Computing 22, 79—91 (1979)

Computing

Fast Algorithms for the Conjugate Periodic Function

M. H. Gutknecht, Zürich

Received September 1, 1978

Abstract — Zusammenfassung

Fast Algorithms for the Conjugate Periodic Function. Two fast algorithms for the approximate computation of the conjugate periodic function are described. They are based on the fast Fourier transform and enable us to reduce the expenses to $O(N \log N)$ operations compared with $O(N^2)$ operations for Wittich's classical method. The second algorithm, for which an ALGOL 60 procedure is listed, allows to evaluate the conjugate function on the even (or odd) numbered lattice points separately. (This feature is important for some applications.)

Schnelle Algorithmen für die Konjugierte einer periodischen Funktion. Es werden zwei schnelle Algorithmen für die angenäherte Berechnung der Konjugierten einer periodischen Funktion beschrieben. Ihre Grundlage ist die schnelle Fourier-Transformation. Gegenüber der klassischen Methode von Wittich wird der Rechenaufwand von $O(N^2)$ Operationen auf $O(N \log N)$ Operationen vermindert. Der zweite Algorithmus, für den eine ALGOL 60-Prozedur angegeben wird, erlaubt es, die Konjugierte separat auf den geraden (oder ungeraden) Gitterpunkten auszuwerten. (Diese Eigenschaft ist in gewissen Anwendungen wichtig.)

1. Introduction

Several numerical methods for conformal mappings and some computational techniques in related areas of applied complex analysis require the approximate construction of the conjugate periodic function KX (defined in Section 2) of a given real 2π-periodic function X. With Wittich's classical method [6, pp. 74—80] one needs $2N$ samples $x_k := X(t_k)$ at equidistant lattice points $t_k := k\pi/N$ and N^2 real multiplications to get $2N$ approximate values y_k of $KX(t_k)$. (We are neglecting additions.) But the same values y_k may be obtained by first constructing the normalized trigonometric polynomial T interpolating X at the lattice points t_k and then evaluating the conjugate trigonometric polynomial KT at these points t_k [6]. Because conjugation in frequency space is trivial, one needs essentially only two fast Fourier transforms (FFT) for the realization of this method, which, e.g., has been applied in [12, 13]. Thus, the costs are cut down to about $\frac{1}{2} N \log_2 N - N$ complex multiplications if, e.g., N is a power of 2. Hence, this algorithm allows to work with large values of N, while former computations were limited to small ones [6]. It will be mentioned briefly by Henrici [11], too, who proposed it to the author in 1974. But here we present some additional details in

0010-485X/79/0022/0079/$ 02.60

Sections 3 and 4. Moreover, we will need later the furmulas upon which this algorithm is based.

As is easy to see from the structure of Wittich's matrix, the even (odd) numbered components of the vector $y \in \mathbb{R}^{2N}$ only depend on the odd (even) numbered components of $x \in \mathbb{R}^{2N}$. Niethammer [14] has taken advantage of this fact when applying the nonlinear *SOR* method to the discretized Theodorsen integral equation. In Section 6 we state a fast algorithm, based on the *FFT* too, for this separate discrete conjugation on even or odd, respectively, lattice points. With this algorithm we need $\frac{1}{2} N \log_2 N - \frac{1}{2} N$ complex multiplications to get either the even or the odd numbered components of y. For the operation count we assume again that N is a power of 2; however, the algorithm works for any even positive N. An ALGOL 60 procedure for it is listed in Section 9.

In Section 7 we discuss the advantages if X is a $(2\pi/m)$-periodic function or an even function. Here too, the frequency space approach easily leads to general results, which surpass those of Gekeler [9].

The fast algorithms described in this paper are stable and relatively indifferent to rounding errors. However, discretization errors must be watched if X is not a sufficiently smooth function. Fortunately, in the last case a modification based upon the theory of attenuation factors [8] often yields good results. For this we refer to a subsequent paper [10].

2. The Conjugate Periodic Function

Let $L_2[0, 2\pi]$ denote the space of real valued 2π-periodic square integrable functions. We recall the

Definition: For $X \in L_2[0, 2\pi]$ the function $Y := KX$ defined by

$$Y(t) := KX(t) := \frac{1}{2\pi} P.V. \int_0^{2\pi} X(\tau) \cot \frac{t-\tau}{2} d\tau \tag{1}$$

is called the *conjugate function* of X. The operator K will be refered to as *conjugation operator*. (The integral in (1) is a Cauchy principal value Lebesgue integral.)

Properties of K:

(i) $KX \in L_2[0, 2\pi]$, i.e. K is a linear operator mapping $L_2[0, 2\pi]$ into itself [22, p. 128].

(ii) If $a_0, a_1, a_2, \ldots, b_1, b_2, \ldots$ are the real Fourier coefficients of X, then KX has the real Fourier coefficients $0, -b_1, -b_2, \ldots, a_1, a_2, \ldots$ [22, p. 128].

(iii) If c_k, $k = 0, \pm 1, \pm 2, \ldots$ are the complex Fourier coefficients of X, then KX has the complex Fourier coefficients $-i \operatorname{sign}(k)\, c_k$, $k = 0, \pm 1, \pm 2, \ldots$, where $i := \sqrt{-1}$.

(iv) The operator norm $\| K \|$ of K equals 1.

(v) K is skrew-symmetric: $(KX_1, X_2) = -(X_1, KX_2)$ if $(.,.)$ denotes the usual inner product in $L_2[0, 2\pi]$.

3. Discrete Conjugation

The simple form of the conjugation operator K when transformed into the space of Fourier coefficients and the efficiency of the fast Fourier transform (FFT) motivate the following algorithm for the approximate computation of KX:

We discretize $[0, 2\pi]$ by $2N$ equidistant lattice points $t_k := k\pi/N$ and let $x_k := X(t_k)$, $k=0, \ldots, 2N-1$. Applying the real discrete Fourier transform F^R_{2N} to

$$x := (x_0, \ldots, x_{2N-1})^T$$

yields the coefficients $a_0, \ldots, a_N, b_1, \ldots, b_{N-1}$ of the trigonometric polynomial

$$T(t) := \frac{a_0}{2} + \sum_{m=1}^{N-1} (a_m \cos mt + b_m \sin mt) + \frac{a_N}{2} \cos Nt \tag{2}$$

that interpolates X at the points t_k:

$$T(t_k) = X(t_k),\ k=0, \ldots, 2N-1. \tag{3}$$

Due to property (ii) of K the conjugate trigonometric polynomial KT is

$$KT(t) = \sum_{m=1}^{N-1} (-b_m \cos mt + a_m \sin mt) + \frac{a_N}{2} \sin Nt.$$

At least if X is a sufficiently smooth function and N is large, we may expect that KT is a good approximation to KX and, in particular, that

$$KT(t_k) \approx KX(t_k),\ k=0, \ldots, 2N-1. \tag{4}$$

These values $KT(t_k)$ can be computed simultaneously by an inverse real discrete Fourier transform $(F^R_{2N})^{-1}$.

Let $y_k := KT(t_k)$, $k=0, \ldots, 2N-1$, and $y := (y_0, \ldots, y_{2N-1})^T$. Then the *discrete conjugation* $x \mapsto y$ is a composition of three linear mappings of $\mathbb{R}^{2N}$ into itself:

$$\begin{array}{ccc} x & \xrightarrow{F^R_{2N}} & (a_0, \ldots, a_N; b_1, \ldots, b_{N-1})^T \\ & & \downarrow K^R_{2N} \\ y & \xleftarrow{(F^R_{2N})^{-1}} & (0, -b_1, \ldots, -b_{N-1}, 0; a_1, \ldots, a_{N-1})^T \end{array} \tag{5}$$

Hence, it is itself such a mapping. The corresponding matrix

$$K_{2N} := (F^R_{2N})^{-1} K^R_{2N} F^R_{2N} \tag{6}$$

is called *Wittich's matrix*. It is a discrete analogue of the conjugation operator K. Its elements are [6, p. 76]

$$(K_{2N})_{kj}=\begin{cases} 0 & \text{if } j-k \text{ even,} \\ \dfrac{1}{N}\cot\dfrac{(k-j)\pi}{2N} & \text{if } j-k \text{ odd,} \end{cases}$$

$(k,j=0,\ldots,2N-1)$. Note that K is a skew-symmetric circulant Toeplitz matrix, which has zeros in a chessboard layout.

The multiplication of $x\in\mathbb{R}^{2N}$ by K_{2N} can also be considered as an application of the trapezoidal rule to the singular integral (1). Before the rediscovery of the fast Fourier transform [5] multiplication by Wittich's matrix was one of several methods [cf. 6, pp. 74—85] commonly used for the numerical computation of conjugate periodic functions. Since every other element of K_{2N} is 0, and every element appears twice (with different sign) in each row, this operation requires N^2 real multiplications.

On the other hand, if N is even, the real discrete Fourier transform F^R_{2N} and its inverse $(F^R_{2N})^{-1}$ can be reduced to a complex $FFT\,F_N$ of half as much complex data (cf. [2, 4] and Section 6). These reductions require only $\frac{1}{2}N$ complex multiplications each. (We assume that multiplications by 2 or $\frac{1}{2}$ and all additions are neglected and that the values $\cos t_k$ and $\sin t_k$ have been stored.) If N is a power of 2, the complex FFT costs $\frac{1}{2}N\log_2 N-N+1$ complex multiplications [1]. If N is a power of 4 or 8, this number can even be cut down [1]. Thus we can execute the discrete conjugation (5) in $N\log_2 N-N+1$ complex multiplications. Note that one could work with an FFT program that yields the coefficients only in reverse binary order [17]. On the other hand, N needs not be a power of 2 [18, 19].

For example, this simple but fast conjugation algorithm has been applied by P. Henrici's students Jeltsch [12] and Lundwall-Skaar [13].

Chawla and Ramakrishnan [3] have proposed a correction term improving the trapezoidal rule for integrals of the type (1). However, it turns out that this correction term vanishes for the y_k considered here.

4. Complex Notation

In complex notation, the real valued trigonometric polynomial T defined in (2) [and characterized by (3) and a vanishing $\sin(Nt)$-term] is

$$T(t)=\sum_{m=-N}^{N}{}' c_m e^{imt}; \tag{7}$$

where the prime indicates that the terms corresponding to $m=-N$ and $m=N$ are to be taken with weight $\frac{1}{2}$, and where

$$c_m=\tfrac{1}{2}(a_m-ib_m),\ c_{-m}=\bar{c}_m,\ m=0,\ldots,N, \tag{8}$$

if $b_0:=b_N:=0$. The complex vector

$$c:=(c_0,\ldots,c_{2N-1}),\ \text{with}\ c_{2N-k}:=c_{-k}\,(k=1,\ldots,N),$$

may be obtained by a complex discrete Fourier transform F_{2N} applied to x, i.e. $c = F_{2N} x$. In view of property (iii) of the conjugation operator K, we get instead of (6)

$$K_{2N} = F_{2N}^{-1} K_{2N}^{C} F_{2N}, \tag{9}$$

where, with $w_{2N} := \exp(i\pi/N)$,

$$(F_{2N})_{kj} := \frac{1}{2N} w_{2N}^{-kj} \ (k, j = 0, \ldots, 2N-1),$$

$$F_{2N}^{-1} = 2N F_{2N}^{H}, \tag{10}$$

$$K_{2N}^{C} := \operatorname{diag}(0, -i, \ldots, -i, 0, i, \ldots, i).$$

Thus — as for any circulant matrix [21, Section 17.6] — the columns of F_{2N}^{-1} are an orthogonal basis of eigenvectors for Wittich's matrix K_{2N}, which has $N-1$ eigenvalues $-i$, $N-1$ eigenvalues $+i$, and two zero eigenvalues. In particular, the spectral norm $\| K_{2N} \|$ equals 1.

Of course, the direct implementation of (9) with the FFT is only about half as efficient as the real version (6).

5. Conjugation on Even or Odd Lattice Points

Since Wittich's matrix K_{2N} features zeros in a chessboard layout including zeros in the diagonal, the odd (even) numbered components of $y = K_{2N} x$ only depend on the even (odd) numbered components of x. In fact, if we let

$$x' := (x_1, x_3, \ldots, x_{2N-1})^T, \ x'' := (x_0, x_2, \ldots, x_{2N-2})^T,$$

and define the permutation matrix P_{2N} by

$$P_{2N} x = \begin{pmatrix} x'' \\ x' \end{pmatrix},$$

then

$$P_{2N} K_{2N} P_{2N} = \begin{pmatrix} 0 & -L_N^T \\ L_N & 0 \end{pmatrix}, \tag{11}$$

where

$$(L_N)_{kj} := (K_{2N})_{2k+1, 2j} = \frac{1}{N} \cot \frac{(2k-2j+1)\pi}{2N}, \ (k, j = 0, \ldots, N-1). \tag{12}$$

Hence, $y = K_{2N} x$ is equivalent to $P_{2N} y = (P_{2N} K_{2N} P_{2N}) P_{2N} x$, or,

$$y'' = -L_N^T x', \tag{13 a}$$

$$y' = L_N x''. \tag{13 b}$$

In some applications [14] it is essential that these two linear transformations can be executed separately. However, (13 a) and (13 b) only define slow algorithms requiring $\frac{1}{2} N^2$ real multiplications. In the next section we will present fast algorithms instead.

6. A Fast Algorithm for the Conjugation on Even or Odd Lattice Points

Since L is a circulant Toeplitz matrix too, there exists a factorization analogous to (9), i.e. $F_N L F_N^{-1}$ is a diagonal matrix, which easily can be obtained explicitely [21, Section 17.6]. We will use an algorithmic approach to derive it and add another more complicated factorization, which will allow to capitalize the fact that x', x'', y', y'' are real vectors.

We let

$$c' := F_N x', \; c'' := F_N x'', \; c := F_{2N} x,$$

and again $w_{2N} := \exp(i\pi/N)$. As is well-known, see e.g. [4], the FFT itself is based on the fact that

$$c_k = \tfrac{1}{2}(c_k'' + w_{2N}^{-k} c_k'), \; c_{k+N} = \tfrac{1}{2}(c_k'' - w_{2N}^{-k} c_k'), \; k = 0, \ldots, N-1,$$

or in matrix notation,

$$c = \tfrac{1}{2}\begin{pmatrix} I_N & W_N^H \\ I_N & -W_N^H \end{pmatrix}\begin{pmatrix} c'' \\ c' \end{pmatrix}, \tag{14}$$

where

$$W_N := \operatorname{diag}(1, w_{2N}, w_{2N}^2, \ldots, w_{2N}^{N-1}).$$

Hence,

$$F_{2N} = \tfrac{1}{2}\begin{pmatrix} I_N & W_N^H \\ I_N & -W_N^H \end{pmatrix}\begin{pmatrix} F_N & 0 \\ 0 & F_N \end{pmatrix} P_{2N}, \tag{15a}$$

$$F_{2N}^{-1} = P_{2N}\begin{pmatrix} F_N^{-1} & 0 \\ 0 & F_N^{-1} \end{pmatrix}\begin{pmatrix} I_N & I_N \\ W_N & -W_N \end{pmatrix}. \tag{15b}$$

If $I_N^\dagger$ denotes the $N \times N$-matrix

$$I_N^\dagger := \operatorname{diag}(0, 1, 1, \ldots, 1),$$

then

$$K_{2N}^C = -i\begin{pmatrix} I_N^\dagger & 0 \\ 0 & -I_N^\dagger \end{pmatrix}, \; \tfrac{1}{2}\begin{pmatrix} I_N & I_N \\ W_N & -W_N \end{pmatrix} K_{2N}^C \begin{pmatrix} I_N & W_N^H \\ I_N & -W_N^H \end{pmatrix} = -i\begin{pmatrix} 0 & I_N^\dagger W_N^H \\ I_N^\dagger W_N & 0 \end{pmatrix}.$$

Using (9) and (15), we get from (11)

$$\begin{pmatrix} 0 & -L_N^T \\ L_N & 0 \end{pmatrix} = -i\begin{pmatrix} 0 & F_N^{-1} I_N^\dagger W_N^H F_N \\ F_N^{-1} I_N^\dagger W_N F_N & 0 \end{pmatrix},$$

or,

$$L_N = -i F_N^{-1} I_N^\dagger W_N F_N, \tag{16a}$$

$$L_N^T = \; i F_N^{-1} I_N^\dagger W_N^H F_N. \tag{16b}$$

Thus, L_N has the eigenvalues 0 and $-i w_{2N}^k$, $k = 1, \ldots, N-1$. The columns of F_N^{-1} are the corresponding right-hand eigenvectors. They are also left-hand eigenvectors, but the correspondence to the non-vanishing eigenvalues is in reverse order. Moreover,

$$L_N^T L_N = F_N^{-1} I_N^\dagger W_N^H W_N F_N = F_N^{-1} I_N^\dagger F_N, \tag{17}$$

i.e. the matrix $L_N^T L_N$ has rank $N-1$ and $N-1$ eigenvalues 1. In particular, spectral norm and spectral radius of L_N are 1.

It would be easy to deduce from (16) similar factorization formulas containing the real transform F_N^R instead of F_N. But it is more rewarding to remember the trick to reduce F_N^R to $F_{N/2}^C$ [2, p. 65; 4]. Assume that N is even, and $n := N/2$. For any $x' \in \mathbb{R}^N$ we can compute

$$c'_0 := \begin{pmatrix} c'_2 \\ c'_1 \end{pmatrix} := G_N x', \text{ where } G_N := \begin{pmatrix} F_n & 0 \\ 0 & F_n \end{pmatrix} P_N \tag{18}$$

by one single $FFT\ F_n$ only. (Here, c'_0, c'_1, c'_2 are vectors, not components of c'.) In fact, if

$$\begin{pmatrix} x'_2 \\ x'_1 \end{pmatrix} := P_N x', \quad d' := F_n(x'_2 + i\, x'_1), \tag{19}$$

then we get $c'_2, c'_1 \in \mathbb{C}^n$ from

$$c'_2 = \tfrac{1}{2}(d' + \overline{R_n d'}), \quad c'_1 = -\tfrac{i}{2}(d' - \overline{R_n d'}), \tag{20}$$

where the bar indicates conjugation of complex numbers, and R_n is the reversion operator:

$$(R_n d)_k := \begin{cases} d_0 & \text{for } k=0, \\ d_{n-k} & \text{for } k=1, \ldots, n-1. \end{cases}$$

On the other hand, for the inverse mapping $G_N^{-1} : c'_0 \mapsto x'$ we use the formulas

$$d' = c'_2 + i\, c'_1, \tag{21 a}$$

$$x'_2 = \mathrm{Re}\,(F_n^{-1} d'), \quad x'_1 = \mathrm{Im}\,(F_n^{-1} d'), \quad x' = P_N \begin{pmatrix} x'_2 \\ x'_1 \end{pmatrix}. \tag{21 b}$$

The mentioned algorithm for F_N^R utilizes the factorization (15 a) (with $2N$ replaced by N) to compute $c' = F_N x'$, but the evaluation of (18) needed in (15 a) is realized according to (19) and (20). Finally, the real coefficients are easily obtained from (8):

$$a'_k := 2\,\mathrm{Re}\, c'_k, \quad b'_k := -2\,\mathrm{Im}\, c'_k, \quad k=0, \ldots, n. \tag{22}$$

(Note that only $[n/2]+1$ of the components of c' are required for that.) Similarly, for the inverse transform $x' = F_N^{-1} c' = (F_N^R)^{-1}\ (a'_0, \ldots, a'_n;\ b'_1, \ldots, b'_{n-1})^T$ we work with (8), (15 b), and (21).

Since the evaluation of (18) and the corresponding inverse transform G_N^{-1} can be done very efficiently, we are inclined to rewrite (13 a), using (16 b) and (15), in the form

$$y'' = -L_N^T x' = -i\, F_N^{-1}\, I_N^\dagger\, W_N^H\, F_N x' = G_N^{-1}\, M'_N\, G_N x', \tag{23}$$

where

$$M'_N := -\tfrac{i}{2} \begin{pmatrix} I_n & I_n \\ W_n & -W_n \end{pmatrix} I_N^\dagger\, W_N^H \begin{pmatrix} I_n & W_n^H \\ I_n & -W_n^H \end{pmatrix}.$$

If we further define

$$V_n := \mathrm{diag}\,(1, w_{2N}, w_{2N}^2, \ldots, w_{2N}^n),$$

then

$$V_n^H = V_n^{-1},\ W_n = V_n^2,\ W_N = \begin{pmatrix} V_n & 0 \\ 0 & i\,V_n \end{pmatrix},$$

and

$$M'_N = \tfrac{1}{2}\left[\begin{pmatrix} -I_n & V_n^{-2} \\ V_n^2 & -I_n \end{pmatrix} - i\begin{pmatrix} I_n^\dagger & 0 \\ 0 & I_n^\dagger \end{pmatrix}\begin{pmatrix} I_n & V_n^{-2} \\ V_n^2 & I_n \end{pmatrix}\right]\begin{pmatrix} V_n^{-1} & 0 \\ 0 & V_n^{-1} \end{pmatrix}. \quad (24)$$

With M'_N in this form it is easy to implement the multiplication of $c'_0 = G_N x'$ by M'_N efficiently, cf. Section 9. Note that

$$M'_N c'_0 = G_N y'' = \begin{pmatrix} F_n y''_2 \\ F_n y''_1 \end{pmatrix} \in [F_n(\mathbb{R}^n)]^2,$$

i.e. both halves of this vector are complex transforms of a real vector. Thus, again, it is sufficient to compute only about half of the components, namely $(M'_N c'_0)_k, k = 0, \ldots, [N/4], N-[N/4]-1, \ldots, N-1$; only N complex multiplications are needed for that. (Here we assume again that the elements of V and V^2 have been stored. In a computer program, cf. Section 9, one would rather compute these cosine and sine values recursively. There are several methods to do that efficiently [16].)

The odd numbered components of y are computed similarly: According to (13 b), (16 a), and (15)

$$y' = L_N x'' = -i\,F_N^{-1} I_N^\dagger W_N F_N x'' = G_N^{-1} M''_N G_N x'', \quad (25)$$

where M''_N may be written in the form

$$M''_N := \tfrac{1}{2}\left[\begin{pmatrix} I_n & -V_n^{-2} \\ -V_n^2 & I_n \end{pmatrix} - i\begin{pmatrix} I_n^\dagger & 0 \\ 0 & I_n^\dagger \end{pmatrix}\begin{pmatrix} I_n & V_n^{-2} \\ V_n^2 & I_n \end{pmatrix}\right]\begin{pmatrix} V_n & 0 \\ 0 & V_n \end{pmatrix}. \quad (26)$$

According to (25), (23) and $G_N^{-1} = n\,G_N^H$ [cf. (10)], the matrices M'_N and M''_N are related to each other by $(M''_N)^H = -M'_N$.

Algorithm: To evaluate $-L_N^T x'$ compute $c'_0 = G_N x'$ [defined by (18)] using formulas (19) and (20), then multiply c'_0 by M'_N written in the form (24), and finally let

$$d'_k := (M'_N c'_0)_k + i\,(M'_N c'_0)_{n+k},\ k = 0, \ldots, n-1, \quad (27\,a)$$

$$y''_2 := \mathrm{Re}\,(F_n^{-1} d'),\ y''_1 := \mathrm{Im}\,(F_n^{-1} d'),\ -L_N^T x' := P_N \begin{pmatrix} y''_2 \\ y''_1 \end{pmatrix}. \quad (27\,b)$$

The computation of $L_N x''$ is done analogously except that M'_N is replaced by M''_N defined in (26).

For the computation of either one of $-L_N^T x'$ or $L_N x''$ we need only $\frac{1}{2} N \log_2 N - \frac{1}{2} N$ complex multiplications. So, for the evaluation of both, i.e. for the calculation of all components of $K_{2N} x$, we need $N \log_2 N - N$ multiplications; thus, the total expenses are the same as with the method sketched in Section 3, which, however, does not allow to compute y' (or y'') if x' (or x'', respectively) is unknown. Actually, in numerical experiments with programs coded similarly for both methods and containing the same complex base 2 FFT procedure, the central processor time

used up was the same for $N=64$, while for larger N the above new algorithm was even slightly faster.

7. Conjugation of Functions With Additional Symmetries

Now, let us assume that x is $2M$-periodic, $m:=N/M$ being an integer. Then $c:=F_{2N}x$ satisfies

$$c_{2Mk+l}=\begin{cases}0, & l=1,\ldots,M-1;\ k=0,\ldots,m-1,\\ \tilde{c}_k, & l=0;\qquad\quad k=0,\ldots,m-1,\end{cases}$$

where $\tilde{c}:=F_{2M}\tilde{x}$, $\tilde{x}:=(x_0,\ldots,x_{2M-1})^T$. Since K_{2N}^C is diagonal, the vector $d:=K_{2N}^C c$ is of the same type, and $y=K_{2N}x=F_{2N}^{-1}d$ is also $2M$-periodic. Its $2M$ first components $\tilde{y}:=(y_0,\ldots,y_{2M-1})$ satisfy

$$\tilde{y}=F_{2M}^{-1}K_{2M}^C F_{2M}\tilde{x}=K_{2M}\tilde{x}. \tag{28}$$

Hence, the conjugation K_{2N} is just reduced to one in $\mathbb{R}^{2M}$, and we can apply the methods of Sections 3 and 6 to compute it efficiently. Hereby the expenses are nearly reduced by a factor of m (if $m\ll M$).

Note that — for any divisor M of N — this reduction is related to a partition of Wittich's matrix K_{2N}:

$$K_{2N}=\begin{bmatrix}A_1 & A_2 & \ldots & A_m\\ A_m & A_1 & \ldots & A_{m-1}\\ \vdots & \vdots & & \vdots\\ A_2 & A_3 & \ldots & A_1\end{bmatrix}; \tag{29 a}$$

where $A_1,\ldots,A_m$ are $2M\times 2M$-matrices, and

$$\sum_{k=1}^{m}A_k=K_{2M}. \tag{29 b}$$

Next, in addition, suppose that $\tilde{x}$ is an even function, i.e. $\tilde{x}_{2M-k}=\tilde{x}_k$, $k=1,\ldots,M$. Then the real discrete Fourier transform F_{2M}^R yields $\tilde{b}_1=\ldots=\tilde{b}_{M-1}=0$, and for the conjugate function we get $\tilde{a}_0=\ldots=\tilde{a}_M=0$, i.e. $\tilde{y}$ is an odd function:

$$\tilde{y}_0=\tilde{y}_M=0,\ \tilde{y}_{M-k}=-\tilde{y}_k,\ k=1,\ldots,M.$$

There is a modified real FFT algorithm [4] that capitalizes these symmetries and nearly halves the expenses (if M is large). Unfortunately, this algorithm is unstable, creating rounding errors proportional to M. This fact — not pointed out in [4] — limits the application of this algorithm to the case where the mantissa of the computer is longer than really needed for a particular practical problem. Another modification by Ziegler [20] is difficult to implement and causes probably much overhead.

For the transformations L and L^T it would also be possible to state fast algorithms taking advantage of an even (or odd) function $\tilde{x}$. But here too, the idea used in [4] would cause instability.

8. Accuracy and Stability

Rounding Errors and Stability

The complex $FFT\,F_N$ and the transforms G_N, M'_N and M''_N are all very stable with respect to rounding errors. In fact, F_N and G_N are up to a scale factor unitary, and every matrix in their factorization corresponding to the FFT algorithm [16] has this property too. Moreover, it is easy to deduce from (17), (23), and (25) that

$$\| M'_N \| = \| M''_N \| = 1. \tag{30}$$

Finally, the number of arithmetic operations performed is relatively small. So, the algorithms described in Sections 3 and 6 are very stable both with respect to perturbation of data and rounding errors.

Discretisation Errors

If the given function X is not sufficiently smooth or if N is not sufficiently large, the trigonometric polynomial T (defined by (2) or (7)) may be a poor approximation to X. Then $K\,T$ is a poor approximation to $K\,X$. Even at the lattice points, where T interpolates X, $K\,T$ may deviate much from $K\,X$. A very simple but competitive error bound in terms of the (complex) Fourier coefficients C_k of X was given by Gaier [7]: Assuming that the Fourier series of X converges absolutely and pointwise he showed that

$$\beta := |C_{-N}| + |C_N| + 2 \sum_{|k|>N} |C_k|$$

is a bound both for the uniform norms $\| X - T \|_\infty$ and $\| KX - KT \|_\infty$. Likewise [7],

$$\beta' := N \sum_{|k|>N} |C_k| + \sum_{|k|\geqq N} |k|\,|C_k|$$

is a bound for the norms $\| X' - T' \|_\infty$ and $\| (KX)' - (KT)' \|_\infty$ if this bound β' exists and X is continuously differentiable. Actually, these bounds are solely based on the well-known formula

$$c_k = \sum_{j=-\infty}^{+\infty} C_{k+2Nj}$$

explaining the so-called *aliasing effect*, cf. e.g. [4].

The bound β confirms that our approach is appropriate if X is analytic on a sufficiently wide strip $S_\alpha := \{z : |\operatorname{Im} z| \leqq \alpha\}$, since

$$\beta \leqq 2 \operatorname{Coth} \tfrac{\alpha}{2}\, e^{-\alpha N} \max_{z \in S_\alpha} |X(z)|$$

in this case [7]. However, in other circumstances the discretization error might be much smaller if we approximated X by elements of another family of functions (e.g. splines) that is more appropriate than the family of trigonometric polynomials of degree N. In [10] we will apply attenuation factors [8] to realize this idea.

9. An ALGOL 60 Procedure for the Second Algorithm

The following ALGOL procedure *CONJUG* is an implementation of the algorithm described in Section 6. The $2N$-th roots of 1 needed, w_{2N}^k, $k=1, \ldots, [N/4]$ are computed according to a difference equation also used by Reinsch [15, p. 435] and described in [16]. It is assumed that a code procedure *COMFFT* for the complex fast Fourier transform without scale factor $1/n$, i.e. for $n F_n$, and for the inverse transform F_n^{-1} is available. (E.g., this could be Singleton's [18] procedure *FFT* for the inverse transform; to implement $n F_n$ one had to apply it to the conjugate complex data and to conjugate the resulting Fourier coefficients.) Of course, if this code procedure *COMFFT* requires that n is a power of 2, then the same restriction holds for the parameter N of *CONJUG*. If *COMFFT* includes scaling, as e.g. Reinsch's procedure *FOUCOM* [15, p. 435], the scaling factor *FAC* in *CONJUG* must be modified: $FAC := N/8.0$.

```
'PROCEDURE' CONJUG (NLONG, EVEN) TRANS:(A, B) ;
   'VALUE' NLONG, EVEN ;
   'INTEGER' NLONG ;   'BOOLEAN' EVEN ;   'ARRAY' A, B ;
   'COMMENT' CONJUGATION OF THE SUBSEQUENCE CORRESPONDING TO THE EVEN,
             IF EVEN = 'TRUE', OR ODD, IF EVEN = 'FALSE', RESPECTIVELY,
             NUMBERED POINTS OF A 2N-PERIODIC ORIGINAL SEQUENCE,
             WHERE N = ABS(NLONG) IS EVEN.
             IF NLONG > 0, THE SUBSEQUENCE MUST BE STORED IN A[0:N-1],
             WHILE B[0:N÷2-1] IS ONLY USED INTERNALLY. IF NLONG < 0,
             THE SUBSEQUENCE MUST BE STORED ALTERNATELY IN A[0:N÷2-1]
             AND B[0:N÷2-1], AND THESE ARRAYS NEED NOT BE LARGER.
             THE RESULT TAKES THE PLACE OF THE GIVEN SUBSEQUENCE.
             EXTERNAL PROCEDURE: COMFFT ;

   'BEGIN'
      'INTEGER' K, K2, N, NH, NH1, NK, NV ;
      'REAL' AR, AI, BR, BI, CK1, CK2, DC1, DS1, FAC, FR, FI, GR, GI,
        H, R, SK1, SK2, TWO ;

      'PROCEDURE' COMFFT (N, ANA, A, B) ;
         'VALUE' N, ANA ;
         'INTEGER' N ;   'BOOLEAN' ANA ;   'ARRAY' A, B ;
         'COMMENT' IF ANA, COMFFT YIELDS THE COMPLEX DISCRETE FOURIER
                   TRANSFORM (ANALYSIS) OF THE SAMPLES A[K] + I * B[K],
                   K=0(1)N-1. THE USUAL SCALE FACTOR 1/N IS SUPPRESSED.
                   IF ¬ANA, COMFFT YIELDS THE INVERSE COMPLEX DISCRETE
                   FOURIER TRANSFORM (SYNTHESIS) OF THE COEFFICIENTS
                   A[K] + I * B[K], K=0(1)N-1.
                   THE RESULT TAKES THE PLACE OF THE GIVEN DATA ;
         'CODE' 0 ;

      N := ABS(NLONG) ;
      NH := N ÷ 2 ;     NH1 := NH - 1 ;
      'IF' NLONG > 0 'THEN'
      'BEGIN'
         'FOR' K := 0 'STEP' 1 'UNTIL' NH1 'DO'
         'BEGIN'
           K2 := 2 * K ;
           A[K] := A[K2] ;
           B[K] := A[K2+1] ;
         'END' K ;
      'END' NLONG>0 ;
      'COMMENT' COMPLEX FFT WITHOUT SCALE FACTOR ;
      'IF' NH > 1 'THEN' COMFFT (NH, 'TRUE', A, B) ;
      'COMMENT' CONJUGATION IN FREQUENCY SPACE ;
      NV := NH ÷ 2 ;
      'IF' NV > 0 'THEN'  H := 2.0*ARCTAN(1.0)/NH 'ELSE' H := .0 ;
      R := 2.0 * SIN(H/2.0) ;     R := - R * R ;
      DC1 := -0.5 * R ;     DS1 := SIN(H) ;
      CK1 := 1.0 ;     SK1 := .0 ;
      FAC := 0.5 / N ;     TWO := 2.0 ;
      H := 2.0 * FAC * (A[0] - B[0]) ;
      'IF' ¬ EVEN 'THEN'
      'BEGIN'
         DS1 := - DS1 ;     H := - H ;     TWO := - TWO ;
      'END' ;
```

```
A[0] := H ;      B[0] := - H ;
'FOR' K := 1 'STEP' 1 'UNTIL' NV 'DO'
'BEGIN'
   DC1 := R * CK1 + DC1 ;      CK1 := CK1 + DC1 ;
   DS1 := R * SK1 + DS1 ;      SK1 := SK1 + DS1 ;
   CK2 := (CK1 - SK1) * (CK1 + SK1) ;
   SK2 := TWO * SK1 * CK1 ;
   NK := NH - K ;
   FR := A[K] + A[NK] ;
   GI := A[NK] - A[K] ;
   FI := B[K] - B[NK] ;
   GR := B[NK] + B[K] ;
   AR := FR * CK1 - FI * SK1 ;
   AI := FI * CK1 + FR * SK1 ;
   BR := GR * CK1 - GI * SK1 ;
   BI := GI * CK1 + GR * SK1 ;
   FR := AR * CK2 - AI * SK2 ;
   FI := AI * CK2 + AR * SK2 ;
   GR := BR * CK2 + BI * SK2 ;
   GI := BI * CK2 - BR * SK2 ;
   'IF' EVEN 'THEN'
   'BEGIN'
     GR := AI - GR ;   GI := AR + GI ;
     FR := BI - FR ;   FI := BR + FI ;
   'END' EVEN
   'ELSE'
   'BEGIN'
     H := GR + AI ;   GR := GI - AR ;   GI := H ;
     H := FR + BI ;   FR := FI - BR ;   FI := H ;
   'END' ODD ;
   AR := GR + GI ;      AI := GR - GI ;
   BR := FR + FI ;      BI := FR - FI ;
   A[K] := (AR - BI) * FAC ;
   A[NK] := (AR + BI) * FAC ;
   B[K] := (BR + AI) * FAC ;
   B[NK] := (BR - AI) * FAC ;
'END' K ;
'COMMENT' INVERSE COMPLEX FFT ;
'IF' NH > 1 'THEN' COMFFT (NH, 'FALSE', A, B) ;
'IF' NLONG > 0 'THEN'
'BEGIN'
   'FOR' K := NH1 'STEP' -1 'UNTIL' 0 'DO'
   'BEGIN'
     K2 := 2 * K ;
     A[K2+1] := B[K] ;
     A[K2] := A[K] ;
   'END' K ;
'END' LONGA ;
'END' CONJUG ;
```

References

[1] Bergland, G. D.: A fast Fourier transform algorithm using base 8 iterations. Math. Comp. *22*, 275—279 (1968).

[2] Bingham, C., Godfrey, M. D., Tukey, J. W.: Modern techniques of power spectrum estimation. IEEE Trans. Audio Electroacoust. AU-*15*, 56—66 (1967).

[3] Chawla, M. M., Ramakrishnan, T. R.: Numerical evaluation of integrals of periodic functions with Cauchy and Poisson type kernels. Numer. Math. *22*, 317—323 (1974).

[4] Cooley, J. W., Lewis, P. A. W., Welch, P. D.: The fast Fourier transform algorithm: Programming considerations in the calculation of sine, cosine and Laplace transforms. J. Sound Vib. *12*, 315—337 (1970).

[5] Cooley, J. W., Tukey, J. W.: An algorithm for the machine calculation of complex Fourier series. Math. Comp. *19*, 297—301 (1965).

[6] Gaier, D.: Konstruktive Methoden der konformen Abbildung. Berlin-Göttingen-Heidelberg: Springer 1964.

[7] Gaier, D.: Ableitungsfreie Abschätzungen bei trigonometrischer Interpolation und Konjugierten-Bestimmung. Computing *12*, 145—148 (1974).

[8] Gautschi, W.: Attenuation factors in practical Fourier analysis. Numer. Math. *18*, 373—400 (1972).

[9] Gekeler, E.: Über Iterationsverfahren bei einer Klasse nichtlinearer Gleichungssysteme. Dissertation, Ruhr-Universität Bochum, 1969.
[10] Gutknecht, M. H.: The evaluation of the conjugate function of a periodic spline. (Forthcoming.)
[11] Henrici, P.: Fast Fourier methods in computational complex analysis. (To appear.)
[12] Jeltsch, R.: Numerische konforme Abbildung mit Hilfe der Formel von Cisotti. Diplomarbeit, ETH Zürich, 1969.
[13] Lundwall-Skaar, C.: Konforme Abbildung mit Fast Fourier Transformationen. Diplomarbeit, ETH Zürich, 1975.
[14] Niethammer, W.: Iterationsverfahren bei der konformen Abbildung. Computing *1*, 146—153 (1966).
[15] Sauer, R., Szabo, I. (Hrsg.): Mathematische Hilfsmittel des Ingenieurs, Teil III. Berlin-Heidelberg-New York: Springer 1968.
[16] Singleton, R. C.: On computing the fast Fourier transform. Comm. ACM *10*, 647—654 (1967).
[17] Singleton, R. C.: Algorithm 338: ALGOL procedures for the fast Fourier transform. Comm. ACM *11*, 773—776 (1968).
[18] Singleton, R. C.: Algorithm 339: An ALGOL procedure for the fast Fourier transform with arbitrary factors. Comm. ACM *11*, 776—779 (1968); Comm. ACM *12*, 187 (1969).
[19] Singleton, R. C.: An algorithm for computing the mixed radix fast Fourier transform. IEEE Trans. Audio Electroacoust. AU-*17*, 93—103 (1969).
[20] Ziegler, H.: A fast Fourier transform algorithm for symmetric real-valued series. IEEE Trans. Audio Electroacoust. AU-*20*, 353—356 (1972).
[21] Zurmühl, H.: Matrizen und ihre technischen Anwendungen, 4. Aufl. Berlin-Göttingen-Heidelberg: Springer 1964.
[22] Zygmund, A.: Trigonometric Series, Vol. I. Cambridge: University Press 1959.

Dr. M. H. Gutknecht
Seminar für angewandte Mathematik
ETH-Zentrum HG
CH-8092 Zürich
Switzerland