Editor: Dianne P. O'Leary, oleary@cs.umd.edu

FAST SOLVERS AND SYLVESTER EQUATIONS: BOTH SIDES NOW

By Dianne P. O'Leary

IN OUR LAST HOMEWORK, WE SOLVED LARGE SPARSE SYSTEMS OF LINEAR EQUATIONS AND LEARNED THAT TO KEEP STORAGE AND COMPUTATIONAL COSTS LOW, IT'S VERY IMPORTANT TO exploit structure in our matrix. We exploited sparsity; in this homework, we'll consider a set of problems that have an additional type of structure.

The Poisson Equation

Recall the Poisson equation from our last homework:

$$-u_{xx} - u_{yy} = f(x, y),$$

with $(x, y) \in \Omega \subset \mathcal{R}^2$ and with appropriate boundary conditions specified. One strategy for solving it is to discretize the equation by choosing mesh points. We can then write an equation for each mesh point by approximating the derivatives u_{xx} and u_{yy} by finite differences, thus giving us a system of linear equations $A\mathbf{u} = \mathbf{f}$ to solve for estimates of each mesh point's value of u.

In one of our examples, Ω was a unit square with the mesh points chosen so they were equally spaced; we also had zero boundary conditions. For a 5×5 grid of mesh points, for example, we might order them as in Figure 1. If we let $x_j = jh$ and $y_k = kh$, with $h = 1/6 = 1/(n + 1)$, we can create two vectors $\mathbf{u} \approx [u(x_1, y_1), \ldots, u(x_n, y_1), u(x_1, y_2), \ldots, u(x_n, y_2), \ldots u(x_1, y_n), \ldots, u(x_n, y_n)]^T$ and $\mathbf{f} = [f(x_1, y_1), \ldots, f(x_n, y_1), f(x_1, y_2), \ldots, f(x_n, y_2), \ldots f(x_1, y_n), \ldots, f(x_n, y_n)]^T$, and write our approximation to $-u_{xx}$ at all mesh points as

$$A_x\mathbf{u} \equiv \frac{1}{h^2}\begin{bmatrix} \boldsymbol{T} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{T} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{T} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{T} \end{bmatrix}\mathbf{u}.$$

In this equation, the ith component of the 25×1 vector $\mathbf{u}$ is our approximation to u at the ith mesh point. The matrix $\mathbf{0}$ is a 5×5 matrix of zeros, and the matrix $\boldsymbol{T}$ is

$$\boldsymbol{T} = \begin{bmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{bmatrix}.$$

Similarly, our approximation to $-u_{yy}$ at all mesh points is

$$A_y\mathbf{u} \equiv \frac{1}{h^2}\begin{bmatrix} 2\boldsymbol{I} & -\boldsymbol{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ -\boldsymbol{I} & 2\boldsymbol{I} & -\boldsymbol{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & -\boldsymbol{I} & 2\boldsymbol{I} & -\boldsymbol{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & -\boldsymbol{I} & 2\boldsymbol{I} & -\boldsymbol{I} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & -\boldsymbol{I} & 2\boldsymbol{I} \end{bmatrix}\mathbf{u},$$

In this Homework

We're using the structure of a problem with n^2 unknowns to reduce the amount of computation from $O(n^6)$ (using the Cholesky decomposition) to $O(n^4)$ (exploiting sparsity) and then to $O(n^3)$ (using the Sylvester structure), a substantial savings when n is large. But knowing just a bit more about the problem's structure allows further reduction, down to $O(n^2 \log_2 n)$ when n is a power of two. Because we're computing n^2 answers, this is close to optimal, and it illustrates the value of exploiting every possible bit of structure in our problems.

where $\boldsymbol{I}$ is the identity matrix of dimension 5×5. So, we want to solve the linear system

$$(\boldsymbol{A}_x + \boldsymbol{A}_y)\mathbf{u} = \mathbf{f}. \quad (1)$$

This gives us a problem with a sparse matrix, and because our usual grids are $n \times n$, where n is much bigger than five, it's important to exploit the sparsity.

But in our particular problem—an equally spaced grid over a square (or a rectangle)—we have even more structure that we can exploit; Problem 1 shows how to write our problem more compactly.

PROBLEM 1.

Show that we can write Equation 1 as

$$(\boldsymbol{B}_y\boldsymbol{U} + \boldsymbol{U}\boldsymbol{B}_x) = \boldsymbol{F}, \quad (2)$$

where the matrix entry u_{jk} is our approximation to $u(x_j, y_k)$, $f_{jk} = f(x_j, y_k)$, and $\boldsymbol{B}_y = \boldsymbol{B}_x = (1/h^2)\boldsymbol{T}$.

The Sylvester Equation

Equation 2 is called a *Sylvester equation*. (It's also called a *Lyapunov equation* because $\boldsymbol{B}_y = \boldsymbol{B}_x^T$.) It looks daunting because the unknowns, $\boldsymbol{U}$, appear on both the left and right sides of other matrices, but as we just showed, the problem is equivalent to a system of linear equations. The Sylvester equation formulation gives us a compact representation of our problem in terms of the data matrix $\boldsymbol{F}$ and two tridiagonal matrices of dimension n. (In fact, because $\boldsymbol{B}_x = \boldsymbol{B}_y$, we really only have one tridiagonal matrix.)

Using the Schur Decomposition

One way to solve our problem efficiently is by using the *Schur decomposition* of a matrix. We'll consider this algorithm in the next problem.

PROBLEM 2.

a. Consider the Sylvester equation $\boldsymbol{L}\boldsymbol{U} + \boldsymbol{U}\boldsymbol{R} = \boldsymbol{C}$, where $\boldsymbol{L}$ is lower triangular and $\boldsymbol{R}$ is upper triangular. Show that we can easily determine the elements of the matrix $\boldsymbol{U}$ either row-by-row or column-by-column. How many arithmetic operations does this algorithm require?

b. By examining your algorithm, determine necessary and sufficient conditions on the main diagonal elements of $\boldsymbol{L}$ and $\boldsymbol{R}$ (that is, their eigenvalues) to ensure that a solution to the Sylvester equation exists.

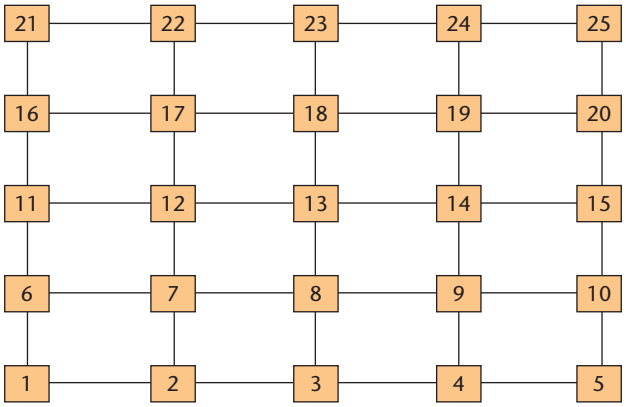

Figure 1. A 5×5 grid of mesh points.

c. Suppose we want to solve the Sylvester equation $\boldsymbol{A}\boldsymbol{U} + \boldsymbol{U}\boldsymbol{B} = \boldsymbol{C}$, where $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ of dimension $n \times n$ are given. ($\boldsymbol{A}$ and $\boldsymbol{B}$ are unrelated to the previously described matrices.) Let $\boldsymbol{A} = \boldsymbol{W}\boldsymbol{L}\boldsymbol{W}^*$ and $\boldsymbol{B} = \boldsymbol{Y}\boldsymbol{R}\boldsymbol{Y}^*$, where $\boldsymbol{W}$ and $\boldsymbol{Y}$ are unitary matrices with $\boldsymbol{W}\boldsymbol{W}^* = \boldsymbol{W}^*\boldsymbol{W} = \boldsymbol{I}$, $\boldsymbol{Y}\boldsymbol{Y}^* = \boldsymbol{Y}^*\boldsymbol{Y} = \boldsymbol{I}$; $\boldsymbol{L}$ is lower triangular; and $\boldsymbol{R}$ is upper triangular. (This is called a *Schur decomposition* of the two matrices.) Show that we can solve the Sylvester equation by applying the algorithm derived in part (a) to the equation $\boldsymbol{L}\hat{\boldsymbol{U}} + \hat{\boldsymbol{U}}\boldsymbol{R} = \hat{\boldsymbol{C}}$, where $\hat{\boldsymbol{U}} = \boldsymbol{W}^*\boldsymbol{U}\boldsymbol{Y}$ and $\hat{\boldsymbol{C}} = \boldsymbol{W}^*\boldsymbol{C}\boldsymbol{Y}$.

The reason for using the Schur decomposition in Problem 2 is that the most compact form we can achieve using a unitary matrix transformation is the triangular form. To preserve stability, we've considered only unitary transforms. One disadvantage of the Schur algorithm applied to real nonsymmetric matrices is that we must perform complex arithmetic, and the resulting computed matrix $\boldsymbol{U}$ could have a small imaginary part due to round-off error, even though the true matrix is guaranteed to be real.

Using the Eigendecomposition

We can solve Equation 2 by using the eigendecomposition instead of the Schur decomposition. This is less efficient for general matrices but more efficient when $\boldsymbol{B}_x$ and $\boldsymbol{B}_y$ have eigenvectors related to the discrete Fourier transform's vectors, as in our case, plus it's stable for symmetric matrices, because the eigenvector matrix is real orthogonal. In fact, the Schur decomposition reduces to the eigendecomposition when $\boldsymbol{A}$ and $\boldsymbol{B}$ are real symmetric: the matrices $\boldsymbol{L}$ and $\boldsymbol{R}$ are then also symmetric and thus diagonal. Let's see how we can solve our problem with an eigendecomposition.

Suppose that $\boldsymbol{B}_x$ and $\boldsymbol{B}_y$ are any two symmetric matrices that have the same eigenvectors, so

Tools

In Matlab, the functions `dst` and `idst` from the PDE Toolbox are useful for solving Problem 5. If the PDE Toolbox isn't available, you can manipulate the results of a fast Fourier transform to obtain the desired result. Also of use is `schur`, which has an option to return either an upper triangular (and possibly complex) factor or a real block upper triangular factor with 1×1 or 2×2 blocks on the main diagonal.

The method used in Problem 4 is extendable. There are fast solvers for solving 3D Poisson problems; for problems on rectangles, circles, and other simple domains; and for problems with different boundary conditions.[1]

The Schur algorithm for the Sylvester equation comes from Richard Bartels and G. W. Stewart.[2]

When we discuss number of operations, we consider the traditional algorithms for matrix product. Faster versions exist, but Webb Miller showed that the stability isn't as good.[3]

The Sylvester equation also arises in state space design in control theory[4] and in image processing.[5]

References

1. P.N. Swarztrauber and R.A. Sweet, "Algorithm 541: Efficient Fortran Subprograms for the Solution of Separable Elliptic Partial Differential Equations," *ACM Trans. Mathematical Software*, vol. 5, no. 3, 1979, pp. 352–364.
2. R. Bartels and G.W. Stewart, "Algorithm 432: The Solution of the Matrix Equation $AX - XB = C$," *Comm. ACM*, vol. 15, no. 9, 1972 pp. 820–826.
3. W. Miller, "Computational Complexity and Numerical Stability," *SIAM J. Computing*, vol. 4, no. 2, 1975, pp. 97–107.
4. B. Datta, *Numerical Methods for Linear Control Systems*, Elsevier, 2003.
5. P.C. Hansen, J.G. Nagy, and D.P. O'Leary, *Deblurring Images*, SIAM Press, 2006, ch. 1.

$$\boldsymbol{B}_x = \boldsymbol{V}\boldsymbol{\Lambda}_x\boldsymbol{V}^T \text{ and } \boldsymbol{B}_y = \boldsymbol{V}\boldsymbol{\Lambda}_y\boldsymbol{V}^T,$$

where the columns of $\boldsymbol{V}$ are the eigenvectors (normalized to length 1), and the entries of the diagonal matrices $\boldsymbol{\Lambda}_x$ and $\boldsymbol{\Lambda}_y$ are the eigenvalues. (Note that because $\boldsymbol{B}_x$ and $\boldsymbol{B}_y$ are symmetric, the columns of $\boldsymbol{V}$ are orthogonal, so $\boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{V}\boldsymbol{V}^T = \boldsymbol{I}$.) Substituting our eigendecompositions, Equation 2 becomes

$$\boldsymbol{V}\boldsymbol{\Lambda}_x\boldsymbol{V}^T\boldsymbol{U} + \boldsymbol{U}\boldsymbol{V}\boldsymbol{\Lambda}_y\boldsymbol{V}^T = \boldsymbol{F},$$

and multiplying this equation by $\boldsymbol{V}^T$ on the left and by $\boldsymbol{V}$ on the right, we get

$$\boldsymbol{\Lambda}_x\boldsymbol{V}^T\boldsymbol{U}\boldsymbol{V} + \boldsymbol{V}^T\boldsymbol{U}\boldsymbol{V}\boldsymbol{\Lambda}_y = \boldsymbol{V}^T\boldsymbol{F}\boldsymbol{V}.$$

Letting $\boldsymbol{Y} = \boldsymbol{V}^T\boldsymbol{U}\boldsymbol{V}$, we have an algorithm:

- Form the matrix $\hat{\boldsymbol{F}} = \boldsymbol{V}^T\boldsymbol{F}\boldsymbol{V}$.
- Solve the equation $\boldsymbol{\Lambda}_x\boldsymbol{Y} + \boldsymbol{Y}\boldsymbol{\Lambda}_y = \hat{\boldsymbol{F}}$, where $\boldsymbol{\Lambda}_x$ and $\boldsymbol{\Lambda}_y$ are diagonal.
- Form the matrix $\boldsymbol{U} = \boldsymbol{V}\boldsymbol{Y}\boldsymbol{V}^T$.

PROBLEM 3.

Determine a way to implement the second step of the algorithm using only $O(n^2)$ arithmetic operations.

Because we have n^2 entries of $\boldsymbol{U}$ to compute in solving our problem, the second step is optimal order, so the algorithm's efficiency depends on the implementation of the first and third steps. In general, each matrix–matrix product of $n \times n$ matrices takes $O(n^3)$ operations, so our complete algorithm would also take $O(n^3)$. In some special cases, though, we can compute the matrix products more quickly, which is true for our model problem, since there are formulas for the eigenvalues and eigenvectors of $\boldsymbol{B}_x$. We exploit this fact in Problem 4.

PROBLEM 4.

a. Denote the elements of the vector $\mathbf{v}_j$ by

$$v_{kj} = \alpha_j \sin\frac{kj\pi}{n+1},$$

where we choose α_j so that $\|\mathbf{v}_j\| = 1$. Show that $\boldsymbol{B}_x\mathbf{v}_j = \lambda_j\mathbf{v}_j$, where

$$\lambda_j = \frac{\left(2 - 2\cos\frac{j\pi}{n+1}\right)}{h^2} \text{ for } j = 1, 2, \ldots, n.$$

b. Show that we can multiply a vector by the matrix $\boldsymbol{V}$ or $\boldsymbol{V}^T$ via a discrete Fourier (sine) transform or inverse Fourier (sine) transform of length n. The discrete sine transform of a vector $\mathbf{x}$ is

$$y_k = \sum_{j=1}^{n} x_j\,(jk\pi / (n+1)).$$

We can accomplish this in $O(n \log_2 n)$ operations if n is a power of two and also in some larger (but still modest) number of operations if n is a composite number with many factors.

Using the multiplication algorithm from Problem 4, we can solve Equation 2 in $O(n^2 \log_2 n)$ operations when n is a

power of two, considerably less than the $O(n^3)$ operations generally required, or the $O(n^4)$ required for the sparse Cholesky decomposition applied to our original matrix problem. (The reordering strategies discussed in the last issue would reduce the factorization complexity somewhat, but wouldn't achieve $O(n^2 \log_2 n)$.)

PROBLEM 5.

Write a well-documented program to solve the discretization of the differential equation using Problem 2's Schur-based algorithm and the algorithm developed in Problem 4. (Debug the Schur-based algorithm using randomly generated real non-symmetric matrices.) Test your algorithms for $n = 2^p$, with $p = 2, \ldots, 9$, and choose the true solution matrix $\boldsymbol{U}$ randomly. Compare the results of the two algorithms with backslash for accuracy and time.

Acknowledgments

I'm grateful to Samuel Lamphier for helpful suggestions on this homework.

PARTIAL SOLUTION TO LAST ISSUE'S HOMEWORK ASSIGNMENT

SOLVING SPARSE LINEAR SYSTEMS: TAKING THE DIRECT APPROACH

By Dianne P. O'Leary

IN THIS PROBLEM, WE WERE TO GIVE ADVICE TO THE CEO OF POISSONISUS.COM ABOUT HOW TO SOLVE SPARSE SYSTEMS OF LINEAR EQUATIONS. WE FIRST TOOK A LINEAR SYSTEM

$$\boldsymbol{A}\mathbf{x} \equiv \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & 0 & 0 & 0 & 0 \\ \times & 0 & \times & 0 & 0 & 0 \\ \times & 0 & 0 & \times & 0 & 0 \\ \times & 0 & 0 & 0 & \times & 0 \\ \times & 0 & 0 & 0 & 0 & \times \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{bmatrix} \equiv \mathbf{b}$$

of size $n = 6$ and reordered it as

$$\begin{bmatrix} \times & 0 & 0 & 0 & 0 & \times \\ 0 & \times & 0 & 0 & 0 & \times \\ 0 & 0 & \times & 0 & 0 & \times \\ 0 & 0 & 0 & \times & 0 & \times \\ 0 & 0 & 0 & 0 & \times & \times \\ \times & \times & \times & \times & \times & \times \end{bmatrix} \begin{bmatrix} x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_1 \end{bmatrix} = \begin{bmatrix} b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_1 \end{bmatrix},$$

using the permutation matrix

$$\boldsymbol{P} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

PROBLEM 1.

a. Verify that the reordered system has the same solution as the original one and that when we use Gauss elimination (or the Cholesky factorization) on it, no new nonzeros are produced. (In particular, the Cholesky factor has $2n - 1$ nonzeros.)

b. Show that our reordered system is

$$\boldsymbol{PAP}^T(\boldsymbol{P}\mathbf{x}) = \boldsymbol{P}\mathbf{b}.$$

Answer:

For part (a), we notice that in Gauss elimination, we need only five row operations to zero elements in the lower triangle of the matrix, and the only row of the matrix that changes is the last row. Because this row has no zeros, no new nonzeros can be produced.

For part (b), because $\boldsymbol{P}^T\boldsymbol{P} = I$, we see that

$$\boldsymbol{A}\mathbf{x} = \mathbf{b} \Leftrightarrow \boldsymbol{PA}\mathbf{x} = \boldsymbol{P}\mathbf{b} \Leftrightarrow \boldsymbol{PAP}^T(\boldsymbol{P}\mathbf{x}) = \boldsymbol{P}\mathbf{b},$$

which verifies that the reordered system has the same solution as the original one.

PROBLEM 2.

a. A matrix $\boldsymbol{A}$ is a *band matrix* with bandwidth ℓ if $a_{jk} = 0$ whenever $|j - k| > \ell$. (An important special case is that of a *tridiagonal matrix*, with $\ell = 1$.) Show that the factor $\boldsymbol{L}$ for $\boldsymbol{A}$ also has bandwidth ℓ.

b. Define a matrix's *profile* to stretch from the first nonzero in each column to the main diagonal element in the column, and from the first nonzero in each row to the main diagonal element. For example, if

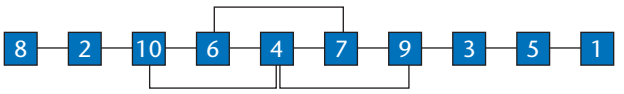

Figure A. Graph of the matrix in Problem 3.

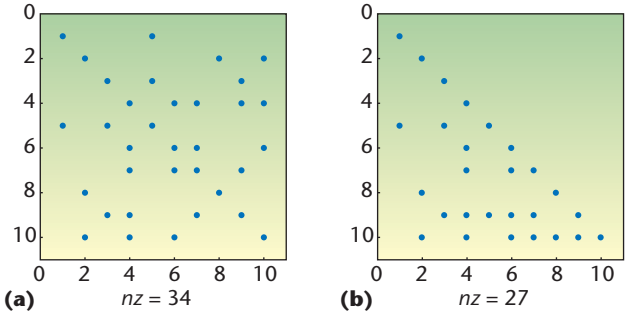

Figure B. Results of using the original ordering for (a) *S* and (b) the Cholesky factor of *S*.

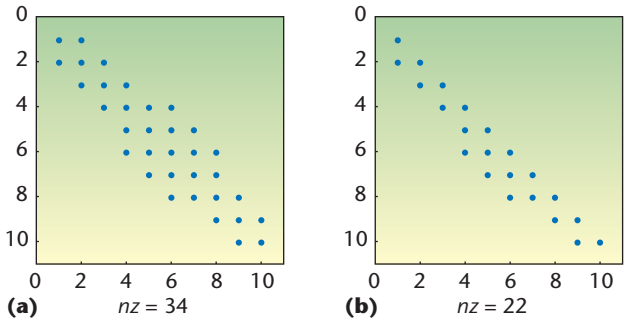

Figure C. Results of reordering using reverse Cuthill-McKee for (a) *S*(*r*, *r*) after Cuthill-McKee ordering and (b) chol(*S*(*r*, *r*)).

$$A = \begin{bmatrix} \times & 0 & \times & 0 & 0 & 0 \\ 0 & \times & 0 & 0 & \times & 0 \\ 0 & 0 & \times & 0 & \times & 0 \\ \times & 0 & 0 & \times & 0 & 0 \\ 0 & 0 & \times & 0 & \times & 0 \\ 0 & \times & 0 & 0 & 0 & \times \end{bmatrix},$$

then the profile of A contains its nonzeros as well as those zeros marked with ⊗:

$$\text{profile}(A) = \begin{bmatrix} \times & 0 & \times & 0 & 0 & 0 \\ 0 & \times & \otimes & 0 & \times & 0 \\ 0 & 0 & \times & 0 & \times & 0 \\ \times & \otimes & \otimes & \times & \otimes & 0 \\ 0 & 0 & \times & \otimes & \times & 0 \\ 0 & \times & \otimes & \otimes & \otimes & \times \end{bmatrix}.$$

Show that the factor L for a symmetric matrix A has no nonzeros outside the profile of A.

Answer:
For part (b), the important observation is that if element k is the first nonzero in row ℓ, then we start the elimination on row ℓ by a pivot operation with row k, after row k already has zeros in its first $k - 1$ positions. Therefore, an induction argument shows that no new nonzeros can be created before the first nonzero in a row. A similar argument works for the columns. Part (a) is a special case of this.

PROBLEM 3.

Draw the graph corresponding to the matrix

$$\begin{bmatrix} \times & 0 & 0 & 0 & \times & 0 & 0 & 0 & 0 & 0 \\ 0 & \times & 0 & 0 & 0 & 0 & 0 & \times & 0 & \times \\ 0 & 0 & \times & 0 & \times & 0 & 0 & 0 & \times & 0 \\ 0 & 0 & 0 & \times & 0 & \times & \times & 0 & \times & \times \\ \times & 0 & \times & \mathbf{0} & \times & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \times & 0 & \times & \times & 0 & 0 & \times \\ 0 & 0 & 0 & \times & 0 & \times & \times & 0 & \times & 0 \\ 0 & \times & 0 & 0 & 0 & 0 & 0 & \times & 0 & 0 \\ 0 & 0 & \times & \times & 0 & \mathbf{0} & \times & 0 & \times & 0 \\ 0 & \times & 0 & \times & 0 & \times & 0 & 0 & 0 & \times \end{bmatrix}.$$

Try each of the three reorderings on this matrix. Compare the sparsity of the Cholesky factors of the reordered matrices with the sparsity of the factor corresponding to the original ordering.

Answer:
Figure A shows the graph. The given matrix is a permutation of a band matrix with bandwidth 2; reverse Cuthill-McKee determined this and produced an optimal ordering. The reorderings and number of nonzeros in the Cholesky factor ($nz(L)$) are

Method	Ordering	$nz(L)$
Original	1 2 3 4 5 6 7 8 9 10	27
Reverse Cuthill-McKee	1 5 3 9 7 4 6 10 2 8	22
Minimum degree	2 8 10 6 1 3 5 9 4 7	24

Nested dissection (1 level)	8 2 10 6 4 9 3 5 1 7	25
Eigenpartition (1 level)	1 3 5 9 2 4 6 7 8 10	25

(Note that these answers aren't unique, due to tiebreaking.) Figures B through F show the resulting matrices and factors.

PROBLEM 4.

Use `slit2.m` and `laplace3d.m` (with $n = 15$) to generate three linear systems. Solve the linear systems using as many of these algorithms as possible:

- Cholesky on the original matrix.
- Cholesky using the reverse Cuthill-McKee ordering.
- Cholesky using the (approximate) minimum degree ordering.
- Cholesky using the nested dissection ordering.
- Cholesky using the eigenvector ordering.

Make a table reporting, for each method,

- time to solve the system (include reordering, factorization, and forward and back substitution);
- storage for the matrix factors; and
- the final relative residual $\|b - Ax_{computed}\|_2 / \|b\|_2$. (These should all be well below the errors due to discretization, so they won't be a factor in your recommendation.)

If possible, run larger problems, too.

Considering the 2D and 3D problems separately, report to the CEO of PoissonIsUs.com the performance of the various methods and your recommendation for what ordering to use.

Answer:

Using a double-precision word (two words, or 8 bytes) as the unit of storage and seconds as the unit of time, Tables 1 through 3 present these results.

All the algorithms produced solutions with small residual norm. On each problem, the approximate minimum degree algorithm gave factors requiring the lowest storage, preserved sparsity the best, and, on the last two problems, used the least time. (Note that local storage used within Matlab's `symrcm`, `symmmd`, `symamd`, and the toolbox `specnd` wasn't counted in this tabulation.) It's quite expensive to compute the eigenpartition ordering, so this method should only be used if the matrices will be used multiple times, to amortize cost. To complete this study, it would be important to try different values of n, to determine the storage and time's rate of increase as n increased.

To judge performance, several hardware parameters are significant, including computer (Sun Blade 1000 Model 1750), processor (Sun UltraSPARC-III), clock speed (750 MHz), and amount of RAM (1 Gbyte). The software specifications of importance include the operating system (Solaris 8) and the Matlab version (6.5.1). Benchmarking is a

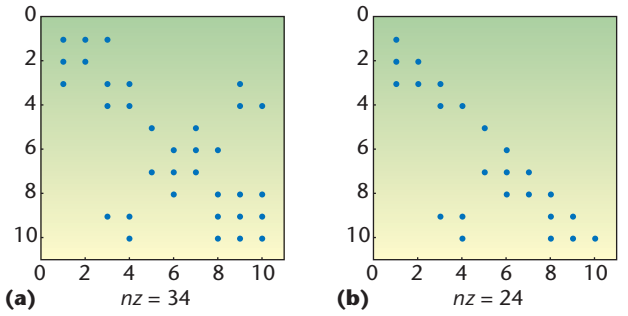

Figure D. Results of reordering using minimum degree. (a) $S(r, r)$ after minimum degree ordering and (b) chol($S(r, r)$).

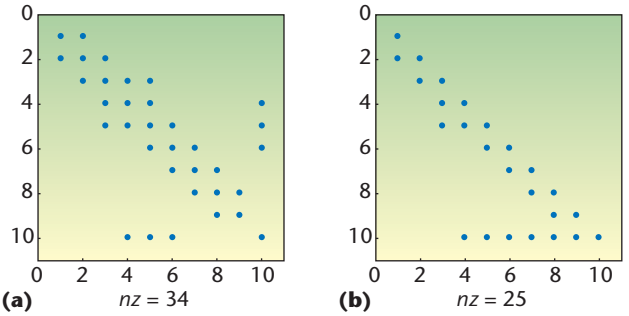

Figure E. Results of reordering using nested dissection. (a) $S(r, r)$ after nested dissection ordering and (b) chol($S(r, r)$).

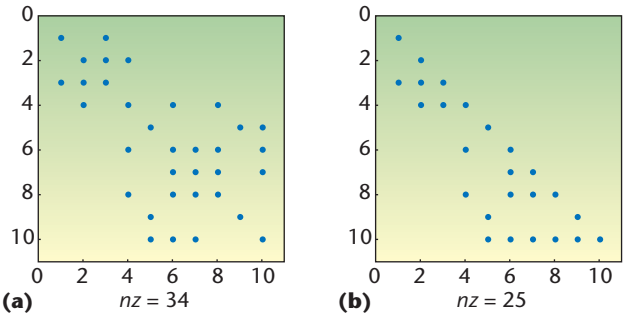

Figure F. Results of reordering using eigenpartitioning. (a) $S(r, r)$ after eigenpartition ordering and (b) chol($S(r, r)$).

Table 1. Solving Laplace equation on circle sector with $n = 1,208$.

Algorithm	Storage	Time	Residual_norm
Cholesky	660640	1.14e+00	4.04e-15
Cholesky, R-Cuthill-McKee	143575	7.21e-02	2.82e-15
Cholesky, minimum degree	92008	5.18e-02	1.96e-15
Cholesky, approximate mindeg	76912	1.70e-01	1.68e-15
Cholesky, eigenpartition	90232	4.59e+00	1.86e-15

Table 2. Solving Laplace equation on circle sector with $n = 4,931$.

Algorithm	Storage	Time	Residual_norm
Cholesky	6204481	3.21e+01	7.73e-15
Cholesky, R-Cuthill-McKee	1113694	7.08e-01	5.30e-15
Cholesky, minimum degree	486751	2.78e-01	2.85e-15
Cholesky, approximate mindeg	444109	2.34e-01	2.81e-15

(Too many recursions in eigenpartition method specnd from the Mesh Partitioning and Graph Separator Toolbox of Gilbert and Teng http://www.cerfacs.fr/algor/Softs/MESHPART/.)

Table 3. Solving Laplace equation on box, with $n = 15,625$.

Algorithm	Storage	Time	Residual_norm
Cholesky	28565072	1.02e+02	6.98e-14
Cholesky, R-Cuthill-McKee	16773590	3.79e+01	6.10e-14
Cholesky, minimum degree	8796896	4.08e+01	4.39e-14
Cholesky, approximate mindeg	7549652	3.08e+01	3.66e-14

(Too many recursions in eigenpartition method `specnd`*.)*

difficult task, depending on the choice of hardware, software, and test problems, and our results on this problem certainly raise more questions than they answer.

Dianne P. O'Leary is a professor of computer science and a faculty member in the Institute for Advanced Computer Studies and the Applied Mathematics Program at the University of Maryland. She has a BS in mathematics from Purdue University and a PhD in computer science from Stanford. O'Leary is a member of SIAM, the ACM, and AWM. Contact her at oleary@cs.umd.edu; www.cs.umd.edu/users/oleary/.