MATHEMATICS OF COMPUTATION, VOLUME 32, NUMBER 142
APRIL 1978, PAGES 441–446

Fast Poisson Solvers for Problems with Sparsity*

By Alexandra Banegas

Abstract. Fast Poisson solvers, which provide the numerical solution of Poisson's equation on regions that permit the separation of variables, have proven very useful in many applications. In certain of these applications the data is sparse and the solution is only required at relatively few mesh points. For such problems this paper develops algorithms that allow considerable savings in computer storage as well as execution speed. Results of numerical experiments are given.

1. Introduction In many applications there arises a need to solve the Poisson equation,

$$-\Delta u = f,$$

by finite difference methods, in regions where the method of separation of variables can be used. The region must, after a possible change of the independent variables, be rectangular and the boundary conditions must not change type along the different sides of the rectangle, see Widlund [13]. Fast, reliable solvers for such problems have been developed by Bank [1], Buneman [2], Buzbee, Golub and Nielson [3], Fischer, Golub, Hald, Leiva and Widlund [6], Hockney [7], [8], Swarztrauber and Sweet [12] and others.

In certain applications the data is sparse and the solution might also be required only at relatively few mesh points. We have such a case if the function f is zero. The only finite difference equations with nonzero right-hand side are then those which involve mesh points on the boundary. A similar situation arises in certain implementations of capacitance matrix methods where the data is nonzero only at two to four mesh points per mesh line, and similarly, the solution is needed only at relatively few points, see O'Leary and Widlund [9], Proskurowski [10], Proskurowski and Widlund [11] and Widlund [14].

The purpose of this paper is to develop methods which exploit this sparsity. In Sections 2 and 3, we will discuss problems in Cartesian coordinates for two and three dimensions, respectively. We will refer to the mesh points with nonzero data as source points and those where the solution is desired as target points. We will show that when the number of source and target points is relatively small, substantial savings can be realized both in terms of computer time and storage. Our method has been used extensively by Proskurowski [10] in his recent work on capacitance matrix methods.

Received June 2, 1977; revised August 8, 1977.
AMS (MOS) subject classifications (1970). Primary 65F05, 65N20.
*The work presented in this paper was supported by the ERDA Mathematics and Computing Laboratory, Courant Institute of Mathematical Sciences, New York University, under Contract EY-76-C-02-3077*000 with the Energy Research and Development Administration.

Results from numerical experiments are presented in Section 4.

Acknowledgement. The work presented in this paper was carried out while the author was a student at the Courant Institute. The project was suggested to her by Professor Olof Widlund.

2. Algorithms for Problems in Two Dimensions. We begin this section by describing a conventional fast Poisson solver based on the fast Fourier transform (FFT). We consider the standard five-point difference approximation of the Poisson equation on a rectangle using a uniform mesh. We further specialize our discussion to a case where the solution is periodic in one variable and Dirichlet conditions are imposed on two opposite horizontal sides of the rectangle.

The five-point approximation gives rise to a block tridiagonal system of linear equations,

$$\begin{pmatrix} A & -I & & & \\ -I & A & -I & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & \\ & & & -I & A \end{pmatrix} \begin{pmatrix} u^{(0)} \\ u^{(1)} \\ \vdots \\ \vdots \\ u^{(m-1)} \end{pmatrix} = \begin{pmatrix} f^{(0)} \\ f^{(1)} \\ \vdots \\ \vdots \\ f^{(m-1)} \end{pmatrix}.$$

The number of blocks, m, equals the number of horizontal mesh lines in the rectangle and the components of the vector $u^{(i)}$ and $f^{(i)}$ represent the values of the approximate solution and the data at the mesh points, respectively. The matrix A is $n \times n$,

$$A = \begin{pmatrix} 4 & -1 & & & -1 \\ -1 & 4 & -1 & & \\ & \ddots & \ddots & & \\ & & \ddots & \ddots & \\ -1 & & & -1 & 4 \end{pmatrix}.$$

Here n is the number of vertical mesh lines. We assume that n is an even number.

Since A is a symmetric matrix, there exists an orthonormal matrix of eigenvectors Q such that

$$Q^T A Q = D = \begin{pmatrix} \lambda_1 & & & & \\ & \lambda_2 & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & \lambda_n \end{pmatrix}.$$

Here the λ_i are the eigenvalues of A. The matrix A has two simple eigenvalues 2 and 6 which have the eigenvectors

$$(1/\sqrt{n})(1, 1, \ldots, 1)^T \quad \text{and} \quad (1/\sqrt{n})(1, -1, \ldots, -1)^T,$$

respectively. There are also $(n-2)/2$ double eigenvalues,

$$4 - 2\cos(2\pi l/n), \qquad l = 1, 2, \ldots, (n-2)/2,$$

with eigenvectors given by

$$\phi^{(l)}_{\mathrm{I},k} = \sqrt{2/n}\,\sin(2\pi kl/n),$$
$$\phi^{(l)}_{\mathrm{II},k} = \sqrt{2/n}\,\cos(2\pi kl/n),$$
$$k = 0, 1, \ldots, n-1, \quad l = 1, 2, \ldots, (n-2)/2.$$

The change of basis corresponding to the diagonalization of A can be carried out very efficiently by using the FFT provided n has many prime factors; see Cooley, Lewis and Welch [4]. We note that we can think of the Fourier coefficients of a vector $f^{(i)}$ in terms of inner products of this vector with the eigenvectors given above.

After this change of basis, the matrix transforms into

$$\begin{pmatrix} Q^T & & & \\ & Q^T & & \\ & & \ddots & \\ & & & Q^T \end{pmatrix} \begin{pmatrix} A & -I & & \\ -I & A & -I & \\ & \ddots & \ddots & \ddots \\ & & -I & A \end{pmatrix} \begin{pmatrix} Q & & & \\ & Q & & \\ & & \ddots & \\ & & & Q \end{pmatrix}$$
$$= \begin{pmatrix} D & -I & & \\ -I & D & -I & \\ & \ddots & \ddots & \ddots \\ & & -I & D \end{pmatrix},$$

A permutation of the rows and columns of this matrix, which preserves symmetry and groups the lth equation of each block together into one block, results in a matrix of the form

$$\begin{pmatrix} \Lambda_1 & & & \\ & \Lambda_2 & & \\ & & \ddots & \\ & & & \Lambda_n \end{pmatrix},$$

where

$$\Lambda_l = \begin{pmatrix} \lambda_l & -1 & & \\ -1 & \lambda_l & -1 & \\ & \ddots & \ddots & \ddots \\ & & -1 & \lambda_l \end{pmatrix}.$$

The algorithm is carried out in three steps:

(i) Apply the FFT to the $f^{(i)}$, $i = 0, \ldots, m-1$, i.e. to the appropriately partitioned data vector.

(ii) Permute the transformed vector and solve the n tridiagonal linear system of equations by Gaussian elimination or by some other means, see Dahlquist, Björck and Anderson [5, p. 166], Fischer, Golub, Hald, Leiva and Widlund [6] and Widlund [13].

(iii) Apply the inverse FFT, after permuting and partitioning.

We now turn to our variant of this algorithm. We use no two dimensional arrays. Since we wish to exploit the sparsity of the data, we choose to store them in terms of the coordinates of the source points and the values of f at these points. We note that we can find any Fourier coefficient at the expense of only a few arithmetic operations if the data differs from zero only at a few points on any horizontal mesh line. By using well-known properties of trigonometric functions, the components of the eigenvectors of A can be precomputed and stored using only on the order of n storage locations. The Fourier coefficients can be computed in an arbitrary order without any penalty, when using this method, and the total number of arithmetic operations required grows as $n \times N_S$, where N_S is the number of source points. To save work space all the data required for each tridiagonal system of equations is computed at one time. The system is then solved as before, and the contributions of the resulting Fourier coefficients to the solution at the prescribed target points are found and accumulated before the right-hand side of the next tridiagonal system is generated. The total cost of this inverse Fourier transform part grows as $n \times N_T$, where N_T is the number of target points. The coordinates of, and values at, the target points are stored in the same way as those of the source points. We note that the work to carry out what corresponds to the second part of the conventional algorithm remains unchanged.

The two methods discussed so far can be regarded as the extreme elements of a whole family of algorithms. Suppose that certain mesh lines have so many source or target points that it is worthwhile, for the sake of speed, to use the standard FFT for these lines. It is then easy to combine the two Fourier methods. In our program we have used an FFT routine by Dr. J. Cooley and ordered and renormalized the eigenvectors of A so that the two different Fourier methods can be used interchangeably. The designation of any mesh line as dense requires n storage locations but, if dense source and target lines coincide, the same storage locations can be used.

Our method can clearly be extended to other boundary conditions and coordinate systems which allow the use of a discrete Fourier transform to separate the variables; see Fischer, Golub, Hald, Leiva and Widlund [6] and Widlund [13] for a discussion of certain such problems.

3. Algorithms for Problems in Three Dimensions. We now consider Poisson's equation in three dimensions,

$$-\Delta u = f,$$

where u and f are functions of three variables x, y and z. We assume that these functions are periodic in x and y and impose Dirichlet conditions on the sides $z = 0$ and

$z = L$. A uniform mesh with $n_1 \times n_2 \times n_3$ mesh points is introduced and the Laplace operator is discretized by using a seven-point formula.

A conventional fast Poisson solver can be designed very much as in the two dimensional case. The FFT is first applied with respect to the x-variable and then with respect to the y-variable. The resulting array is permuted and provides data for $n_1 \times n_2$ tridiagonal systems of equations. The solutions of these systems give us the data for inverse Fourier transforms with respect to x and y. The algorithm requires const. $\times n_1 \times n_2 \times n_3 \times (\log_2 n_1 + \log_2 n_2 + O(1))$ arithmetic operations, where the constant is of moderate size if n_1 and n_2 have many prime factors.

We next consider ways of exploiting sparsity of the data. To simplify our notations, we assume that $n_1 = n_2 = n_3 = n$. If we have only a few source points on each mesh line, the number of such points, N_S, will be on the order of n^2. A direct extension of the algorithm of the previous section would require only on the order of n computer words of storage for the solution of the tridiagonal systems but on the order of n^4 arithmetic operations.

A better compromise can be found between storage and speed. Exploiting the sparsity of the data, a specific Fourier coefficient with respect to x is computed for all values of y and z. The resulting two dimensional array is dense and we use the FFT to find its Fourier transform with respect to y. We are then ready to solve n_2 tridiagonal systems of equations. The inverse FFT is used on the rows of the resulting two dimensional array, and we can then compute the contribution of these Fourier modes to the target points just as in the case of two dimensions. It is again clear that considerable savings in speed and storage can be realized if the number of source and target points is relatively small. As in the two dimensional case, certain rows parallel to the x-axis can be designated as dense source and target rows and the two Fourier transforms used interchangeably.

4. Numerical Experiments. A Fortran program was prepared for and run on the CDC 6600 at the ERDA Mathematics and Computing Laboratory of the Courant Institute of Mathematical Sciences, New York University. Our program, which solves problems in two dimensions, was later revised by Dr. Włodzimierz Proskurowski and used extensively by him on a CDC 7600 at the Lawrence Berkeley Laboratory, cf. Proskurowski [10]. The execution times given below refer to runs at Berkeley using a FTN 4 (OPT = 2) compiler.

Proskurowski's program requires $4(N_S + N_T) + 2(m + n)$ storage locations, where N_S and N_T denote the number of source and target points, respectively. If the sets of source and target points coincide, this requirement is decreased to $4N_S + 2(m + n)$. We note that a conventional fast Poisson solver requires at least $n \times m$ storage locations.

Operation counts given by Proskurowski [10] show that a conventional solver given in Fischer, Golub, Hald, Leiva and Widlund [6] should be about 11% slower if $N_S + N_T = 10n$ and $n = m = 64$. The advantage of our method grows slowly with increasing values of n and m. The actual execution times given below reflect these differences in the operation counts fairly closely. No experiment was carried out with

the conventional solver for the 256 × 256 case since such a run would have required the use of the extended core storage of the CDC 7600.

TABLE

CPU *Time in Seconds on a* CDC 7600 *Using a* FTN 4 (OPT = 2) *Compiler*

	Mesh	Time in Seconds	Number of Source and Target Points
Conventional Poisson Solver Using FFT	64 × 64	0.043	–
	128 × 128	0.203	–
Our Solver	64 × 64	0.027	392
	64 × 64	0.0385	600
	128 × 128	0.156	1224
	256 × 256	0.622	2472

Department of Mathematics
Universidad Nacional Autónoma de Honduras
Tegucigalpa, D. C., Honduras

1. R. E. BANK, *Marching Algorithms and Gaussian Elimination,* Proc. Sympos. on Sparse Matrix Computations, Argonne National Lab., Sept. 1975 (J. R. Bunch and D. J. Rose, Editors), Academic Press, New York, 1976.

2. O. BUNEMAN, *A Compact Noniterative Poisson Solver,* Rep. SUIPR-294, Inst. Plasma Research, Stanford Univ., 1969.

3. B. L. BUZBEE, G. H. GOLUB & C. W. NIELSON, "On direct methods for solving Poisson's equation," *SIAM J. Numer. Anal.,* v. 7, 1970, pp. 627–656.

4. J. W. COOLEY, P. A. W. LEWIS & P. D. WELCH, "The fast Fourier transform algorithm: Programming consideration in the calculation of sine, cosine and Laplace transform," *J. Sound Vib.,* v. 12, 1970, pp. 315–337.

5. G. DAHLQUIST, A. BJÖRCK & N. ANDERSON, *Numerical Methods,* Prentice-Hall, Englewood Cliffs, N. J., 1974.

6. D. FISCHER, G. GOLUB, O. HALD, C. LEIVA & O. WIDLUND, "On Fourier-Toeplitz methods for separable elliptic problems," *Math. Comp.,* v. 28, 1974, pp. 349–368.

7. R. W. HOCKNEY, "A fast direct solution of Poisson's equation using Fourier analysis," *J. Assoc. Comput. Mach.,* v. 12, 1965, pp. 95–113.

8. R. W. HOCKNEY, "The potential calculation and some applications," *Methods in Computational Physics,* Vol. 9, Academic Press, New York, 1970.

9. D. P. O'LEARY & O. WIDLUND, ERDA-NYU report. (To appear.)

10. W. PROSKUROWSKI, *Numerical Solution of Helmholtz's Equation by Implicit Capacitance Matrix Methods,* Report 6402, Lawrence Berkeley Laboratory, February 1977.

11. W. PROSKUROWSKI & O. WIDLUND, "On the numerical solution of Helmholtz's equation by the capacitance matrix method," *Math. Comp.,* v. 30, 1976, pp. 433–468. Appeared also as an ERDA-NYU report COO–3077–99.

12. P. SWARZTRAUBER & R. SWEET, *Efficient FORTRAN Subprograms for the Solution of Elliptic Partial Differential Equations,* Report NCAR-1N/1A-109, National Center for Atmospheric Research, Boulder, Colorado, 1975.

13. O. WIDLUND, "On the use of fast methods for separable finite difference equations for the solution of general elliptic problems," *Sparse Matrices and Their Applications* (D. J. Rose and R. A. Willoughby, Editors), Plenum Press, New York, 1972.

14. O. WIDLUND, *Capacitance Matrix Methods for Helmholtz' Equation on General Bounded Regions,* Proc. from a July 1976 Meeting in Oberwolfach. (To appear.)