JOURNAL OF COMPUTATIONAL PHYSICS 75, 123–137 (1988)

Fast Fourier Transforms for Direct Solution of Poisson's Equation with Staggered Boundary Conditions

ULRICH SCHUMANN

DFVLR, Institute of Atmospheric Physics, D-8031 Oberpfaffenhofen, West Germany

AND

ROLAND A. SWEET

Computational Mathematics Group, University of Colorado at Denver, Denver, Colorado, and Scientific Computing Division, National Bureau of Standards, Boulder, Colorado

Received March 13, 1987; revised May 19, 1987

This paper describes pre- and postprocessing algorithms used to incorporate the fast Fourier transform (FFT) into the solution of finite difference approximations to multi-dimensional Poisson's equation on a staggered grid where the boundary is located midway between two grid points. All frequently occurring boundary conditions (Neumann, Dirichlet, or cyclic) are considered including the combination of staggered Neumann boundary condition on one side with nonstaggered Dirichlet boundary condition on the other side. Experiences from implementing these algorithms in vectorized coding in Fortran subroutines are reported.

1. INTRODUCTION

In many applications of computational physics the solution of a sequence of multi-dimensional Poisson or Helmholtz equations is required. An example is Poisson's equation for the pressure in simulations of incompressible fluid flows which has to be solved every time-step. Since the solution time required for the Poisson equation can represent a significant portion of the total computation time, efficient algorithms are required for this purpose.

For finite difference approximations of fluid flow equations, staggered grids have been found to be well suited. In such a grid the finite difference approximation to Poisson's equation implies that the boundary is located midway between two adjacent grid points. Typically, either Dirichlet (specified solution), Neumann (specified normal derivative of the solution), or cyclic (periodic solution) boundary conditions are prescribed.

Two known fast, direct solution algorithms for Poisson's equation on staggered

0021-9991/88 $3.00

TABLE I

Availability of Pre- and Postprocessing for FFT of Transforms for Different Combinations of Boundary Conditions at $i=1$ and $i=n$

		At $i=n$			
		D	N	DS	NS
At $i=1$	D	7,6	5,6	—	*
	N	5,6	7,6	—	—
	DS	—	—	5,6	*
	NS	*	—	*	4,6

Note. In addition to [6, 7] for cyclic boundary conditions, from Ref. [4] Wilhelmson and Ericksen, [5] Swarztrauber (1977), [6] Swarztrauber (1986), [7] Cooley, Lewis, and Welch, [*] this paper, [—] not available.

grids are the Buneman variant of cyclic odd–even reduction [1–3] and the eigenfunction expansion method that employs the fast Fourier transform (FFT) [4, 5]. An asymptotic operation count for these two methods applied to three-dimensional problems [4] demonstrates the superiority of the latter method in higher dimensions.

The Fourier transform algorithm presented in [4] applied only to the Poisson equation with staggered Neumann boundary conditions. Corresponding Fourier transform algorithms for nonstaggered boundary conditions are given in [5–7]. These transforms employ pre- and postprocessing algorithms to convert the nonperiodic transforms into a periodic form which can be handled by available FFT software. "Symmetric FFTs" which avoid such pre- and postprocessings have been described by Swartztrauber [6]. They are available for transforms which arise for nonstaggered boundary conditions and either Neumann or Dirichlet boundary conditions on both sides of a staggered grid.

The purpose of this paper is to complete the set of pre- and postprocessing algorithms for efficient evaluation of the Fourier transforms required for boundary conditions on a staggered grid. Also, the often used combination of nonstaggered Dirichlet boundary condition on one side with staggered Neumann boundary condition on the other side is considered. Table I gives an overview on the availability of fast Fourier transform algorithms for various boundary condition combinations.

In Section 2, we state the precise problem, the eigenfunction expansion method and the related Fourier transforms for all common boundary conditions. In Section 3, we deduce pre- and postprocessing algorithms to incorporate fast Fourier transforms into the eigenfunction expansion method for those combinations of boundary conditions not treated previously. Section 4 describes software implementations of these algorithms and related experiences.

2. Solution of Poisson's Equation by Eigenfunction Expansion Method and Related Fourier Transforms

We assume that the discretized Poisson (or Helmholtz) equation can be written as

$$x_{i-1} - 2x_i + x_{i+1} - Ax_i = y_i, \qquad i = 1, 2, \ldots, n, \tag{1}$$

in which for multi-dimensional problems x_i and y_i are vectors and A a nonnegative definite matrix. Further we assume that either of the following boundary conditions (defined, e.g., at $i = 1$) applies:

Dirichlet, D: $x_0 = 0$,

Neumann, N: $x_2 - x_0 = 0$,

Dirichlet-staggered, DS: $x_1 + x_0 = 0$,

Neumann-staggered, NS: $x_1 - x_0 = 0$,

cyclic, C: $x_0 = x_n$, $x_{n+1} = x_1$.

Inhomogeneous boundary values can be incorporated into y_1 and y_n.

Poisson's equation (1) is to be solved by the eigenfunction expansion method [5]: Let ϕ_i^j be the ith component of the jth eigenvector and $\lambda_j \leqslant 0$ the corresponding eigenvalue satisfying

$$\phi_{i-1}^j - 2\phi_i^j + \phi_{i+1}^j = \lambda_j \phi_i^j, \tag{2}$$

$i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, n$, with the same homogeneous boundary conditions associated with Eq. (1). Then, solution of Eq. (1) can be obtained in three steps:

1. (*Analysis*) Determine $\bar{y}_j$, $j = 1, 2, \ldots, n$, from the inverse (which exists due to orthogonality of the ϕ_i^j) of

$$y_i = \sum_{j=1}^{n} \phi_i^j \bar{y}_j, \qquad i = 1, 2, \ldots, n. \tag{3}$$

2. Solve

$$(\lambda_j I - A)\, \bar{x}_j = \bar{y}_j, \qquad j = 1, 2, \ldots, n \tag{4}$$

for $\bar{x}_j$ (I is the unit matrix).

3. (*Synthesis*) Compute the solution x_i from

$$x_i = \sum_{j=1}^{n} \phi_i^j \bar{x}_j, \qquad i = 1, 2, \ldots, n. \tag{5}$$

If any eigenvalue, say λ_1, is zero and $\det(A) = 0$ then $\bar{y}_1$ has to satisfy a consistency

condition ($\bar{y}_1 = 0$ if $\bar{y}_1$ is a scalar) and an arbitrary additional equation has to be provided to determine $\bar{x}_1$ uniquely.

The eigenfunctions ϕ_i^j are composed of sine and cosine expressions. Hence Eq. (5) is basically a Fourier transform. The details of ϕ_i^j are given implicitly below.

The efficiency of this algorithm results from the ability to calculate the coefficients in steps 1 and 3 using the fast Fourier transform and thereby reducing an apparently n^2 calculation to order $n \log n$ operations.

Higher dimensional Poisson equations may be readily solved by the same algorithm with the simple modification that additional transforms in steps 1 and 3 must be made to separate the equations with respect to the additional dimensions.

For Fourier *synthesis* the "backward" transforms which satisfy the boundary conditions identically (including nonstaggered boundary conditions for completeness) are for $i = 1, 2, \ldots, n$:

$$\text{C–C:} \quad x_i = \frac{1}{2}\bar{x}_1 + \sum_{j=1}^{n/2-1}\left(\bar{x}_{2j}\cos\frac{2ij\pi}{n} + \bar{x}_{2j+1}\sin\frac{2ij\pi}{n}\right) + \frac{1}{2}\bar{x}_n(-1)^i, \tag{6a}$$

when n is even, and

$$x_i = \frac{1}{2}\bar{x}_1 + \sum_{j=1}^{(n-1)/2}\left(\bar{x}_{2j}\cos\frac{2ij\pi}{n} + \bar{x}_{2j+1}\sin\frac{2ij\pi}{n}\right), \tag{6b}$$

when n is odd; and for other boundary conditions:

$$\text{D–D:} \quad x_i = \sum_{j=1}^{n}\bar{x}_j \sin\frac{ij\pi}{n+1}; \tag{7}$$

$$\text{N–N:} \quad x_i = \frac{1}{2}\bar{x}_1 + \sum_{j=2}^{n-1}\bar{x}_j\cos\frac{(i-1)(j-1)\,\pi}{n-1} - \frac{1}{2}\bar{x}_n(-1)^i; \tag{8}$$

$$\text{D–N:} \quad x_i = \sum_{j=1}^{n}\bar{x}_j \sin\frac{i(2j-1)\,\pi}{2n}; \tag{9}$$

$$\text{N–D:} \quad x_i = \sum_{j=1}^{n}\bar{x}_j\cos\frac{(i-1)(2j-1)\,\pi}{2n}; \tag{10}$$

$$\text{DS–DS:} \quad x_i = \sum_{j=1}^{n}\bar{x}_j \sin\frac{(2i-1)\,j\pi}{2n}; \tag{11}$$

$$\text{NS–NS:} \quad x_i = \sum_{j=1}^{n}\bar{x}_j\cos\frac{(2i-1)(j-1)\,\pi}{2n}; \tag{12}$$

$$\text{DS–NS:} \quad x_i = \sum_{j=1}^{n}\bar{x}_j \sin\frac{(2i-1)(2j-1)\,\pi}{4n}; \tag{13}$$

$$\text{NS–DS:} \quad x_i = \sum_{j=1}^{n}\bar{x}_j\cos\frac{(2i-1)(2j-1)\,\pi}{4n}; \tag{14}$$

$$\text{D–NS:} \quad x_i = \sum_{j=1}^{n} \bar{x}_j \sin \frac{i(2j-1)\pi}{2n+1}; \tag{15}$$

$$\text{NS–D:} \quad x_i = \sum_{j=1}^{n} \bar{x}_j \cos \frac{(2i-1)(2j-1)\pi}{2n+1}. \tag{16}$$

The "forward" transforms for Fourier *analysis* are consequences of the orthogonality of the trigonometric functions:

$$\text{C–C:} \quad \bar{x}_1 = \frac{2}{n} \sum_{i=1}^{n} x_i, \tag{17a}$$

$$\bar{x}_{2j} = \frac{2}{n} \sum_{i=1}^{n} x_i \cos \frac{2ij\pi}{n}, \tag{17b}$$

$$\bar{x}_{2j+1} = \frac{2}{n} \sum_{i=1}^{n} x_i \sin \frac{2ij\pi}{n}, \qquad j = 1, 2, \ldots, m, \tag{17c}$$

where $m = n/2 - 1$ when n is even and $m = (n-1)/2$ when n is odd, and, when n is even

$$\bar{x}_n = \frac{2}{n} \sum_{i=1}^{n} x_i (-1)^i. \tag{17d}$$

For the other boundary conditions, Fourier analysis gives $\bar{x}_j$, $j = 1, 2, \ldots, n$ as follows:

$$\text{D–D:} \quad \bar{x}_j = \frac{2}{n+1} \sum_{i=1}^{n} x_i \sin \frac{ij\pi}{n+1}; \tag{18}$$

$$\text{N–N:} \quad \bar{x}_j = \frac{1}{n-1} x_1 + \frac{2}{n-1} \times \sum_{i=2}^{n-1} x_i \cos \frac{(i-1)(j-1)\pi}{n-1} - \frac{1}{n-1} x_n (-1)^j; \tag{19}$$

$$\text{D–N:} \quad \bar{x}_j = \frac{1}{n} (-1)^{j+1} x_n + \frac{2}{n} \sum_{i=1}^{n-1} x_i \sin \frac{i(2j-1)\pi}{2n}; \tag{20}$$

$$\text{N–D:} \quad \bar{x}_j = \frac{1}{n} x_1 + \frac{2}{n} \sum_{i=1}^{n-1} x_i \cos \frac{(i-1)(2j-1)\pi}{2n}; \tag{21}$$

$$\text{DS–DS:} \quad \bar{x}_j = \frac{2}{n} \sum_{i=1}^{n} x_i \sin \frac{(2i-1)j\pi}{2n}; \tag{22}$$

$$\text{NS–NS:}\quad \bar{x}_j = \frac{1}{d_j}\sum_{i=1}^{n} x_i \cos\frac{(2i-1)(j-1)\,\pi}{2n}, \tag{23}$$

$$d_j = n \quad \text{for} \quad j = 1,$$

$$= \frac{n}{2} \quad \text{for} \quad j > 1;$$

$$\text{DS–NS:}\quad \bar{x}_j = \frac{2}{n}\sum_{i=1}^{n} x_i \sin\frac{(2i-1)(2j-1)\,\pi}{4n}; \tag{24}$$

$$\text{NS–DS:}\quad \bar{x}_j = \frac{2}{n}\sum_{i=1}^{n} x_i \cos\frac{(2i-1)(2j-1)\,\pi}{4n}; \tag{25}$$

$$\text{D–NS:}\quad \bar{x}_j = \frac{4}{2n+1}\sum_{i=1}^{n} x_i \sin\frac{i(2j-1)\,\pi}{2n+1}; \tag{26}$$

$$\text{NS–D:}\quad \bar{x}_j = \frac{4}{2n+1}\sum_{i=1}^{n} x_i \cos\frac{(2i-1)(2j-1)\,\pi}{2n+1}. \tag{27}$$

The eigenvalues λ_j required for Eq. (4) are

$$\text{C–C:}\quad \lambda_1 = 0 \tag{28a}$$

$$\lambda_{2j} = \lambda_{2j+1} = -4\sin^2\frac{j\pi}{n}, \tag{28b}$$

for $j = 1, 2, \ldots, n/2 - 1$ when n is even, $j = 1, 2, \ldots, (n-1)/2$ when n is odd, and, when n is even,

$$\lambda_n = -4: \tag{28c}$$

$$\text{D–D:}\quad \lambda_j = -4\sin^2\frac{j\pi}{2(n+1)}, \tag{29}$$

$$\text{N–N:}\quad \lambda_j = -4\sin^2\frac{(j-1)\,\pi}{2(n-1)}, \tag{30}$$

$$\text{D–N or N–D:}\quad \lambda_j = -4\sin^2\frac{(2j-1)\,\pi}{4n}, \tag{31}$$

$$\text{DS–DS:}\quad \lambda_j = -4\sin^2\frac{j\pi}{2n}, \tag{32}$$

$$\text{NS–NS:}\quad \lambda_j = -4\sin^2\frac{(j-1)\,\pi}{2n}, \tag{33}$$

$$\text{DS–NS or NS–DS:} \quad \lambda_j = -4 \sin^2 \frac{(2j-1)\pi}{4n}. \tag{34}$$

$$\text{D–NS or NS–D:} \quad \lambda_j = -4 \sin^2 \frac{(2j-1)\pi}{2(2n+1)}, \qquad j = 1, 2, ..., n. \tag{35}$$

Note that $\lambda_1 = 0$ for cases C–C, N–N, and NS–NS. Also note that the transform pairs for cases DS–DS and D–N are identical except for scaling.

For efficient evaluation of the real periodic transform (C–C) fast Fourier transform subroutines are available in practically all computer libraries. We make use of the subroutine package FFTPAK developed by Swarztrauber on the basis of the autosort algorithm credited to Stockham [8]. This package contains routines for cases C–C, D–D, N–N, D–N, and N–D. There are no restrictions on the number of data n; however, the usual considerations apply, namely, that all the routines are more efficient when n is a product of many small prime numbers.

3. Pre- and Postprocessing Algorithms for Fourier Transforms Associated with Staggered Boundary Conditions

The transform pair, Eqs. (6) and (17), for real periodic data can be evaluated efficiently by existing FFT algorithms. Hence, if one expresses the other transforms in terms of the real periodic transform pair then they can be evaluated efficiently employing the same FFT algorithms. The generic expression implies the necessary pre- and postprocessing of the data before and after calling the FFT subroutine. The pre- and postprocessing steps for Fourier analysis are the inverses of those required for Fourier synthesis. For symmetric transform pairs (as for DS–NS) the pre- and postprocessing algorithms for analysis and for synthesis are identical except for scaling.

In Section 3.1 we deduce the generic expression and the resultant pre- and postprocessing algorithms for case DS–NS and in Section 3.2 likewise for case D–NS. For all other cases, the algorithms are either known from previous publications, see Table I, or can be adapted to such existing algorithms. These adaptions are discussed in Section 3.3.

3.1. *Case* DS–NS, *Dirichlet–Neumann Boundary Conditions on a Staggered Grid*

Subsequently, the algorithm is deduced for fast evaluation of synthesis, Eq. (13), first for even n: Starting from

$$x_i = \sum_{j=0}^{n/2-1} \bar{x}_{2j+1} \sin \frac{(2i-1)(4j+1)\pi}{4n} + \sum_{j=1}^{n/2} \bar{x}_{2j} \sin \frac{(2i-1)(4j-1)\pi}{4n}.$$

after replacing

$$\sin\frac{(2i-1)(4j\pm 1)\,\pi}{4n}=\sin\frac{(2i-1)\,j\pi}{n}\cos\frac{(2i-1)\,\pi}{4n}\pm\cos\frac{(2i-1)\,j\pi}{n}\sin\frac{(2i-1)\,\pi}{4n}$$

we form

$$a_i\equiv x_i\left[\sin\frac{(2i-1)\,\pi}{4n}+\cos\frac{(2i-1)\,\pi}{4n}\right]$$
$$+x_{n+1-i}\left[\cos\frac{(2i-1)\,\pi}{4n}-\sin\frac{(2i-1)\,\pi}{4n}\right] \tag{36}$$

and obtain

$$a_i=\bar{x}_1+\sum_{j=1}^{n/2-1}\left[(\bar{x}_{2j}+\bar{x}_{2j+1})\sin\frac{(2i-1)\,j\pi}{n}\right.$$
$$\left.+(\bar{x}_{2j+1}-\bar{x}_{2j})\cos\frac{(2i-1)\,j\pi}{n}\right]-\bar{x}_n(-1)^i. \tag{37}$$

This can be rewritten as

$$a_i=\frac{1}{2}\bar{a}_1+\sum_{j=1}^{n/2-1}\left[\bar{a}_{2j}\cos\frac{2ij\pi}{n}+\bar{a}_{2j+1}\sin\frac{2ij\pi}{n}\right]+\frac{1}{2}\bar{a}_n(-1)^i, \tag{38}$$

where

$$\bar{a}_1=2\bar{x}_1, \qquad \bar{a}_n=-2\bar{x}_n. \tag{39a}$$

$$\bar{a}_{2j}=(\bar{x}_{2j+1}-\bar{x}_{2j})\cos\frac{j\pi}{n}-(\bar{x}_{2j}+\bar{x}_{2j+1})\sin\frac{j\pi}{n}, \tag{39b}$$

$$\bar{a}_{2j+1}=(\bar{x}_{2j}+\bar{x}_{2j+1})\cos\frac{j\pi}{n}+(\bar{x}_{2j+1}-\bar{x}_{2j})\sin\frac{j\pi}{n},$$

$$j=1, 2, ..., \frac{n}{2}-1. \tag{39c}$$

Hence, if we preprocess $\bar{x}_j$ by computing $\bar{a}_j$ from Eq. (39), then the real periodic transform Eq. (38) can be used to compute a_i. But Eq. (36) can be rewritten as

$$x_i=\frac{1}{2}a_i\left[\sin\frac{(2i-1)\,\pi}{4n}+\cos\frac{(2i-1)\,\pi}{4n}\right]$$
$$-\frac{1}{2}a_{n+1-i}\left[\cos\frac{(2i-1)\,\pi}{4n}-\sin\frac{(2i-1)\,\pi}{4n}\right], \qquad i=1, 2, ..., n. \tag{40}$$

Thus, once the a_i are determined from Eq. (38), they can be postprocessed by Eq. (40) to compute the x_i.

For odd n, the generic expression, replacing Eq. (37), is

$$a_i = \bar{x}_1 + \sum_{j=1}^{(n-1)/2} \left[(\bar{x}_{2j} + \bar{x}_{2j+1}) \sin \frac{(2i-1)\, j\pi}{n} + (\bar{x}_{2j+1} - \bar{x}_{2j}) \cos \frac{(2i-1)\, j\pi}{n} \right], \tag{41}$$

which can be rewritten as

$$a_i = \frac{1}{2} \bar{a}_1 + \sum_{j=1}^{(n-1)/2} \left[\bar{a}_{2j} \cos \frac{2ij\pi}{n} + \bar{a}_{2j+1} \sin \frac{2ij\pi}{n} \right], \tag{42}$$

where $\bar{a}_1$ is as defined in Eq. (39a), and the $\bar{a}_{2j}$, $\bar{a}_{2j+1}$ are as given in Eqs. (39b), (39c), for $j = 1, 2, ..., (n-1)/2$. From these relations the pre- and postprocessing steps are easily deduced.

Note that only a real transform of length n is required. The same algorithm is used to perform Fourier analysis by computing $\bar{x}_j$ from given values of $(2/n)\, x_i$.

3.2. *Case* D–NS, *Mixed Nonstaggered Dirichlet with Staggered Neumann Boundary Conditions*

The general concept of the previous and similar pre- and postprocessing algorithms is to halve the number of data to be transformed by making use of the symmetry properties of the data. The transforms for D–NS boundary conditions, Eqs. (15, 26), is intrinsically related to a sine transform for an odd number $(2n+1)$ of data which excludes the possibility of halving the number of data. Therefore, we embed two sequences x_i and y_i of length n into one vector a_i of length $2n+1$ such that the transforms can be reduced to that for real periodic data. Thus, the following algorithm requires that transforms are to be evaluated for pairs of data vectors.

Equation (15) for two transforms

$$x_i = \sum_{j=1}^{n} \bar{x}_j \sin \frac{i(2j-1)\,\pi}{2n+1},$$

$$y_i = \sum_{j=1}^{n} \bar{y}_j \sin \frac{i(2j-1)\,\pi}{2n+1}, \qquad i = 1, 2, ..., n,$$

can be rewritten by combining the two vectors into one vector a_i of length $2n+1$:

$$a_i = x_i \sin \frac{i\pi}{2n+1} + y_i \cos \frac{i\pi}{2n+1} \tag{43}$$

$$= \sum_{j=1}^{n} \bar{x}_j \sin \frac{i(2j-1)\,\pi}{2n+1} \sin \frac{i\pi}{2n+1} + \bar{y}_j \sin \frac{i(2j-1)\,\pi}{2n+1} \cos \frac{i\pi}{2n+1}. \tag{44}$$

Using $\sin\alpha\sin\beta=[\cos(\alpha-\beta)-\cos(\alpha+\beta)]/2$ and $\sin\alpha\cos\beta=[\sin(\alpha-\beta)+\sin(\alpha+\beta)]/2$, we obtain

$$\begin{aligned} a_i &= \sum_{j=1}^{n} \bar{x}_j\left[\cos\frac{2i(j-1)\,\pi}{2n+1}-\cos\frac{2ij\pi}{2n+1}\right] \\ &\quad + \bar{y}_j\left[\sin\frac{2i(j-1)\,\pi}{2n+1}+\sin\frac{2ij\pi}{2n+1}\right] \\ &= \bar{x}_1+\sum_{j=1}^{n-1}\left[(\bar{x}_{j+1}-\bar{x}_j)\cos\frac{2ij\pi}{2n+1}+(\bar{y}_j+\bar{y}_{j+1})\sin\frac{2ij\pi}{2n+1}\right] \\ &\quad -\bar{x}_n\cos\frac{2in\pi}{2n+1}+\bar{y}_n\sin\frac{2in\pi}{2n+1}. \end{aligned} \tag{45}$$

This takes the form of the standard transform for an odd number $(2n+1)$ of real periodic data,

$$a_i=\frac{1}{2}\bar{a}_1+\sum_{j=1}^{n}\left(\bar{a}_{2j}\cos\frac{2ij\pi}{2n+1}+\bar{a}_{2j+1}\sin\frac{2ij\pi}{2n+1}\right), \qquad i=1, 2, ..., 2n+1, \tag{46}$$

if

$$\begin{aligned} &\bar{a}_1=2\bar{x}_1, && \bar{a}_{2n}=-\bar{x}_n, && \bar{a}_{2n+1}=\bar{y}_n, \\ &\bar{a}_{2j}=\bar{x}_{j+1}-\bar{x}_j, && \bar{a}_{2j+1}=\bar{y}_j+\bar{y}_{j+1}, && j=1, 2, ..., n-1. \end{aligned} \tag{47}$$

Hence, for synthesis we preprocess $\bar{x}_j$ and $\bar{y}_j$ by computing $\bar{a}_j$ from Eq. (47), then the real periodic synthesis Eq. (46) can be evaluated using a FFT library routine to obtain a_i. Equation (43) can be inverted to determine the x_i, y_i by postprocessing:

$$x_i=(a_i+a_{2n+1-i})\Big/\left[2\sin\frac{i\pi}{2n+1}\right], \tag{48a}$$

$$y_i=(a_i-a_{2n+1-i})\Big/\left[2\cos\frac{i\pi}{2n+1}\right], \qquad i=1, 2, ..., n. \tag{48b}$$

This completes the Fourier synthesis, Eq. (15).

For Fourier analysis, Eq. (26), the above algorithm is to be inverted. Thus, we start by preprocessing

$$a_i=x_i\sin\frac{i\pi}{2n+1}+y_i\cos\frac{i\pi}{2n+1}, \tag{49a}$$

$$a_{2n+1-i}=x_i\sin\frac{i\pi}{2n+1}-y_i\cos\frac{i\pi}{2n+1}, \qquad i=1, 2, ..., n, \tag{49b}$$

$$a_{2n+1}=0. \tag{49c}$$

The resultant a_i are supplied to a FFT routine for real periodic analysis to determine

$$\bar{a}_1 = \frac{2}{2n+1} \sum_{i=1}^{2n+1} a_i, \tag{50a}$$

$$\bar{a}_{2j} = \frac{2}{2n+1} \sum_{i=1}^{2n+1} a_i \cos \frac{2ij\pi}{2n+1}, \tag{50b}$$

$$\bar{a}_{2j+1} = \frac{2}{2n+1} \sum_{i=1}^{2n+1} a_i \sin \frac{2ij\pi}{2n+1}, \qquad j = 1, 2, ..., n. \tag{50c}$$

Finally, postprocessing gives the required modes,

$$\bar{x}_1 = \frac{\bar{a}_1}{2}, \qquad \bar{x}_{j+1} = \bar{x}_j + \bar{a}_{2j}, \qquad j = 1, 2, ..., n-1, \tag{51a}$$

$$\bar{y}_n = \bar{a}_{2n+1}, \qquad \bar{y}_j = \bar{a}_{2j+1} - \bar{y}_{j+1}, \qquad j = n-1, n-2, ..., 1. \tag{51b}$$

This completes Fourier analysis for D–NS boundary conditions.

Note that a real periodic transform of length $2n+1$ is required. Hence, it is essential for this algorithm to have a library routine which allows us to compute such transforms for an odd number of data. In order to be efficient, $(2n+1)$ must be highly composite. For example, $(2n+1)$ factors into primes 3 and 5 if $n \in \{1, 2, 4, 7, 12, 13, 22, 37, 40, 62, 67, 112, ...\}$. Thus, the requirements on n for efficient transforms are more stringent in this case than for other transforms.

3.3. *Other Boundary Conditions*

As has been shown above, the pre- and postprocessing algorithms are simple to deduce once the given transform is expressed in the form of the real periodic transform. Thus for the other transforms it suffices to give these expressions. For DS–DS, Eq. (22) can be rewritten as

$$\bar{a}_j \equiv \frac{n}{2}\left[\bar{x}_j\left(\sin\frac{j\pi}{2n} + \cos\frac{j\pi}{2n}\right) + \bar{x}_{n-j}\left(\cos\frac{j\pi}{2n} - \sin\frac{j\pi}{2n}\right)\right], \tag{52}$$

$$\bar{a}_j = x_1 + \sum_{i=1}^{n/2-1}\left[(x_{2i+1} - x_{2i})\cos\frac{2ij\pi}{n} + (x_{2i+1} + x_{2i})\sin\frac{2ij\pi}{n}\right], \\ - x_n(-1)^j, \qquad j = 1, 2, ..., n \ (\bar{x}_0 \equiv 0) \tag{53}$$

if n is even, and

$$\bar{a}_j = x_1 + \sum_{i=1}^{(n-1)/2}\left[(x_{2i+1} - x_{2i})\cos\frac{2ij\pi}{n} + (x_{2i+1} + x_{2i})\sin\frac{2ij\pi}{n}\right] \tag{54}$$

if n is odd.

For inversion of Eq. (52) we note

$$\bar{x}_j = \frac{1}{n}\left[\left(\sin\frac{j\pi}{2n} + \cos\frac{j\pi}{2n}\right)\bar{a}_j + \left(\sin\frac{j\pi}{2n} - \cos\frac{j\pi}{2n}\right)\bar{a}_{n-j}\right]. \tag{55}$$

The resultant pre- and postprocessing algorithms are identical (up to scaling) to the algorithm for the efficient calculation of sine quarter-wave transforms developed by Swarztrauber [5] for D–N boundary conditions. A symmetric FFT for this case which avoids pre- and postprocessing is described in [6].

For NS–NS, Eq. (23) can be rewritten as

$$\begin{aligned}\bar{a}_{j+1} \equiv d_{j+1}\,\bar{x}_{j+1}&\left(\cos\frac{j\pi}{2n} + \sin\frac{j\pi}{2n}\right)\\ &+ d_{n+1-j}\,\bar{x}_{n+1-j}\left(\sin\frac{j\pi}{2n} - \cos\frac{j\pi}{2n}\right)\end{aligned} \tag{56}$$

$$\begin{aligned}\bar{a}_{j+1} = x_1 + \sum_{i=1}^{n/2-1}&\left[(x_{2i} + x_{2i+1})\cos\frac{2ij\pi}{n} + (x_{2i} - x_{2i+1})\sin\frac{2ij\pi}{n}\right]\\ &+ x_n(-1)^j, \qquad j = 0, 1, \ldots, n-1.\end{aligned} \tag{57}$$

so that

$$\begin{aligned}\bar{x}_{j+1} = \Bigg[&\left(\sin\frac{j\pi}{2n} + \cos\frac{j\pi}{2n}\right)\bar{a}_{j+1}\\ &+ \left(\cos\frac{j\pi}{2n} - \sin\frac{j\pi}{2n}\right)\bar{a}_{n+1-j}\Bigg] \Big/ (2d_{j+1}),\end{aligned} \tag{58}$$

from which we obtain the pre- and postprocessing algorithms as given by Wilhelmson and Ericksen [4]. (In [4], however, the normalization d_j is deleted from Eq. (23) and included in Eq. (12) instead.) Again, a symmetric FFT for this case is described in [6].

For NS–DS and NS–D we obtain the algorithms as for DS–NS and D–NS due to the mirror symmetry of these by replacing x_i, $\bar{x}_j$ with x_{n+1-i}, $(-1)^{j+1}\bar{x}_j$, respectively.

4. Software and Performance

The Fourier transforms described in the previous section have been independently implemented by the two authors in two sets of Fortran subroutines TRANX/Y (by U.S.) and VSFFT (by R.S.). The routines perform the transforms on three-dimensional arrays $x_i \equiv x_{i,j,k}$. The most inner loops are those over k and are vectorizable, therefore. To maintain the vectorization, a portion of the scalar FFT

package FFTPAK [8] has been recoded (by R.S.) into a vectorizable real periodic FFT package VRFFTPK. It uses a pre- and postprocessing algorithm to convert the real transforms to complex ones. The packages VSFFT and VRFFTPK have been used as the basis for a vectorized three-dimensional Poisson (or Helmholtz) equation solver, HS3CRT, that has been written in the style of a similar two-dimensional solver in the Poisson package FISHPAK [9].

TRANX/Y has been tested on a CRAY-1 computer for a three-dimensional vector $x_i \equiv x_{i,k,j}$, $i = 1, 2, ..., n$, $j = 1, 2, ..., J$, $k = 1, 2, ..., K$ with $J = K = 64$ and $n = 62, 63$, or 64 whichever case is the most suitable one for efficient FFT. The maximum relative error measured by comparing a random number data vector which is transformed and then inversely transformed with the original vector is less than $2 \cdot 10^{-12}$ in all cases (machine accuracy approximately 10^{-14}). The computation times for Fourier synthesis and analysis (i.e., for 2 transforms) are given in Table II. For comparison, a vectorized algorithm which computes the transforms without FFT takes about 2.8 s computation time for these vector sizes. The symmetric FFTs described by Swarztrauber [6] should give even faster transforms for several cases (see Table I) but software implementations are not available. No comparable algorithms exist for cases DS–NS, D–NS, and its mirror symmetric variants.

The dependence of computation time on the value of n for case D–NS is given in Table III. It shows, as expected, that it is important to select n such that $2n + 1$ factors in at least a few prime numbers.

In a preliminary stage of coding, TRANX/Y employed a subroutine FFT99 by Temperton (personal communication, 1979). This routine is available both in CRAY assembler language (CAL) and in Fortran. It allows us to perform real periodic Fourier transforms on data which need not be stored with a unit increment in main storage (non-unit "stride"). This allows us to eliminate copying the data from one format into another; this simple operation takes about 30% of the com-

TABLE II

Computing Times in Seconds on a CRAY-1 for Fourier Synthesis and Analysis of a Three-Dimensional Array of Size $n \times J \times K$ Transformed in One Direction

		At $i = n$			
		D	N	DS	NS
At $i = 1$	D	0.190	0.206	—	0.322
	N	0.206	—	—	—
	DS	—	—	0.199	0.193
	NS	0.320	—	0.194	0.196

Note. With $n = 63$ (D–D), $n = 62$ (D–NS, and NS–D), or $n = 64$ (otherwise), $J = K = 64$, for various boundary conditions using TRANX (—: not coded). For periodic boundary conditions the computing time is 0.178 s.

TABLE III

Factorization of $(2n+1)$ and Computing Time/Second on a CRAY-1 versus n for D–NS, $J=K=64$

n	$(2n+1)$	CPU time
52	$3\times5\times7$	0.367
53	107	2.487
54	109	2.578
55	3×37	0.914
56	113	2.765
57	5×23	0.674
58	$3\times3\times13$	0.501
59	7×17	0.727
60	11×11	0.698
61	3×41	1.100
62	$5\times5\times5$	0.321
63	127	3.470
64	3×43	1.200

putation time for cyclic boundary conditions. Moreover the CAL version is about 40% faster than VRFFT while the Fortran-version is about 5% slower. However, the main shortcoming of FFT99 is its restriction to even values of n; so it cannot be used for case D–NS or NS–D. As has been pointed out by a reviewer, Temperton [10] has developed a version FFT77 which allows for odd n, non-unit stride and is about 30% faster than FFT99, but this version was not available to us.

The three-dimensional Poisson solver based on FFT using the routine TRANX/Y is about twice as fast as a coding based on cyclic reduction for staggered boundary conditions [1, 2]. TRANX/Y has been implemented in a fluid dynamics code in which Poisson's equation in terrain following coordinates—due to variable coefficients—is solved by iteratively employing the Poisson solver for a nontransformed grid [11]. In the iteration for D–NS the faster algorithms for DS–NS and for D–N may be applied alternately. This is an efficient approach if n does not allow for high factorization of $2n+1$.

ACKNOWLEDGMENTS

We thank Dr. H. Volkert for help in developing the subroutine TRANX/Y and valuable suggestions on the manuscript. Mrs. Linda Lindgren for help in developing the packages VRFFTPK and VSFFT, and Mrs. J. Freund for writing the manuscript with a text editor system.

REFERENCES

1. U. SCHUMANN AND R. A. SWEET, *J. Comput. Phys.* **20**, 171 (1976).
2. U. SCHUMANN AND R. A. SWEET, *Proceedings, 5th Internat. Conf. Numerical Methods in Fluid Dynamics*, Lecture Notes in Phys. Vol. 59 (Springer, Berlin, 1976), p. 398.

3. R. A. SWEET, *J. Comput. Phys.* **12**, 422 (1973).
4. R. B. WILHELMSON AND J. H. ERICKSEN, *J. Comput. Phys.* **25**, 319 (1977).
5. P. N. SWARZTRAUBER, *SIAM Rev.* **19**, 490 (1977).
6. P. N. SWARZTRAUBER, *Math. Comput.* **47**, 323 (1986).
7. J. W. COOLEY, P. A. W. LEWIS, AND P. D. WELCH, *J. Sound Vib.* **12**, 315 (1970).
8. P. N. SWARZTRAUBER, in *Large Scale Scientific Computation*, edited by S. Parter (Academic Press, New York, 1984), p. 271.
9. J. ADAMS, P. N. SWARZTRAUBER, AND R. SWEET, *ACM Trans. Math. Software* **5**, 352 (1979).
10. C. TEMPERTON, *J. Comput. Phys.* **52**, 340 (1983).
11. U. SCHUMANN AND H. VOLKERT, in *Efficient Solutions of Elliptic Systems*, Notes in Numerical Fluid Mechanics Vol. 10, edited by W. Hackbusch (Vieweg, Braunschweig, 1984), p. 109.