_{k-1} = Q_k H_{k,k-1}$:

Arnoldi

$$AQ_{k-1} = \underbrace{\begin{bmatrix} Aq_1 & \cdots & Aq_{k-1} \end{bmatrix}}_{n \text{ by } k-1} = \underbrace{\begin{bmatrix} q_1 & \cdots & q_k \end{bmatrix}}_{n \text{ by } k} \underbrace{\begin{bmatrix} h_{11} & h_{12} & \cdot & h_{1,k-1} \\ h_{21} & h_{22} & \cdot & h_{2,k-1} \\ 0 & h_{23} & \cdot & \cdot \\ 0 & 0 & \cdot & h_{k,k-1} \end{bmatrix}}_{k \text{ by } k-1}. \quad (9)$$

That matrix $H_{k,k-1}$ is "upper Hessenberg" because it has only one nonzero diagonal below the main diagonal. We check that the first column of this matrix equation

(multiplying by columns!) produces q_2:

$$Aq_1 = h_{11}q_1 + h_{21}q_2 \qquad \text{or} \qquad q_2 = \frac{Aq_1 - h_{11}q_1}{h_{21}}\,. \tag{10}$$

That subtraction is Step **4** in Arnoldi's algorithm. Division by h_{21} is Step **6**.

Unless more of the h_{ij} are zero, the cost is increasing at every iteration. We have k dot products to compute at step **3** and **5**, and k vector updates in steps **4** and **6**. A *short recurrence* means that most of these h_{ij} are zero. That happens when $A = A^{\mathrm{T}}$.

The matrix H is tridiagonal when A is symmetric. This fact is the foundation of conjugate gradients. For a matrix proof, multiply equation (9) by Q_{k-1}^{T}. The right side becomes H without its last row, because $(Q_{k-1}^{\mathrm{T}}Q_k)H_{k,k-1} = [\,I \;\; \mathbf{0}\,]\,H_{k,k-1}$. The left side $Q_{k-1}^{\mathrm{T}}AQ_{k-1}$ is always symmetric when A is symmetric. So that H matrix has to be symmetric, *which makes it tridiagonal*. There are only three nonzeros in the rows and columns of H, and Gram-Schmidt to find q_{k+1} only involves q_k and q_{k-1}:

$$\textbf{Arnoldi when } \boldsymbol{A = A^{\mathrm{T}}} \qquad Aq_k = h_{k+1,k}\,q_{k+1} + h_{k,k}\,q_k + h_{k-1,k}\,q_{k-1}\,. \tag{11}$$

This is the *Lanczos iteration*. Each new $q_{k+1} = (Aq_k - h_{k,k}q_k - h_{k-1,k}q_{k-1})/h_{k+1,k}$ involves one multiplication Aq_k, two dot products for new h's, and two vector updates.

The QR Method for Eigenvalues

Allow me an important comment on the *eigenvalue problem* $Ax = \lambda x$. We have seen that $H_{k-1} = Q_{k-1}^{\mathrm{T}}AQ_{k-1}$ is tridiagonal if $A = A^{\mathrm{T}}$. When $k-1$ reaches n and Q_n is square, the matrix $H = Q_n^{\mathrm{T}}AQ_n = Q_n^{-1}AQ_n$ has the same eigenvalues as A:

$$\textbf{Same } \boldsymbol{\lambda} \qquad Hy = Q_n^{-1}AQ_n y = \lambda y \quad \text{gives} \quad Ax = \lambda x \;\; \text{with} \;\; x = Q_n y\,. \tag{12}$$

It is much easier to find the eigenvalues λ for a tridiagonal H than the for original A.

The famous "**QR method**" for the eigenvalue problem starts with $T_1 = H$, factors it into $T_1 = Q_1R_1$ (this is Gram-Schmidt on the short columns of T_1), and reverses order to produce $T_2 = R_1Q_1$. The matrix T_2 is again tridiagonal, and its off-diagonal entries are normally smaller than for T_1. The next step is Gram-Schmidt on T_2, orthogonalizing its columns in Q_2 by the combinations in the upper triangular R_2:

$$\textbf{QR Method} \;\; \text{Factor } T_2 \text{ into } Q_2R_2\,. \text{ Reverse order to } T_3 = R_2Q_2 = Q_2^{-1}T_2Q_2 \tag{13}$$

By the reasoning in (12), *any* $Q^{-1}TQ$ has the same eigenvalues as T. So the matrices $T_2, T_3, \ldots$ all have the same eigenvalues as $T_1 = H$ and A. (These square Q_k from Gram-Schmidt are entirely different from the rectangular Q_k in Arnoldi.) We can even *shift* T before Gram-Schmidt, and we should, provided we remember to shift back:

$$\textbf{Shifted QR} \quad \text{Factor } \; T_k - s_kI = Q_kR_k\,. \; \text{Reverse to } \; T_{k+1} = R_kQ_k + b_kI\,. \tag{14}$$

When the shift s_k is chosen to be the n,n entry of T_k, the last off-diagonal entry of T_{k+1} becomes very small. The n,n entry of T_{k+1} moves close to an eigenvalue. *Shifted*

QR *is one of the great algorithms of numerical linear algebra.* It solves moderate-size eigenvalue problems with great efficiency. This is the core of MATLAB's eig(A).

For a *large* symmetric matrix, we often stop the Arnoldi-Lanczos iteration at a tridiagonal H_k with $k < n$. The full n-step process to reach H_n is too expensive, and often we don't need all n eigenvalues. So we compute (by the same QR method) the k eigenvalues of H_k instead of the n eigenvalues of H_n. These computed $\lambda_{1k}, \lambda_{2k}, \ldots, \lambda_{kk}$ can provide good approximations to the first k eigenvalues of A. And we have an excellent start on the eigenvalue problem for H_{k+1}, if we decide to take a further step.

This **Lanczos method** will find, approximately and iteratively and quickly, the leading eigenvalues of a large symmetric matrix.

The Conjugate Gradient Method

We return to iterative methods for $Ax = b$. The Arnoldi algorithm produced orthonormal basis vectors $q_1, q_2, \ldots$ for the growing Krylov subspaces $\mathbf{K}^1, \mathbf{K}^2, \ldots$. Now we select vectors $x_1, x_2, \ldots$ in those subspaces that approach the exact solution to $Ax = b$. We concentrate on the *conjugate gradient method for symmetric positive definite* A.

The rule for x_k in conjugate gradients is that the residual $r_k = b - Ax_k$ should be orthogonal to all vectors in $\mathbf{K}^k$. Since r_k will be in $\mathbf{K}^{k+1}$, it must be a multiple of Arnoldi's next vector q_{k+1}! Each residual is therefore orthogonal to all previous residuals (which are multiples of the previous q's):

Orthogonal residuals $$r_i^{\mathrm{T}} r_k = 0 \quad \text{for } i < k\,. \tag{15}$$

The difference between r_k and q_{k+1} is that the q's are normalized, as in $q_1 = b/\|b\|$.

Since r_{k-1} is a multiple of q_k, the difference $r_k - r_{k-1}$ is orthogonal to each subspace $\mathbf{K}^i$ with $i < k$. Certainly $x_i - x_{i-1}$ lies in that $\mathbf{K}^i$. So Δr is orthogonal to earlier Δx's:

$$(x_i - x_{i-1})^{\mathrm{T}}(r_k - r_{k-1}) = 0 \quad \text{for } i < k\,. \tag{16}$$

These differences Δx and Δr are directly connected, because the b's cancel in Δr:

$$r_k - r_{k-1} = (b - Ax_k) - (b - Ax_{k-1}) = -A(x_k - x_{k-1})\,. \tag{17}$$

Substituting (17) into (16), the updates in the x's are "A-orthogonal" or **conjugate**:

Conjugate updates Δx $$(x_i - x_{i-1})^{\mathrm{T}} A(x_k - x_{k-1}) = 0 \quad \text{for } i < k\,. \tag{18}$$

Now we have all the requirements. Each conjugate gradient step will find a new "search direction" d_k for the update $x_k - x_{k-1}$. From x_{k-1} it will move the right distance $\alpha_k d_k$ to x_k. Using (17) it will compute the new r_k. The constants β_k in the search direction and α_k in the update will be determined by (15) and (16) for $i = k - 1$. For symmetric A the orthogonality in (15) and (16) will be automatic for $i < k - 1$, as in Arnoldi. We have a "short recurrence" for the new x_k and r_k.

Here is one cycle of the algorithm, starting from $x_0 = 0$ and $r_0 = b$ and $\beta_1 = 0$. It involves only two new dot products and one matrix-vector multiplication Ad:

Conjugate	**1**	$\beta_k = r_{k-1}^{\mathrm{T}} r_{k-1} / r_{k-2}^{\mathrm{T}} r_{k-2}$	% Improvement this step
Gradient	**2**	$d_k = r_{k-1} + \beta_k d_{k-1}$	% Next search direction
Method	**3**	$\alpha_k = r_{k-1}^{\mathrm{T}} r_{k-1} / d_k^{\mathrm{T}} A d_k$	% Step length to next x_k
	4	$x_k = x_{k-1} + \alpha_k d_k$	% Approximate solution
	5	$r_k = r_{k-1} - \alpha_k A d_k$	% New residual from (17)

The formulas **1** and **3** for β_k and α_k are explained briefly below—and fully by Trefethen-Bau ( ) and Shewchuk ( ) and many other good references.

Different Viewpoints on Conjugate Gradients

I want to describe the (same!) conjugate gradient method in two different ways:

1. It solves a tridiagonal system $Hy = f$ recursively

2. It minimizes the energy $\frac{1}{2}x^{\mathrm{T}}Ax - x^{\mathrm{T}}b$ recursively.

How does $Ax = b$ change to the tridiagonal $Hy = f$? That uses Arnoldi's orthonormal columns $q_1, \ldots, q_n$ in Q, with $Q^{\mathrm{T}}Q = I$ and $Q^{\mathrm{T}}AQ = H$:

$$Ax = b \quad \text{is} \quad (Q^{\mathrm{T}}AQ)(Q^{\mathrm{T}}x) = Q^{\mathrm{T}}b \quad \text{which is} \quad Hy = f = (\|b\|, 0, \ldots, 0)\,. \tag{19}$$

Since q_1 is $b/\|b\|$, the first component of $f = Q^{\mathrm{T}}b$ is $q_1^{\mathrm{T}}b = \|b\|$ and the other components are $q_i^{\mathrm{T}}b = 0$. The conjugate gradient method is implicitly computing this symmetric tridiagonal H and updating the solution y at each step. Here is the third step:

$$H_3 y_3 = \begin{bmatrix} h_{11} & h_{12} & \\ h_{21} & h_{22} & h_{23} \\ & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} \\ y_3 \\ \\ \end{bmatrix} = \begin{bmatrix} \|b\| \\ 0 \\ 0 \end{bmatrix}. \tag{20}$$

This is the equation $Ax = b$ *projected* by Q_3 onto the third Krylov subspace $\mathbf{K}^3$.

These h's never appear in conjugate gradients. We don't want to do Arnoldi too! It is the LDL^{T} factors of H that CG is somehow computing—two new numbers at each step. Those give a fast update from y_{j-1} to y_j. The corresponding $x_j = Q_j y_j$ from conjugate gradients approaches the exact solution $x_n = Q_n y_n$ which is $x = A^{-1}b$.

If we can see conjugate gradients also as an energy minimizing algorithm, we can extend it to nonlinear problems and use it in optimization. For our linear equation $Ax = b$, the energy is $E(x) = \frac{1}{2}x^{\mathrm{T}}Ax - x^{\mathrm{T}}b$. Minimizing $E(x)$ is the same as solving $Ax = b$, when A is positive definite (the main point of Section 1.). **The CG iteration minimizes $E(x)$ on the growing Krylov subspaces**. On the first subspace $\mathbf{K}^1$, the line where x is $\alpha b = \alpha d_1$, this minimization produces the right

value for α_1:

$$E(\alpha b) = \frac{1}{2}\alpha^2 b^{\mathrm{T}} A b - \alpha b^{\mathrm{T}} b \quad \text{is minimized at} \quad \alpha_1 = \frac{b^{\mathrm{T}} b}{b^{\mathrm{T}} A b}\,. \tag{21}$$

That α_1 is the constant chosen in step **3** of the first conjugate gradient cycle.

The gradient of $E(x) = \frac{1}{2}x^{\mathrm{T}}Ax - x^{\mathrm{T}}b$ is exactly $Ax - b$. *The steepest descent direction at x_1 is along the negative gradient, which is r_1!* This sounds like the perfect direction d_2 for the next move. But the great difficulty with steepest descent is that this r_1 can be too close to the first direction. Little progress that way. So we add the right multiple $\beta_2 d_1$, in order to make $d_2 = r_1 + \beta_2 d_1$ A-orthogonal to the first direction d_1.

Then we move in this conjugate direction d_2 to $x_2 = x_1 + \alpha_2 d_2$. This explains the name *conjugate gradients*, rather than the pure gradients of steepest descent. Every cycle of CG chooses α_j to minimize $E(x)$ in the new search direction $x = x_{j-1} + \alpha d_j$. The last cycle (if we go that far) gives the overall minimizer $x_n = x = A^{-1}b$.

Example

$$Ax = b \quad \text{is} \quad \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} 3 \\ -1 \\ -1 \end{bmatrix} = \begin{bmatrix} 4 \\ 0 \\ 0 \end{bmatrix}.$$

From $x_0 = 0$ and $\beta_1 = 0$ and $r_0 = d_1 = b$ the first cycle gives $\alpha_1 = \frac{1}{2}$ and $x_1 = \frac{1}{2}b = (2, 0, 0)$. The new residual is $r_1 = b - Ax_1 = (0, -2, -2)$. Then the second cycle yields

$$\beta_2 = \frac{8}{16}, \quad d_2 = \begin{bmatrix} 2 \\ -2 \\ -2 \end{bmatrix}, \quad \alpha_2 = \frac{8}{16}, \quad x_2 = \begin{bmatrix} 3 \\ -1 \\ -1 \end{bmatrix} = A^{-1}b\,!$$

The correct solution is reached in two steps, where normally it will take $n = 3$ steps. The reason is that this particular A has only two distinct eigenvalues 4 and 1. In that case $A^{-1}b$ is a combination of b and Ab, and this best combination x_2 is found at cycle 2. The residual r_2 is zero and the cycles stop early—very unusual.

Energy minimization leads in [ ] to an estimate of the convergence rate for the error $e = x - x_j$ in conjugate gradients, using the A-norm $\|e\|_A = \sqrt{e^{\mathrm{T}}Ae}$:

Error estimate $$\|x - x_j\|_A \le 2\left(\frac{\sqrt{\lambda_{\max}} - \sqrt{\lambda_{\min}}}{\sqrt{\lambda_{\max}} + \sqrt{\lambda_{\min}}}\right)^j \|x - x_0\|_A\,. \tag{22}$$

This is the best-known error estimate, although it doesn't account for any clustering of the eigenvalues of A. It involves only the condition number $\lambda_{\max}/\lambda_{\min}$. Problem ____ gives the "optimal" error estimate but it is not so easy to compute. That optimal estimate needs all the eigenvalues of A, while (22) uses only the extreme eigenvalues $\lambda_{\max}(A)$ and $\lambda_{\min}(A)$—which in practice we can bound above and below.

Minimum Residual Methods

When A is not symmetric positive definite, conjugate gradient is not guaranteed to solve $Ax = b$. Most likely it won't. We will follow van der Vorst [ ] in briefly describing the minimum norm residual approach, leading to MINRES and GMRES.

These methods choose x_j in the Krylov subspace $\mathbf{K}^j$ so that $\|b - Ax_j\|$ is minimal. First we compute the orthonormal Arnoldi vectors $q_1, \ldots, q_j$. They go in the columns of Q_j, so $Q_j^{\mathrm{T}} Q_j = I$. As in (19) we set $x_j = Q_j y$, to express the solution as a combination of those q's. Then the norm of the residual r_j using (9) is

$$\|b - Ax_j\| = \|b - AQ_j y\| = \|b - Q_{j+1} H_{j+1,j} y\| \,. \tag{23}$$

These vectors are all in the Krylov space $\mathbf{K}^{j+1}$, where $r_j^{\mathrm{T}}(Q_{j+1}Q_{j+1}^{\mathrm{T}} r_j) = r_j^{\mathrm{T}} r_j$. This says that the norm is not changed when we multiply by Q_{j+1}^{T}. Our problem becomes:

$$\textbf{Choose } y \textbf{ to minimize} \quad \|r_j\| = \|Q_{j+1}^{\mathrm{T}} b - H_{j+1,j}\, y\| = \|f - Hy\| \,. \tag{24}$$

This is an ordinary least squares problem for the equation $Hy = f$ with only $j+1$ equations and j unknowns. The right side $f = Q_{j+1}^{\mathrm{T}} b$ is $(\|r_0\|, 0, \ldots, 0)$ as in (19). The matrix $H = H_{j+1,j}$ is Hessenberg as in (9), with one nonzero diagonal below the main diagonal. We face a completely typical problem of numerical linear algebra: *Use the special properties of H and f to find a fast algorithm that computes y.* The two favorite algorithms for this least squares problem are closely related:

MINRES A is symmetric (probably indefinite, or we use CG) and H is tridiagonal

GMRES A is *not* symmetric and the upper triangular part of H can be full

In both cases we want to clear out that nonzero diagonal below the main diagonal of H. The natural way to do that, one nonzero entry at a time, is by "*Givens rotations.*" These plane rotations are so useful and simple (the essential part is only 2 by 2) that we complete this section by explaining them.

Givens Rotations

The direct approach to the least squares solution of $Hy = f$ constructs the normal equations $H^{\mathrm{T}} H\widehat{y} = H^{\mathrm{T}} f$. That was the central idea in Chapter 1, but you see what we lose. If H is Hessenberg, with many good zeros, $H^{\mathrm{T}} H$ is full. Those zeros in H should simplify and shorten the computations, so we don't want the normal equations.

The other approach to least squares is by Gram-Schmidt. **We factor H into orthogonal times upper triangular**. Since the letter Q is already used, the orthogonal matrix will be called G (after Givens). The upper triangular matrix is $G^{-1}H$. The 3 by 2 case shows how a plane rotation G_{21}^{-1} can clear out the subdiagonal entry

h_{21}:

$$G_{21}^{-1}H = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \\ 0 & h_{32} \end{bmatrix} = \begin{bmatrix} * & * \\ \mathbf{0} & * \\ 0 & * \end{bmatrix}. \tag{25}$$

That bold zero entry requires $h_{11}\sin\theta = h_{21}\cos\theta$, which determines θ. A second rotation G_{32}^{-1}, in the 2-3 plane, will zero out the $3,2$ entry. Then $G_{32}^{-1}G_{21}^{-1}H$ is a square upper triangular matrix U above a row of zeros!

The Givens orthogonal matrix is $G = G_{21}G_{32}$ but there is no reason to do this multiplication. We use each G_{ij} as it is constructed, to simplify the least squares problem. Rotations (and all orthogonal matrices) leave the lengths of vectors unchanged:

$$\|Hy - f\| = \|G_{32}^{-1}G_{21}^{-1}Hy - G_{32}^{-1}G_{21}^{-1}f\| = \| \begin{bmatrix} U \\ 0 \end{bmatrix} y - \begin{bmatrix} F \\ e \end{bmatrix} \|. \tag{26}$$

This length is what MINRES and GMRES minimize. The row of zeros below U means that the last entry e is the error—we can't reduce it. But we get all the other entries exactly right by solving the j by j system $Uy = F$ (here $j = 2$). This gives the best least squares solution y. Going back to the original problem of minimizing $\|r\| = \|b - Ax_j\|$, the best x_j in the Krylov space $\mathbf{K}^j$ is $Q_j y$.

For non-symmetric A (GMRES rather than MINRES) we don't have a short recurrence. The upper triangle in H can be full, and step j becomes expensive and possibly inaccurate as j increases. So we may change "full GMRES" to GMRES(m), which restarts the algorithm every m steps. It is not so easy to choose a good m.

Problem Set 3.6

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

2 As N increases, what parts of the $\mathbf{LU}$ factors of $\mathbf{K_{2D}}$ are filled in?

3 Can you answer the same question for $\mathbf{K_{3D}}$? 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 to compare the solution time for $\mathbf{K_{2D}x} =$ random $\mathbf{f}$ in ordinary MATLAB and sparse MATLAB (where $\mathbf{K_{2D}}$ is defined as a sparse matrix). Above what value of N does the sparse routine $\mathbf{K}\backslash\mathbf{f}$ win?

5 Compare ordinary vs. sparse solution times in the three-dimensional $\mathbf{K_{3D}x} =$ random $\mathbf{f}$. At which N does the sparse $\mathbf{K}\backslash\mathbf{f}$ begin to win?

6 Incomplete LU

7 Conjugate gradients

8 Draw the next step after Figure 3.19 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?

9 Redraw the right side of Figure 3.19 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.

10 To show that $T^{-1}K = I + \ell e_1^{\mathrm{T}}$ in (7), with $e_1^{\mathrm{T}} = [\,1\ 0\ \ldots\ 0\,]$, we can start from $K = T + e_1 e_1^{\mathrm{T}}$. Then $T^{-1}K = I + (T^{-1}e_1)e_1^{\mathrm{T}}$ and we verify that $e_1 = T\ell$:

$$T\ell = \begin{bmatrix} 1 & -1 & & \\ -1 & 2 & -1 & \\ & \cdot & \cdot & \cdot \\ & & -1 & 2 \end{bmatrix} \begin{bmatrix} N \\ N-1 \\ \cdot \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ \cdot \\ 0 \end{bmatrix} = e_1\,.$$

Second differences of a linear vector ℓ are zero. Now multiply $T^{-1}K = I + \ell e_1^{\mathrm{T}}$ times $I - (\ell e_1^{\mathrm{T}})/(N+1)$ to establish the inverse matrix $K^{-1}T$ in (7).

11 Arnoldi expresses each Aq_k as $h_{k+1,k}q_{k+1} + h_{k,k}q_k + \cdots + h_{1,k}q_1$. Multiply by q_i^{T} to find $h_{i,k} = q_i^{\mathrm{T}}Aq_k$. *If A is symmetric* you can write this as $(Aq_i)^{\mathrm{T}}q_k$. Explain why $(Aq_i)^{\mathrm{T}}q_k = 0$ for $i < k-1$ by expanding Aq_i into $h_{i+1,i}q_{i+1} + \cdots + h_{1,i}q_1$. We have a *short recurrence if* $A = A^{\mathrm{T}}$ (only $h_{k+1,k}$ and $h_{k,k}$ and $h_{k-1,k}$ are nonzero).