3.5 Finite Differences and Fast Poisson Solvers

It is extremely unusual to use eigenvectors to solve a linear system $KU = F$. You need to know all the eigenvectors of K, and (much more than that) the eigenvector matrix S must be especially fast to work with. Both S and S^{-1} are required, because $K^{-1} = S\Lambda^{-1}S^{-1}$. The eigenvalue matrices Λ and Λ^{-1} are diagonal and quick. When the derivatives in Poisson's equation $-u_{xx} - u_{yy} = f(x, y)$ are replaced by second differences, we do know the eigenvectors (discrete sines in the columns of S). Then the Fourier transform quickly inverts and multiplies by S.

On a square mesh those differences have $-1, 2, -1$ in the x-direction and $-1, 2, -1$ in the y-direction (divided by h^2, where h = meshwidth). Figure 3.20 shows how the second differences combine into a "5-*point molecule*" for the discrete Laplacian. Boundary values $u = u_0(x, y)$ are assumed to be given along the sides of a unit square.

The square mesh has N interior points in each direction ($N = 5$ in the figure). In this case there are $n = N^2 = 25$ unknown mesh values U_{ij}. When the molecule is centered at position (i, j), the discrete Poisson equation gives a row of $K2\text{D}\,U = F$:

$$\boldsymbol{K}\mathbf{2D\,U = F} \qquad 4u_{ij} - u_{i,j-1} - u_{i-1,j} - u_{i+1,j} - u_{i,j+1} = h^2 f(ih, jh) = F_{ij}\,. \quad (1)$$

The inside rows of $\boldsymbol{K}$2D have five nonzero entries 4, −1, −1, −1, −1. When (i, j) is next to a boundary point of the square, the known value of u_0 at that neighboring boundary point moves to the right side of equation (1). It becomes part of the vector F, and a -1 drops out of the corresponding row of K. So K2D has five nonzeros on inside rows and fewer nonzeros on next-to-boundary rows.

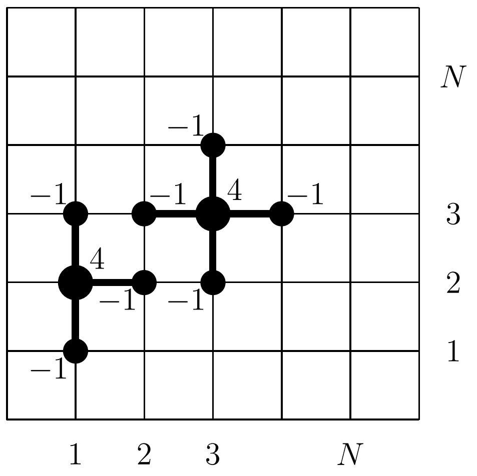

Figure 3.20: 5-point molecules at inside points (fewer -1's next to boundary).

This matrix K2D is sparse. Using blocks of size N, we can create the 2D matrix from the familiar N by N second difference matrix K. 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 $\boldsymbol{K}$**2D**. The 2D matrix is *block tridiagonal with tridiagonal blocks*:

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

Size $\boldsymbol{N}$ **Size $\boldsymbol{n = N^2}$** **Bandwidth $\boldsymbol{w = N}$**
Time $\boldsymbol{N}$ **Space $\boldsymbol{nw = N^3}$** **Time $\boldsymbol{nw^2 = N^4}$**

The matrix K2D has 4's down the main diagonal in equation (1). Its bandwidth $w = N$ is the distance from the main diagonal to the nonzeros in $-I$. Many of the spaces in between are filled during elimination! This is discussed in Section 6.1.

Kronecker product One good way to create K2D from K and I is by the `kron` command. When A and B have size N by N, the matrix `kron`(A, B) is N^2 by N^2. *Each number a_{ij} is replaced by the block $a_{ij}B$.*

To take second differences in all rows at the same time, `kron`(I, K) produces a block diagonal matrix of K's. In the y-direction, `kron`(K, I) multiplies -1 and 2 and -1 by I (dealing with a column of meshpoints at a time). Add x and y differences:

$$\boldsymbol{K}\mathbf{2D} = \texttt{kron}(I, K) + \texttt{kron}(K, I) = \begin{bmatrix} K & & \\ & K & \\ & & \cdot \end{bmatrix} + \begin{bmatrix} 2I & -I & \cdot \\ -I & 2I & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \tag{3}$$

This sum agrees with the 5-point matrix in (2). The computational question is how to work with K2D. We will propose three methods:

1. **Elimination in a good order** (not using the special structure of K2D)
2. **Fast Poisson Solver** (applying the FFT = Fast Fourier Transform)
3. **Odd-Even Reduction** (since K2D is block tridiagonal).

The novelty is in the Fast Poisson Solver, which uses the known eigenvalues and eigenvectors of K and K2D. It is strange to solve linear equations $KU = F$ by expanding F and U in eigenvectors, but here it is extremely successful.

Elimination and Fill-in

For most two-dimensional problems, elimination is the way to go. The matrix from a partial differential equation is sparse (like K2D). It is banded but the bandwidth is not so small. (Meshpoints cannot be numbered so that all five neighbors in the molecule receive nearby numbers.) Figure 3.21 has points $1, \ldots, N$ along the first row, then a row at a time going up the square. *The neighbors above and below point j have numbers $j - N$ and $j + N$.*

Ordering by rows produces the -1's in $K2\mathrm{D}$ that are N places away from the diagonal. The matrix $K2\mathrm{D}$ has bandwidth N. **The key point is that elimination fills in the zeros inside the band.** We add row 1 (times $\frac{1}{4}$) to row 2, to eliminate the -1 in position $(2,1)$. But the last -1 in row 1 produces a new $-\frac{1}{4}$ in row 2. A zero inside the band has disappeared. As elimination continues from A to U, virtually the whole band is filled in.

In the end, U has about 5 times 25 nonzeros (this is N^3, the space needed to store U). There will be about N nonzeros next to the pivot when we reach a typical row, and N nonzeros below the pivot. Row operations to remove those nonzeros will require up to N^2 multiplications, and there are N^2 pivots. So the count of multiplications is about 25 times 25 (this is N^4, for elimination in 2D).

Figure 3.21: Typical rows of $K2\mathrm{D}$ have 5 nonzeros. Elimination fills in the band.

Section 6.1 will propose a different numbering of the meshpoints, to reduce the fill-in that we see in U. This reorders the rows of $K2\mathrm{D}$ by a permutation matrix P, and the columns by P^{T}. The new matrix $P(K2\mathrm{D})P^{\mathrm{T}}$ is still symmetric, but elimination (with fill-in) proceeds in a completely different order. The MATLAB command `symamd`(K) produces a nearly optimal choice of P.

Elimination is fast in two dimensions (but a Fast Poisson Solver is faster!). In three dimensions the matrix size is N^3 and the bandwidth is N^2. By numbering the nodes a plane at a time, vertical neighbors are N^2 nodes apart. The operation count for elimination becomes N^7, which can be seriously large. Chapter 6 on *Solving Large Systems* will introduce badly needed alternatives to elimination in 3D.

Solvers Using Eigenvalues

Our matrices K and $K2\mathrm{D}$ are extremely special. We know the eigenvalues and eigenvectors of the second-difference matrix K. The eigenvalues have the special form $\lambda = 2 - 2\cos\theta$, and the eigenvectors are discrete sines. There will be a similar pattern for $K2\mathrm{D}$, which is formed in a neat way from K (by Kronecker product). The Poisson Solver uses those eigenvalues and eigenvectors to solve $(K2\mathrm{D})(U2\mathrm{D}) = (F2\mathrm{D})$. On a square mesh it is much faster than elimination.

Here is the idea, first in one dimension. The matrix K has eigenvalues $\lambda_1, \ldots, \lambda_N$ and eigenvectors $y_1, \ldots, y_N$. There are three steps to the solution of $KU = F$:

1. Expand F as a combination $F = a_1 y_1 + \cdots + a_N y_N$ of the eigenvectors

2. Divide each a_k by λ_k

3. Recombine eigenvectors into $U = (a_1/\lambda_1)\, y_1 + \cdots + (a_N/\lambda_N)\, y_N$.

The success of the method depends on the speed of steps 1 and 3. Step 2 is fast.

To see that U in step 3 is correct, *multiply it by the matrix* K. Every eigenvector gives $Ky = \lambda y$. That cancels the λ in each denominator. The result is that KU agrees with the vector F in step 1.

Now look at the calculation required in each step, using matrices. Suppose S is the **eigenvector matrix**, with the eigenvectors $y_1, \ldots, y_N$ of K in its columns. Then the coefficients $a_1, \ldots, a_N$ come by solving $Sa = F$:

$$\textbf{Step 1} \quad \textbf{Solve } \boldsymbol{Sa = F} \qquad \begin{bmatrix} & & \\ y_1 & \cdots & y_N \\ & & \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_N \end{bmatrix} = a_1 y_1 + \cdots + a_N y_N = F\,. \qquad (4)$$

Thus $a = S^{-1}F$. Then step 2 divides the a's by the λ's to find $\Lambda^{-1}a = \Lambda^{-1}S^{-1}F$. (The **eigenvalue matrix** Λ is just the diagonal matrix of λ's.) Step 3 uses those coefficients a_k/λ_k in recombining the eigenvectors into the solution vector U:

$$\textbf{Step 3} \qquad U = \begin{bmatrix} & & \\ y_1 & \cdots & y_N \\ & & \end{bmatrix} \begin{bmatrix} a_1/\lambda_1 \\ \vdots \\ a_N/\lambda_N \end{bmatrix} = \boldsymbol{S\Lambda^{-1}a = S\Lambda^{-1}S^{-1}F}\,. \qquad (5)$$

You see the matrix $K^{-1} = S\Lambda^{-1}S^{-1}$ appearing in that last formula. We have $K^{-1}F$ because K itself is $S\Lambda S^{-1}$—the usual diagonalization of a matrix. The eigenvalue method is using this $S\Lambda S^{-1}$ factorization instead of $K = LU$ from elimination.

The speed of steps 1 and 3 depends on multiplying quickly by S^{-1} and S. Those are full matrices, not sparse like K. Normally they both need N^2 operations in one dimension (where the matrix size is N). But the "sine eigenvectors" in S give the Discrete Sine Transform, and the *Fast Fourier Transform* executes S and S^{-1} in $N \log_2 N$ steps. In one dimension this is not as fast as cN from elimination, but in two dimensions $N^2 \log(N^2)$ easily wins.

Column k of the matrix S contains the eigenvector y_k. The number $S_{jk} = \sin \frac{jk\pi}{N+1}$ is the jth component of that eigenvector. For our example with $N = 5$ and $N+1 = 6$, you could list the sines of every multiple of $\pi/6$. Here are those numbers:

$$\textbf{Sines } \frac{1}{2}, \frac{\sqrt{3}}{2}, 1, \frac{\sqrt{3}}{2}, \frac{1}{2}, 0, \ \textbf{(repeat with minus signs)} \ \textbf{(repeat 12 numbers forever)}.$$

The kth column of S (kth eigenvector y_k) takes every kth number from that list:

$$y_1 = \begin{bmatrix} 1/2 \\ \sqrt{3}/2 \\ 1 \\ \sqrt{3}/2 \\ 1/2 \end{bmatrix} \quad y_2 = \begin{bmatrix} \sqrt{3}/2 \\ \sqrt{3}/2 \\ 0 \\ -\sqrt{3}/2 \\ -\sqrt{3}/2 \end{bmatrix} \quad y_3 = \begin{bmatrix} 1 \\ 0 \\ -1 \\ 0 \\ 1 \end{bmatrix} \quad y_4 = \begin{bmatrix} \sqrt{3}/2 \\ -\sqrt{3}/2 \\ 0 \\ \sqrt{3}/2 \\ -\sqrt{3}/2 \end{bmatrix} \quad y_5 = \begin{bmatrix} 1/2 \\ -\sqrt{3}/2 \\ 1 \\ -\sqrt{3}/2 \\ 1/2 \end{bmatrix}$$

Those eigenvectors are orthogonal! This is guaranteed by the symmetry of K. All these eigenvectors have length $3 = (N+1)/2$. Dividing each column by $\sqrt{3}$, we have *orthonormal eigenvectors*. $S/\sqrt{3}$ is an orthogonal matrix Q, with $Q^{\mathrm{T}}Q = I$.

In this special case, S and Q are also symmetric. So $Q^{-1} = Q^{\mathrm{T}} = Q$.

Notice that y_k has $k-1$ changes of sign. It comes from k loops of the sine curve. The eigenvalues are increasing: $\lambda = 2-\sqrt{3},\ 2-1,\ 2-0,\ 2+1,\ 2+\sqrt{3}$. Those eigenvalues add to 10, which is the sum down the diagonal (the *trace*) of K_5. The product of the 5 eigenvalues (easiest by pairs) confirms that $\det(K_5) = 6$.

Fast Poisson Solvers

To extend this eigenvalue method to two dimensions, we need the eigenvalues and eigenvectors of $K2\mathrm{D}$. The key point is that the N^2 eigenvectors of $K2\mathrm{D}$ are *separable*. Each eigenvector y_{kl} separates into a product of sines:

Eigenvectors $\boldsymbol{y_{kl}}$ The (i,j) component of y_{kl} is $\sin\dfrac{ik\pi}{N+1}\sin\dfrac{jl\pi}{N+1}$. (6)

When you multiply that eigenvector by $K2\mathrm{D}$, you have second differences in the x-direction and y-direction. The second differences of the first sine (x-direction) produce a factor $\lambda_k = 2-2\cos\frac{k\pi}{N+1}$. This is the eigenvalue of K in 1D. The second differences of the other sine (y-direction) produce a factor $\lambda_l = 2-2\cos\frac{l\pi}{N+1}$. So the eigenvalue in two dimensions is the sum $\lambda_k+\lambda_l$ of one-dimensional eigenvalues:

$\boldsymbol{(K2\mathrm{D})y_{kl} = \lambda_{kl}y_{kl}}$ $\lambda_{kl} = (2-2\cos\frac{k\pi}{N+1}) + (2-2\cos\frac{l\pi}{N+1})$. (7)

Now the solution of $\boldsymbol{K2\mathrm{D}\,\mathrm{U} = \mathrm{F}}$ comes by a *two-dimensional sine transform*:

$$F_{i,j} = \sum\sum a_{kl}\sin\frac{ik\pi}{N+1}\sin\frac{jl\pi}{N+1} \qquad U_{i,j} = \sum\sum\frac{a_{kl}}{\lambda_{kl}}\sin\frac{ik\pi}{N+1}\sin\frac{jl\pi}{N+1} \tag{8}$$

Again we find the a's, divide by the λ's, and build U from the eigenvectors in S:

Step 1 $a = S^{-1}F$ **Step 2** $\Lambda^{-1}a = \Lambda^{-1}S^{-1}F$ **Step 3** $U = S\Lambda^{-1}S^{-1}F$

Swartztrauber [SIAM Review **19** (1977) 490] gives the operation count $2N^2\log_2 N$. This uses the Fast Sine Transform (based on the FFT) to multiply by S^{-1} and S. The Fast Fourier Transform is explained in Section 4.3.

Note We take this chance to notice the good properties of a Kronecker product $C = \texttt{kron}(A,B)$. Suppose A and B have their eigenvectors in the columns of S_A and S_B. The eigenvalues are in Λ_A and Λ_B. Then we know S_C and Λ_C:

The eigenvectors of $\texttt{kron}(\boldsymbol{A,B})$ ***are in*** $\texttt{kron}(\boldsymbol{S_A,S_B})$.
The eigenvalues are in $\texttt{kron}(\boldsymbol{\Lambda_A,\Lambda_B})$. (9)

The diagonal blocks in $\texttt{kron}(\Lambda_A,\Lambda_B)$ are entries $\lambda_k(A)$ times the diagonal matrix Λ_B. So the eigenvalues λ_{kl} of C are just the products $\lambda_k(A)\lambda_l(B)$.

In our case A and B were I and K. The matrix $K2\text{D}$ added the two products `kron`(I, K) and `kron`(K, I). Normally we cannot know the eigenvectors and eigenvalues of a matrix sum—except when the matrices commute. Since all our matrices are formed from K, *these Kronecker products do commute.* This gives the separable eigenvectors and eigenvalues in (6) and (7).

Cyclic Odd-Even Reduction

There is an entirely different (and very simple) approach to $\mathbf{KU} = \mathbf{F}$. I will start in one dimension, by writing down three rows of the usual second difference equation:

$$\begin{array}{lrcl} \text{Row } i-1 & -U_{i-2} + 2U_{i-1} - U_i & & = F_{i-1} \\ \text{Row } i & -U_{i-1} + 2U_i - U_{i+1} & & = F_i \\ \text{Row } i+1 & -U_i + 2U_{i+1} - U_{i+2} & & = F_{i+1} \end{array} \qquad (10)$$

Multiply the middle equation by 2, and add. *This eliminates U_{i-1} and U_{i+1}*:

Odd-even reduction in 1D $\quad -U_{i-2} + 2U_i - U_{i+2} = F_{i-1} + 2F_i + F_{i+1}\,. \qquad (11)$

Now we have a **half-size system**, involving only half of the U's (with even indices). The new system (11) has the same tridiagonal form as before. When we repeat, *cyclic reduction produces a quarter-size system.* Eventually we can reduce $KU = F$ to a very small problem, and then cyclic back-substitution produces the whole solution.

How does this look in two dimensions? The big matrix $K2\text{D}$ is block triangular:

$$\boldsymbol{K}\mathbf{2D} = \begin{bmatrix} A & -I & & \\ -I & A & -I & \\ & \cdot & \cdot & \cdot \\ & & -I & A \end{bmatrix} \quad \text{with} \quad A = K + 2I \quad \text{from equation (4)}\,. \qquad (12)$$

The three equations in (10) become *block equations for whole rows of N mesh values.* We are taking the unknowns $\mathbf{U}_i = (U_{i1}, \ldots, U_{iN})$ a row at a time. If we write three rows of (10), the block A replaces the number 2 in the scalar equation. The block $-I$ replaces the number -1. To reduce $(K2\text{D})(U2\text{D}) = (F2\text{D})$ to a half-size system, multiply the middle equation (with i even) by A and add the three block equations:

Reduction in 2D $\quad -I\mathbf{U}_{i-2} + (A^2 - 2I)\mathbf{U}_i - I\mathbf{U}_{i+2} = \mathbf{F}_{i-1} + A\mathbf{F}_i + \mathbf{F}_{i+1}\,. \ (13)$

The new half-size matrix is still block tridiagonal. *The diagonal blocks that were previously A in* (12) *are now* $A^2 - 2I$, with the same eigenvectors. The unknowns are the $\frac{1}{2}N^2$ values $U_{i,j}$ at meshpoints with even indices i.

The bad point is that each new diagonal block $A^2 - 2I$ has *five diagonals*, where the original block $A = K + 2I$ had three diagonals. This bad point gets worse as cyclic reduction continues. At step r, the diagonal blocks become $A_r = A_{r-1}^2 - 2I$ and their bandwidth doubles. We could find tridiagonal factors $(A - \sqrt{2}I)(A + \sqrt{2}I) = A^2 - 2I$, but the number of factors grows quickly. Storage and computation and roundoff error are increasing too rapidly with more reduction steps.

Stable variants of cyclic reduction were developed by Buneman and Hockney. The clear explanation by Buzbee, Golub, and Nielson [SIAM Journal of Numerical Analysis **7** (1970) 627] allows other boundary conditions and other separable equations, coming from polar coordinates or convection terms like $C\partial u/\partial x + D\partial u/\partial y$. After m steps of cyclic reduction, Hockney went back to a Fast Poisson Solver.

This combination FACR(m) is widely used, and the optimal number of cyclic reduction steps (before the FFT takes over) is small. For $N = 128$ a frequent choice is $m = 2$. Asymptotically m_{opt} grows like $\log\log N$ and the operation count for FACR(m_{opt}) is $3N^2 \log\log N$. In practical scientific computing with N^2 unknowns (or with N^3 unknowns in three dimensions), a Fast Poisson Solver is a winner.

******** Add code for Fast Poisson *********

Problem Set 3.5

Problems 1– are for readers who get enthusiastic about `kron`.

1 Why is the transpose of $C = \texttt{kron}(A, B)$ equal to $\texttt{kron}(A^{\mathrm{T}}, B^{\mathrm{T}})$? Why is the inverse equal to $C^{-1} = \texttt{kron}(A^{-1}, B^{-1})$? You have to transpose each block $a_{ij}B$ of the Kronecker product C, and then patiently multiply CC^{-1} by blocks.

C is symmetric (or orthogonal) when A and B are symmetric (or orthogonal).

2 Why is the matrix $C = \texttt{kron}(A, B)$ times the matrix $D = \texttt{kron}(S, T)$ equal to $CD = \texttt{kron}(AS, BT)$? This needs even more patience with block multiplication. The inverse $CC^{-1} = \texttt{kron}(I, I) = I$ is a special case.

Note Suppose S and T are eigenvector matrices for A and B. From $AS = S\Lambda_A$ and $BT = T\Lambda_B$ we have $CD = \texttt{kron}(AS, BT) = \texttt{kron}(S\Lambda_A, T\Lambda_B)$. Then $CD = D\,\texttt{kron}(\Lambda_A, \Lambda_B) = D\Lambda_C$. So $D = \texttt{kron}(S, T)$ is the eigenvector matrix for C.