stichting

mathematisch

centrum

∑
MC

C.G. VAN DER LAAN

A PROPOSAL FOR THE C06 CHAPTER OF THE NAG ALGOL 68 LIBRARY

kruislaan 413   1098 SJ   amsterdam

A proposal for the C06 chapter of the NAG ALGOL 68 Library.

by

C.G. van der Laan[*)]

ABSTRACT

A proposal and future plans for the C06 chapter of the NAG ALGOL 68 library are provided. Operators in ALGOL 68 and their documentation are given for the problems connected with the Discrete Fourier Transform and the evaluation of trigonometric sums.

---

*)
   Rekencentrum, University of Groningen, Postbus 800, 9700 AV Groningen,
   The Netherlands.

CONTENTS

PREFACE

This publication is intended to serve the goals:

. to make the proposed included software free,

. to open a discussion over the proposed and future contents of the C06 chapter,

. to stimulate contributors.

The author is custodian for the C06 chapter of the NAG ALGOL 68 library.

Contributors to this work are my colleagues Mr. R.J. van Oosten and Mr. J.P. Hollenberg. The former programmed and documented a preliminary version of this collection of operators as well as the technical routine C06FFT; the latter worked out the final version and its test programs. The theoretical foundation of the included operators and a discussion of the available implementations in a variety of libraries was published by the author (in Dutch) in the MC Syllabus 29.1b: Colloquium Numerieke programmatuur.

Although a lot of good and fast FFT-routines are available in FORTRAN, we have programmed our own for portability reasons; users of the provided operators may interface to a FORTRAN version of C06FFT. A machine readable copy of the source texts can be obtained by sending a magnetic tape to the author.

Because the NAG ALGOL 68 library has absorbed the matrix/vector package TORRIX of VAN DER MEULEN & VELDHORST no explicit reference is given to this work.

Finally, I would like to thank the members of the working group Approximation of Functions, especially Dr. N.M. Temme, for the necessary environment and the Mathematical Centre for providing the opportunity to publish this proposal in their TW-reports series; Prof. L.M. Delves and Dr. G. Hodgson of NAG are kindly acknowledged for reading and commenting a previous version of the documentation units.

I INTRODUCTION TO THE C06 CHAPTER OF THE NAG ALGOL 68 LIBRARY

CONTENTS OF THIS INTRODUCTION

1. Scope of the Chapter

Provided are implementations related to

$$(1.1) \qquad s(t) = \sum_{k=1}^{u} f_k(t)$$

with t a parameter and l,u either finite or infinite.

2. Background of the Problems

The summation problems are distinguished according to the structure of the terms $f_k(t)$.

2.1 Types of summation problems

As a particular case of (1.1) we have

$$(2.1) \qquad f_k(t) = a_k \, \phi_k(t)$$

with $\underline{a}$ the coefficient vector of the series expansion of s with respect to the basis functions $\{\phi_k\}$.

A variety of implementations have emerged according to the available information and desired results.

Examples of $\{\phi_k\}$ are:

. polynomials (see chapter E02 of the NAG library);

. trigonometric functions (this chapter);

. B-splines (see chapter E02 of the NAG library).

With the vector $\underline{a}$ we mean $(\ldots a_{-n}, a_{-n+1}, \ldots, a_{-1}, a_0, a_1, \ldots, a_{n-1}, a_n, \ldots)$ or a finite part of it.

2.1.1 Summation of trigonometric functions

Within this context relation (1.1) reads

$$(2.2) \qquad s(t) = \sum_{k=1}^{u} \sigma_k \, e^{ikt}$$

The implementations given consider finite special cases of (2.2):

. evaluation of s for one argument (dyadic operators);

. evaluation of s for equidistant arguments (monadic operators).

2.1.2 General summation

So far no implementations are provided; in the ALGOL 68 report the routine EULER is given.

2.2 Accuracy

An implementation is generally used on samples in a digital computer, with its intrinsic limitations, whereas we are thinking in abstractions. The difference herein is covered by the accuracy concept, which may be sub-divided into the error effects:

. residual or truncation error, i.e. the discrepancy between the sample information and the theoretical information of our problem;

. propagated error, i.e. the effect of perturbations;

. generated or rounding error, i.e. the effect of finite precision
  arithmetic.

In order to grasp these errors we use the concepts of the condition of a
problem (section 2.2.1) and the growth of an algorithm (section 2.2.2).
In each documentation unit bounds for the propagated and generated errors
are given in terms of the condition and growth. Apart from the residual
error a first order bound for the total error is the sum of the
indicated errors.

Example.

Given the table

Formula	Propagated error	Generated error
CO6EXPSUM a	$\| \underline{\Delta a} \|_2$ / max $(1/\sqrt{n}, \| \underline{a} \|_2)$	g_a * *small real*

we obtain for the error bound of $\{\underline{c} \mid c_j = \sum\limits_{k=0}^{n} a_k \, e^{2\pi ikj/n}$,

$$j = 0,1,\ldots,n-1\}$$

$$\frac{\| \, c - CO6EXPSUM \, a \, \|_2}{\max \, (1, \| c \|_2)} \leq \text{propagated error + generated error.}$$

2.2.1 Condition of a problem

Consider a problem T, which transforms data $\underline{a}$ into $\underline{b}$, then the first order
effect of the perturbation $\underline{\Delta a}$ of $\underline{a}$ is governed by

$$\underline{\Delta b} = \sum_j \frac{\partial Ta}{\partial a_j} \, \Delta a_j ,$$

a weighted sum of the perturbations in the data.

The amplification or condition of the problem can be defined by

$$C = \frac{\| \underline{\Delta b} \|}{\| \underline{b} \|} \, / \, \frac{\| \underline{\Delta a} \|}{\| \underline{a} \|} .$$

If we choose the 2-norm then we obtain for the Discrete Fourier Transform
(DFT)

4

$$C = 1.$$

2.2.2 Growth of an algorithm

Generally a process T in finite precision arithmetic is thought of as a process in exact arithmetic with perturbated parameters, $\underline{\Delta a}$. A bound for these perturbations can be obtained by backward error analysis; the factors involved are the so-called growth and the machine precision, ε. In other words, consider a process P for the above problem T in finite precision arithmetic, then there exists a $\underline{\Delta a}$, such that

$$P(\underline{a}) = fl(T(\underline{a})) = T(\underline{a} + \underline{\Delta a});$$

with the growth function, g, defined by

$$\frac{\|\underline{\Delta a}\|}{\|\underline{a}\|} = g * \varepsilon.$$

2.3 Data representation

Definitions:

o.   a vector is called a periodic n-vector if $a_k = a_{k+n}$, for all integer k;

r.   a vector is called a real periodic n-vector if (o) and the elements are real;

r.s.   a vector is called a symmetric real periodic n-vector if (r) and $a_k = a_{n-k}$, for all integer k;

r.a.   a vector is called an anti-symmetric real periodic n-vector if (r) and $a_k = -a_{n-k}$, for all integer k;

c.   a vector is called a complex periodic n-vector if (o) and the elements are complex;

c.h.   a vector is called a Hermitian complex periodic n-vector if (c) and $a_k = \bar{a}_{n-k}$, for all integer k;

c.a.   a vector is called an anti-Hermitian complex periodic n-vector if (c) and $a_k = -\bar{a}_{n-k}$, for all integer k.

In the operators related to the DFT we will consider the significant part of a periodic n-vector; in the same way we will talk about a

symmetric, an anti-symmetric, a Hermitian or an anti-Hermitian vector. We use the notation:

$\underline{a}$, for either a real or complex vector;

$\underline{r}$, for a real vector;

$\underline{c}$, for a complex vector.

The data representation of an element a_k of a periodic n-vector is the number $a[LWB\ a + k\ MOD\ n]$, with $LWB\ a$ a parameter free for choice.

3. Fourier Analysis and Synthesis

For the definition of DFT and IDFT and related terminology we adhere to [6]. For simplicity we restrict ourselves to periodic functions on $[0,T)$, and trigonometric functions as basis functions.

Within this context relation (1.1) reads

$$(3.1) \qquad s(t) = \sum_{k=-\infty}^{\infty} \sigma_k\ e^{2\pi ikt/T}, \qquad t \in [0,T)$$

with the so-called Fourier coefficients

$$\sigma_k = \frac{1}{T} \int_0^T s(t)\ e^{-2\pi ikt/T}\ dt, \quad k = \ldots,-n,\ldots,0,\ldots,n,\ldots$$

[11;509].

From relation (3.1) we considered two summation problems:

. Fourier analysis: given s obtain $\{\sigma_k\}$;

. Fourier synthesis: given $\{\sigma_k\}$ obtain s.

A variety of implementations may result according to the actual information used and delivered.

3.1 Fourier analysis

Generally only equidistant samples $\{s(2\pi k/n|\ k=0,1,\ldots,n-1\}$ are available and the first $\{\sigma_k|\ k=0,1,\ldots,n-1\}$ are desired.

The σ_k can be approximated by using attenuation factors τ_k as follows

$$(3.2) \qquad \sigma_k' = \tau_k * \sum_{j=0}^{n-1} s(2\pi j/n)\ w_n^{-kj}, \qquad k = 0,1,\ldots,n-1$$

6

with

$$W_n^{kj} = \exp(2\pi ikj/n)$$

and where the bar denotes complex conjugation and the prime the approximated value. The second factor of (3.2) apart from a factor is known as the Discrete Fourier Transform (DFT), [10;573].

For τ_k we may use a variety of possibilities, e.g.

τ_k	used information
1	$s(2\pi k/n)$, $k = 0,1,\ldots,n-1$.
$(\dfrac{\sin\ \pi k/n}{\pi k/n})^2$	$s(2\pi k/n)$, $k = 0,1,\ldots,n-1$, and function between samples is linear; σ_k' are the Fourier coefficients of the linear interpolated samples.

For more information about the attenuation factors, see [4].

Remarks

. The DFT can be handled by the monadic operators *C06EXPSUM* etc.

. The values of $\sin(2\pi k/n)$, $k = 0,1,\ldots,n-1$, can be obtained in $s[0:n-1]$ by the call of the technical routine *C06SINTWI* as follows

     C06SINTWI (n,s).

3.2 Fourier synthesis

Generally the first $\{\sigma_k \mid k=0,1,\ldots,n-1\}$ and either one argument t or n equidistant arguments $\{t_k \mid t_k = kT/n,\ k=0,1,\ldots,n-1\}$ are available while $s(t)$ or $\{s(t_k)\}$ are desired.

This reduces to the problems:

. the evaluation of a trigonometric sum

$$\sum_{k=0}^{n-1} \sigma_k\ e^{2\pi ikt/T}$$

. the IDFT (the Inverse Discrete Fourier Transform)

$$\sum_{k=0}^{n-1} \sigma_k \, W_n^{kj}, \qquad j = 0,1,\ldots,n-1.$$

3.3 Applications

A collection of papers on the application of the DFT within the context of digital signal processing is given in [9,10]; with respect to time series analysis a concise survey of the backgrounds and a discussion of the application of the DFT to the calculation of sample covariance and cross-covariance functions, to the estimation of variance spectra and cross-spectra and very briefly to the implementation of moving average digital filters is given in [3]. In [6] the DFT is discussed and the following computational problems in (mainly) complex analysis where it can be fruitfully applied, are suggested: calculation of Fourier coefficients using attenuation factors; solution of Symm's integral equation in conformal mapping; trigonometric interpolation; determination of conjugate periodic functions and their application to Theodorsen's integral equation for the conformal mapping of simply and of doubly connected regions; determination of Laurent's coefficients with applications to numerical differentiation; generating functions and the numerical inversion of Laplace transforms; determination of the density of the zeros of high degree polynomials; convolution and its application to time series analysis, to the multiplication of polynomials and of large integers, and to fast Poisson solvers; manipulation of power series.

As an example we consider the evaluation of

$$\Phi(\omega) = \int_{-\infty}^{\infty} s(t) e^{-2\pi i \omega t} dt.$$

For the samples $\{\phi(j/T) \mid j=0,\pm1,\pm2,\ldots\}$

the above integral reduces to

$$(3.3) \qquad \Phi(j/T) = \int_{0}^{T} s_p(t) e^{-2\pi i j t/T} dt.$$

with the periodic alias function

$$s_p(t) = \sum_{k=-\infty}^{\infty} s(t+kT)$$

and T a free parameter. The integrals (3.3) can be evaluated by (3.2).

Remarks

. $s_p(t)$ is generally approximated by s(t), $t \in [-T/2,T/2]$ with T such that the integration beyond $[-T/2,T/2]$ is negligible.

. With bandlimited functions we know already that $\Phi(\omega) = 0$ for $|\omega| > \Omega$.

This results in

$$-\Omega < j/T < \Omega, \quad j=0,\pm1,\ldots,\pm(n-1)$$

with $T = n\Delta t$ and Δt the sampling distance.

From a given $\Delta t \leq 1/(2\Omega)$ we can choose either the size n of the DFT or T. The behaviour of s and the desired accuracy of the approximation of s_p determines the total number of samples.

3.4 Implementation survey

For the moment only operators related to Fourier analysis and Fourier synthesis are available. These are grouped in four problem sets:

. evaluation of IDFT (monadic *C06EXPSUM* etc.):

$$\sum_{k=0}^{n-1} a_k e^{2\pi ikj/n}, \quad j = 0,1,\ldots,n-1;$$

. evaluation of exponential sums (dyadic *C06EXPSUM* etc.):

$$f(\theta,\underline{a}) = \sum_{k=0}^{n} a_k z^k, \quad z = e^{i\theta};$$

. evaluation of Discrete Harmonic Analysis (DHA) and Discrete Harmonic Synthesis (DHS) (monadic *C06TRGSUM* etc.):

from

$$f_j = \sum_{k=0}^{m} {}'' a_j \cos \pi kj/m + b_j \sin \pi kj/m, \quad j=0,1,\ldots,n-1 \text{ and } n=2*m$$

then the problems are characterized by

DHS: obtain $\underline{f}$ from $\underline{a}$ and $\underline{b}$

DHA: obtain $\underline{a}$ and $\underline{b}$ from f;

. evaluation of trigonometric sums (dyadic *C06TRGSUM* etc.):

$$f(\theta,\underline{a},\underline{b}) = \sum_{k=0}^{m} a_k \cos k\theta + b_k \sin k\theta.$$

Each problem set gives rise to a number of operators according to the data type and symmetries in the data: each problem set, and the details of the involved operators, is separately described in a documentation unit.

REFERENCES

[1] COCHRAN, W.T. c.s. (G-AE subcommittee on measurement concepts)
   What is the Fast Fourier Transform?
   IEEE Trans. Audio-Electroacoust., AU-15, 45-55. 1967.
   Reprinted in RABINER L.R.c.s. [9,240-250].

[2] COOLEY, J.W. c.s.
   The Fast Fourier Transform Algorithm: programming considerations in the calculation of sine, cosine and Laplace transforms.
   J. Sound. Vib, 12, 315-337. 1970.
   Reprinted in RABINER L.R. c.s. [9,271-293].

[3] COOLEY, J.W. c.s.
   The FFT and its application to time series analysis.
   In: ENSLEIN, K. c.s.
      Statistical methods for digital computers, 377-423, 1977.

10

[4]   GAUTSCHI, W.

      Attenuation factors in practical Fourier Analysis.

      Numer. Math., 18, 373-400. 1972.

[5]   GENTLEMAN, M.W. & SANDE, G.

      Fast Fourier Transforms for fun and profit.

      AFIPS, 29, 563-578. 1966.

[6]   HENRICI, P.

      Fast Fourier methods in computational complex analysis.

      SIAM Review, 21, 4, 481-527. 1979.

[7]   OLIVER, J.

      Stable methods for evaluating the points $\cos(i\pi/n)$.

      JIMA, 16, 247-257. 1975.

[8]   RAMOS, G.

      Roundoff error analysis of the Fast Fourier Transform.

      Math. Comp., 25, 757-768. 1971.

[9]   RABINER, L.R. c.s.

      Digital signal processing I.

      1972.

      IEEE-press.

[10] RABINER, L.R. c.s.

      Digital signal processing II.

      1975.

      IEEE-press.

[11] HAMMING, R.W.

      Numerical methods for scientists and engineers. Second edition.

      1973.

II. DOCUMENTATION AND SOURCE TEXTS OF THE OPERATORS

II.1 *CO6EXPSUM, CO6EXPSUMHRM, CO6EXPSUMANH.*

1. Purpose

The monadic operators

       CO6EXPSUM, CO6EXPSUMHRM, CO6EXPSUMANH

evaluate the Inverse Discrete Fourier Transform (IDFT) of a real or complex vector. Advantage has been taken of symmetry in the data in

       CO6EXPSUMHRM - data vector is Hermitian symmetric -

       CO6EXPSUMANH - data vector is skew-Hermitian symmetric.

IMPORTANT:

2. Specification (Algol 68)

```
MODE SCAL = REAL, COSCAL = COMPL;
MODE VEC        = REF[ ] SCAL,
    COVEC       = REF[ ] COSCAL;
OP CO6EXPSUM    = (VEC r)     COVEC:
OP CO6EXPSUM    = (COVEC c)   COVEC:
OP CO6EXPSUMHRM = (VEC r)     VEC  :
OP CO6EXPSUMHRM = (COVEC c)   VEC  :
OP CO6EXPSUMANH = (VEC r)     VEC  :
OP CO6EXPSUMANH = (COVEC c)   VEC  :
```

3. Description

The operators calculate

$$\sum_{k=0}^{n-1} a_k \, e^{2\pi i k j/n}, \qquad j=0,1\ldots,n-1$$

with $\underline{a}$ a real or complex vector. Used is the Cooley-Sande-Stockham algorithm; the auxiliary twiddle-factors are calculated by an extension of the Hopgood-Litherland algorithm.

4. References

See chapter introduction [1,2,7]

12

5. Parameters

General:

. n is the size of an array; for n is even we use m to denote n/2.

. The lower bound of the result equals the lower bound of the operand .

. Only the size of the operands matters: the k-th element of a vector
  a is assumed to be a[LWB a + k], so the lower bound of the data
  representation of the vector does not matter and is free for choice.

. The operands are not preserved.

Formula	Operand	Result
C06EXPSUM a	a real or complex array variable with n elements.	a complex array variable with n elements.
C06EXPSUMHRM r	a real array variable with m+1 elements: the first elements of a symmetric n-vector, with n=2*m.	a real array variable with m+1 elements: the first elements of the resulting symmetric n-vector.
C06EXPSUMHRM c	a complex array variable with m+1 elements: the first elements of a Hermitian n-vector, with n=2*m.	a real array variable with n elements.
C06EXPSUMANH r	a real array variable with m+1 elements: the first elements of an anti-symmetric n-vector (the first and last element must contain zero), with n=2*m.	a real array variable with m+1 elements: the first elements of the imaginary part of the resulting complex Hermitian n-vector with zero real part. (The first and last element contain zero).
C06EXPSUMANH c	a complex array variable with m+1 elements: the first elements of a skew-Hermitian n-vector (the real parts of the first and last element must contain zero), with n=2*m.	a real array variable with n elements: the imaginary part of the resulting complex vector with zero real part.

6. Error indicators

In the event of an error condition being detected, the error routine:
c06fail of mode REF NAGFAIL, is called with the parameters listed below.
These are printed and in case the value of c06fail is nagsoft the

execution is continued (see in Introduction of the NAG manual the document on the ALGOL 68 error mechanism). The operators were designed with *nagsoft* as the user-friendly error-handling mechanism in mind.

parameter   message

1   OPERAND OF <operator name> OF WRONG SIZE

                           The given array is of too small size, so the IDFT is
                           an empty sum; the result yielded is the operand.

2   OPERAND OF <operator name> NOT <symmetry kind>

                           The given part of the symmetric array is not of the
                           symmetry kind expected by the operator, because of
                           the symmetry and periodicity; the calculation is
                           performed with an operand adapted to the operator,
                           by setting elements to zero where appropriate.

7. Auxiliary routines

The used NAG library operators - all with a complex operand - are given in the following table.

Formula	Used NAG library operators	Used Torrix operators/ generators
CO6EXPSUM r	none	*gencoarray*1, *widen, conj*
CO6EXPSUM c	none	none
CO6EXPSUMHRM r	m is odd: *CO6EXPSUM*	*genarray*1, *gencoarray*1, *widen, conj*
	m is even: *CO6EXPSUMHRM*	*genarray*1, *gencoarray*1, *widen, conj*
CO6EXPSUMHRM c	none	*genarray*1, *gencoarray*1, *widen, conj*
CO6EXPSUMANH r	m is odd: *CO6EXPSUM*	*genarray*1, *gencoarray*1, *widen, conj*
	m is even: *CO6EXPSUMHRM*	*genarray*1, *gencoarray*1, *widen*
CO6EXPSUMANH c	*CO6EXPSUMHRM*	*genarray*1, -, *<, *widen*

8. Timing

The time taken is proportional to p*n, where p is the sum of the prime factors of n.

14

9. Storage

The storage required by internally declared arrays is given in the following table.

Formula	Specification situation	Internally declared arrays
CO6EXPSUM r	n is even	n+n÷2+1 complex elements.
	n is odd	n elements.
CO6EXPSUM c		none.
CO6EXPSUMHRM r	m is even	m÷2+1 real and complex elements.
	m is odd	m÷2+1 real and m complex elements.
CO6EXPSUMHRM c		n real and m÷2+1 complex elements.
CO6EXPSUMANH r	m is even	m÷2+1 real and complex elements.
	m is odd	m+1 real and m÷2+1 complex elements.
CO6EXPSUMANH c		none.

10. Accuracy

Let us denote by:

. $\underline{a}$: the coefficient vector;

. a : the machine representation of the (measured) $\underline{a}$;

. Δa : $a - a$;

. g_a : the growth factor of order $\sim \sum_{i=1}^{j} p_i^{1.5}$

with $n = \prod_{i=1}^{j} p_i$ (given implicitly in [8]).

Error bounds (first order)

Formula	Propagated error	Generated error
CO6EXPSUM a	$\|\Delta\underline{a}\|_2/\max(1/\sqrt{n},\ \|\underline{a}\|_2)$	$g_a *$ small real

The bounds for CO6EXPSUMHRM, and CO6EXPSUMANH are similar.

11. Further comments

The related problems - the DFT apart from a factor 1/n -

$$\sum_{k=0}^{n-1} a_k \, e^{-2\pi i \, kj/n}$$

can be obtained as follows, where the parameters are prescribed as in 5. Parameters.

Operand	Formula
complex n-vector	*CONJ C06EXPSUM CONJ c*
Hermitian n-vector	*C06EXPSUMHRM CONJ* c
skew-Hermitian n-vector	− *C06EXPSUMANH CONJ* c
real n-vector	*CONJ C06EXPSUM* r
symmetric real n-vector	*C06EXPSUMHRM* r
anti-symmetric real n-vector	− *C06EXPSUMANH* r

12. Keywords

Discrete Fourier Transform.

Fast Fourier Transform.

Cooley-Sande-Stockham algorithm.

13. Examples

C06EXPSUM

13.1 Program text.

```
'BEGIN'
    #
    AS AN ILLUSTRATION OF THE USE OF 'C06EXPSUM'
    THIS PROGRAM CALCULATES APPROXIMATELY THE IDFT OF:
    - A COMPLEX VECTOR;
    - A REAL VECTOR.
    #
    'COVEC'C=GENCOVEC(2):=(1.0'I'2.0,3.0'I'4.0);
    WRITEF(($28A,2(L,16A,2(L,-D.DDQI-D.DD))$,
            "'C06EXPSUM' EXAMPLE PROGRAM.",
            "COMPLEX OPERAND:",
            C,
            "COMPLEX RESULT:",
            'C06EXPSUM'C));
```

```
'VEC'R=GENVEC(4):=(1.0,0.0,3.0,4.0);
WRITEF(($2L,13A,4(Q-D.DD),L,15A,4(L,-D.DDQI-D.DD)$,
        "REAL OPERAND:",
        R,
        "COMPLEX RESULT:",
        'CO6EXPSUM'R))
```

'END'#OF 'CO6EXPSUM' EXAMPLE PROGRAM#

13.2 Data for program. None.

13.3 Results.

'CO6EXPSUM' EXAMPLE PROGRAM..

COMPLEX OPERAND:
 1.00 I 2.00
 3.00 I 4.00
COMPLEX RESULT:
 4.00 I 6.00
-2.00 I-2.00

REAL OPERAND: 1.00 0.00 3.00 4.00
COMPLEX RESULT:
 8.00 I 0.00
-2.00 I-4.00
 0.00 I 0.00
-2.00 I 4.00

CO6EXPSUMHRM

13.1 Program text.

```
'BEGIN'
    #
    AS AN ILLUSTRATION OF THE USE OF 'CO6EXPSUMHRM' THIS PROGRAM
    CALCULATES APPROXIMATELY THE IDFT OF:
    - A COMPLEX HERMITIAN VECTOR;
    - A REAL SYMMETRIC VECTOR.
    #
    'COVEC'C=GENCOVEC(3):=(I.0'I'0.0,0.0'I'1.0,2.0'I'0.0);
    WRITEF(($31A,L,54A,3(L,-D.DDQI-D.DD),L,22A,4(Q-D.DDQ)$,
            "'CO6EXPSUMHRM' EXAMPLE PROGRAM.",
            "SIGNIFICANT PART OF COMPLEX HERMITIAN VECTOR ON INPUT:",
            C,
            "REAL VECTOR DELIVERED:",
            'CO6EXPSUMHRM'C));
```

```
    'VEC'R=GENVEC(3):=(1.0,2.0,3.0);
    WRITEF(($2L,51A,3(Q-D.DD),L,42A,3(Q-D.DD)$,
            "SIGNIFICANT PART OF REAL SYMMETRIC VECTOR ON INPUT:",
            R,
            "SIGNIFICANT PART OF REAL SYMMETRIC RESULT:",
            'C06EXPSUMHRM'R))

    'END'#OF 'C06EXPSUMHRM' EXAMPLE PROGRAM#
```

13.2 Data for program. None.

13.3 Results.

```
'C06EXPSUMHRM' EXAMPLE PROGRAM.
SIGNIFICANT PART OF COMPLEX HERMITIAN VECTOR ON INPUT:
 1.00 I 0.00
 0.00 I 1.00
 2.00 I 0.00
REAL VECTOR DELIVERED: 3.00 -3.00 3.00 1.00

SIGNIFICANT PART OF REAL SYMMETRIC VECTOR ON INPUT: 1.00 2.00 3.00
SIGNIFICANT PART OF REAL SYMMETRIC RESULT: 8.00 -2.00 0.00
```

C06EXPSUMANH

13.1 Program text.

```
'BEGIN'
    #
    AS AN ILLUSTRATION OF THE USE OF 'C06EXPSUMANH' THIS PROGRAM
    CALCULATES APPROXIMATELY THE IDFT OF:
    - A COMPLEX ANTI-HERMITIAN VECTOR;
    - A REAL ANTI-SYMMETRIC VECTOR.
    #
    'COVEC'C=GENCOVEC(3):=(0.0'I'3.0,0.0'I'1.0,0.0'I'2.0);
    WRITEF(($31A,L,59A,3(L,-D.DDQI-D.DD),L,25A,4(Q-D.DD)$,
            "'C06EXPSUMANH' EXAMPLE PROGRAM.",
            "SIGNIFICANT PART OF COMPLEX ANTI-HERMITIAN VECTOR ON INPUT:",
            C,
            "DELIVERED IMAGINARY PART:",
            'C06EXPSUMANH'C));
'VEC'R=GENVEC(3):=(0.0,2.0,0.0);
WRITEF(($2L,56A,3(Q-D.DD),L,37A,3(Q-D.DD)$,
        "SIGNIFICANT PART OF REAL ANTI-SYMMETRIC VECTOR ON INPUT:",
        R,
        "SIGNIFICANT PART OF IMAGINARY RESULT:",
        'C06EXPSUMANH'R))

'END'#OF 'C06EXPSUMANH' EXAMPLE PROGRAM#
```

13.2 Data for program. None.

13.3 Results.

'CO6EXPSUMANH' EXAMPLE PROGRAM.
SIGNIFICANT PART OF COMPLEX ANTI-HERMITIAN VECTOR ON INPUT:
 0.00 I 3.00
 0.00 I 1.00
 0.00 I 2.00
DELIVERED IMAGINARY PART: 7.00 1.00 3.00 1.00

SIGNIFICANT PART OF REAL ANTI-SYMMETRIC VECTOR ON INPUT: 0.00 2.00 0.00
SIGNIFICANT PART OF IMAGINARY RESULT: 0.00 4.00 0.00

14. Source texts

OP CO6EXPSUM =(COVEC c) COVEC :

#purpose: approximately calculated is

                   $n-1$
                   sum w[j,k]×c[k+ LWB c], j=0,1, ... ,n-1
                   k=0
                   with n= SIZE c,w[j,k]=exp(0 I j×k×2×pi/n),
input: see above formula. c is not preserved.
results: the idft of c is delivered
                   with bounds similar to those of c.
exception handling:if SIZE c<1 then c is delivered and cO6fail is
                   called.#

IF INT n= SIZE c;n>1
THEN cO6fft(c[AT 0]);c
ELSE cO6fail(1,"expsumoperand of CO6EXPSUM of wrong size");c

FI ,

```
OP  CO6EXPSUM =( VEC r) COVEC :

#purpose:              approximately calculated is

                       n-1
                       sum w[j,k]×r[k+ LWB r],  j=0,1, ... ,n-1
                       k=0

                       with  n= SIZE r, w[j,k]=exp(0 I 2×pi×j×k/n).
input:                 see above formula. r is not preserved.
results:               the idft of (real) r with similar bounds as r.
exception handling:if  SIZE r<1 then r, widened to a  COVEC , is
                       delivered and c06fail is called.#


IF  INT n= SIZE r;n>0
THEN  INT l= LWB r,u= UPB r;
    IF  ODD n
    THEN  COVEC xy= WIDEN r[ AT 0];
       c06fft(xy);
       INT uu:=n-1;
       FOR 11 WHILE 11<uu
       DO  REF  COSCAL xy1=xy[11],xyu=xy[uu];
           COSCAL rs=(xy1+ CONJ xyu)/ WIDEN 2;
           (xy1:=rs,xyu:= CONJ rs);
           uu-:=1
       OD ;
       xy[ AT 1]
    ELSE  INT n2=n OVER 2,n4=n OVER 4, INT j:=-1;
       COVEC xy=gencoarray1(0,n-1);
       FOR i FROM 1 BY 2 TO u-1
       DO xy[j+:=1]:=r[i] I r[i+1] OD ;
       c06fft(xy[0:n2-1 AT 0]);
       BEGIN  SCAL normdiv2= WIDEN 1/ WIDEN 2;

           COVEC wn=gencoarray1(0,n4);
           c06initw(n,wn);
           FOR i TO n4
           DO  REF  COSCAL s=xy[i],t=xy[n2-i],
              COSCAL wni=wn[i];
              COSCAL p=(s+ CONJ t)×normdiv2,
              q=(s- CONJ t)×(im OF wni I -re OF wni)×normdiv2;
              (s:=p+q,t:= CONJ (p-q))
           OD ;
           ( SCAL s=re OF xy[0],t=im OF xy[0];
           (xy[0]:= WIDEN (s+t),xy[n2]:= WIDEN ( s-t)))
       END ;
       INT uu:=n-1;
       FOR 11 WHILE 11<uu
       DO xy[uu]:= CONJ xy[11];uu-:=1 OD ;
       xy[ AT 1]
    FI
ELSE c06fail(1,"expsumoperand of  CO6EXPSUM  of wrong size");
    WIDEN r
FI ,
```

OP   CO6EXPSUMHRM =(COVEC c) VEC :

#purpose: approximately calculated is

$$\sum_{k=0}^{n-1} w[j,k] \times c[\; LWB\; c+k], \quad j=0,1, \ldots ,n-1$$

with n=n2×2, the size of the complete hermitian
vector c, (i.e. c[LWB c+k]= CONJ c[LWB c+n-k],
k=1,2, ... , n-1) and w[j,k]=exp(0 I 2×pi×j×k/n).

input: c[LWB c: LWB c+n2] is assumed to be supplied.
                   c is not preserved. for odd n use CO6EXPSUM .

results: the idft of c is a real vector and is delivered
                   as a VEC with the lower bound similar to that of
                   c and with LWB c+n-1 as upper bound.

exception handling:if the first or last element of the given c have
                   nonzero imaginairy parts (so the complete vector
                   is not hermitian symmetric) then the calculation is
                   done with these parts put to zero and c06fail is
                   called. if n<2 then a VEC with bounds LWB c and
                   LWB c+n-1 is delivered and c06fail is called.#

```
IF   INT l= LWB c,u= UPB c; INT n2=u-1; INT n=n2×2;n2>1
THEN  IF  SCAL zero= WIDEN 0;
      im OF c[l]=zero AND im OF c[u]=zero
   THEN  INT n4=n2 OVER 2;
      COVEC czer=c[1:1+n2-1 AT 0];
      COVEC wn=gencoarrayl(0,n4);
      c06initw(n,wn);
      czer[0]:=( SCAL c0=re OF czer[0],cn2=re OF c[u];
      (c0+cn2) I (c0-cn2));
      FOR k TO n4
      DO  REF  COSCAL ck=czer[k],cn2k=czer[n2-k];
          COSCAL s= ck+ CONJ cn2k,
                 t=(ck- CONJ cn2k)×wn[k]×(0 I 1);
          (ck:=s+t,cn2k:= CONJ (s-t))
      OD ;
      c06fft(czer);
      VEC result=genarrayl(0,n-1);
      INT j:=-1;
      FOR k BY 2 TO n-1
      DO result[k-1]:=re OF czer[j+:=1];
         result[k  ]:=im OF czer[j]
      OD ;
      result[ AT 1]
   ELSE c06fail(2,"exphrmoperand of  CO6EXPSUMHRM  not hermitian");
      im OF c[l]:=im OF c[u]:=zero; CO6EXPSUMHRM c
   FI
ELSE c06fail(1,"exphrmoperand of  CO6EXPSUMHRM  of wrong size");
   genarrayl(1,1+n-1)
FI ,
```

```
OP  CO6EXPSUMHRM =( VEC r) VEC :

#purpose:          approximately calculated is

                   n-1
                   sum w[j,k]×r[ LWB r+k], j=0,1, ... ,n-1
                   k=0

                   with n=n2×2, the size of the complete symmetric
                   vector r, (i.e. r[ LWB r+k]=r[ LWB r+n-k],
                   k=1,2, ... ,n-1) and w[j,k]=exp(0 I 2×pi×j×k/n).
input:             r[ LWB r: LWB r+n2] is assumed to be supplied.
                   r is not preserved. for odd n use  CO6EXPSUM .
results:           the dft of r is again real and symmetric, so only
                   the first n2+1 elements are delivered with bounds
                   similar to those of r.
exception handling:if n<2 then cO6fail is called and the original
                   vector is delivered.#

IF  INT n2= SIZE r-1;n2>0
THEN  SCAL oddsum:= WIDEN 0, VEC rzer=r[ AT 0];
   FOR k BY 2 TO n2-1
   DO oddsum+:=2×rzer[k] OD ;
   IF  ODD n2
   THEN oddsum+:=rzer[n2];
      COVEC c=gencoarrayl(0,n2-1);
      INT j:=0;c[0]:= WIDEN rzer[0];
      FOR k FROM 2 BY 2 TO n2
      DO  COSCAL cs=c[j+:=1]:=rzer[k] I (rzer[k+1]-rzer[k-1]);
         c[n2-j]:= CONJ cs
      OD ;
      rzer[0:n2-1 AT 0]:=re OF ( CO6EXPSUM c)
   ELSE  INT n4=n2 OVER 2; COVEC c=gencoarrayl(0,n4);
      INT j:=0;c[0]:= WIDEN rzer[0];
      FOR k FROM 2 BY 2 TO n2-2
      DO c[j+:=1]:=rzer[k] I (rzer[k+1]-rzer[k-1]) OD ;
      c[n4]:= WIDEN rzer[n2];
      rzer[0:n2-1 AT 0]:= CO6EXPSUMHRM c
   FI ;
   SCAL evensum=rzer[0];
   (rzer[0]+:=oddsum,rzer[n2]:=evensum-oddsum);
   INT m=n2 OVER 2;
   VEC sintwi=genarrayl(0,m);
   cO6sintwi(2×n2,sintwi);
   INT uu:=n2;
   SCAL two= WIDEN 2;
   FOR k TO m
   DO  REF  SCAL s=rzer[k],t=rzer[uu-:=1];
      SCAL sk=s+t,tk=(s-t)/(two×sintwi[k]);
      (s:=(sk+tk)/two,t:=(sk-tk)/two)
   OD ;
   r
ELSE cO6fail(1,"exphrmoperand of  CO6EXPSUMHRM  of wrong size");r
FI ,
```

OP CO6EXPSUMANH =(COVEC c) VEC :

#purpose: approximately calculated is

                      n-1
                      sum w[j,k]×c[LWB c+k], j=0,1, ... ,n-1
                      k=0

                      with n=n2×2, the size of the complete anti-
                      hermitian vector c,
                      (i.e. c[LWB c+k]=- CONJ c[LWB c+n-k],
                      k=1,2, ... ,n-1) and w[j,k]=exp(0 I 2×pi×j×k/n).
input: c[LWB c: LWB c+n2] is assumed to be supplied.
                      c is not preserved. for odd n use CO6EXPSUM .
results: the dft of c is a vector with a zero real part.
                      the imaginary part is delivered as a VEC with the
                      same lower bound as c and with LWB c+n-1 as upper
                      bound.
exception handling:if the first or last element of the given c have
                      non zero real parts (so the complete vector is not
                      anti-hermitian symmetric) then the calculation is
                      done with these parts put to zero and c06fail is
                      called. if n<2 then a VEC with bounds LWB c and
                      LWB c+n-1 is delivered and c06fail is called.#

IF INT 1= LWB c,u= UPB c; INT n=(u-1)×2;n>1
THEN IF   SCAL zero= WIDEN 0;
      re OF c[1]=zero AND re OF c[u]=zero
   THEN CO6EXPSUMHRM (-c×<(0 I 1))
   ELSE c06fail(2,
      "expanhoperand of CO6EXPSUMANH not anti-hermitian");
      re OF c[1]:=re OF c[u]:=zero;
      CO6EXPSUMHRM (-c×<(0 I 1))
   FI
ELSE c06fail(1,"expanhoperand of CO6EXPSUMANH of wrong size");
   genarray1(1,1+n-1)
FI ,

OP CO6EXPSUMANH =(VEC r) VEC :

#purpose: approximately calculated is

                      n-1
                      sum w[j,k]×r[LWB r+k], j=0,1, ... ,n-1
                      k=0

                      with n=n2×2, the size of the complete anti-
                      symmetric vector r,
                      (i.e. r[LWB r+k]=-r[LWB r+n-k], k=1,2, ... , n-1)
                      and w[j,k]=exp(0 I 2×pi×j×k/n).

```
input:              r[ LWB r: LWB r+n2] is assumed to be supplied.
                    r is not preserved. for odd n use  CO6EXPSUM .
results:            the dft of r is a complex hermitian vector with a
                    zero real part.
                    only the first n2+1 elements of the imaginary part
                    are delivered as a vec, with bounds similar to
                    those of r.
                    note that the first and the last element of the
                    delivered vec are zero.
exception handling:if the first and the last elements of the given r
                    are not zero then the calculation is done with
                    these parts put to zero and c06fail is called. if
                    n<2 then c06fail is called and the original vector
                    is delivered.#


IF   INT l= LWB r,u= UPB r; INT n2=u-1;n2>0
THEN  IF  SCAL zero= WIDEN 0;
       r[l]=zero AND r[u]=zero
    THEN  VEC rzer=r[ AT 0]; SCAL two= WIDEN 2;
       IF  ODD n2
       THEN  COVEC c=gencoarrayl(0,n2-1);
           INT j:=0;c[0]:= WIDEN (two×rzer[l]);
           FOR k FROM 2 BY 2 TO n2
           DO  COSCAL cs=c[ j+:=1]:=(rzer[k+1]-rzer[k-1]) I rzer[k];
                 c[n2-j]:= CONJ cs
           OD ;
           rzer[0:n2-1 AT 0]:=re OF ( CO6EXPSUM c)
       ELSE  INT n4=n2 OVER 2; COVEC c=gencoarrayl(0,n4);
           INT j:=0;c[0]:= WIDEN (two×rzer[l]);
           FOR k FROM 2 BY 2 TO n2-2
           DO c[ j+:=1]:=(rzer[k+1]-rzer[k-1]) I rzer[k] OD ;
           c[n4]:= WIDEN (-two×rzer[n2-1]);
           rzer[0:n2-1 AT 0]:= CO6EXPSUMHRM c
       FI ;
       INT m=n2 OVER 2;
       VEC sintwi=genarrayl(0,m);
       c06sintwi(2×n2,sintwi);
       INT uu:=n2;
       FOR k TO m
       DO  REF  SCAL s=rzer[k],t=rzer[uu-:=1];
           SCAL sk=t-s,tk=(t+s)/(two×sintwi[k]);
           (s:=(sk+tk)/two,t:=(tk-sk)/two)
       OD ;
       rzer[0]:=rzer[n2]:=zero;
       r
    ELSE c06fail(2,
           "expanhoperand of  CO6EXPSUMANH  not anti-symmetric");
       r[l]:=r[u]:=zero; CO6EXPSUMANH r
   FI
ELSE c06fail(1,"expanhoperand in  CO6EXPSUMANH  of wrong size");r
FI
```

II.2 *C06EXPSUM* (dyadic)

1. Purpose

The dyadic operators

    C06EXPSUM

evaluate an exponential sum with real or complex coefficients.

IMPORTANT:....

2. Specification (Algol 68)

    MODE SCAL = REAL, COSCAL = COMPL;
    MODE VEC   = REF[ ]SCAL,
        COVEC   = REF[ ]COSCAL;
    OP C06EXPSUM   = (SCAL t, VEC r)   COSCAL:
    OP C06EXPSUM   = (SCAL t, COVEC c) COSCAL:
    PRIO C06EXPSUM   = 8.

3. Description

The operators calculate

$$f(\Theta, \underline{a}) = \sum_{k=0}^{n} a_k z^k, \quad z=e^{i\Theta},$$

with $\underline{a}$ a real or complex vector. The problem is reduced to the problem of evaluating trigonometric sums - i.e. sine and cosine sums - by considering the real and imaginary parts. Used is the Clenshaw algorithm with the modifications due to Reinsch.

4. References

[1] OLIVER, J.

An error analysis of the modified Clenshaw method for evaluating Chebyshev and Fourier series.

JIMA, vol. 20, 379-391. 1977.

5. Parameters

General:

. Both operands are preserved.

. The result is a complex constant: the exponential sum.

. Only the size of the right operand matters: the k-th element of
a vector $\underline{a}$ is assumed to be represented by $a[LWB\ a\ +\ k]$ so the lower
bound of the data representation of the vector does not matter and is
free for choice.

Left operand t: the angle θ, a real constant (it is advised to supply a
value within $[-\pi,\pi)$).

Right operand : a real or complex array with n+1 elements.

6. Error indicators

In the event of an error condition being detected, the error routine:
$c06fail$ of mode $REF\ NAGFAIL$, is called with the parameters listed below.
These are printed and in case the value of $c06fail$ is $nagsoft$ the execution
is continued (see in Introduction of the NAG manual the document on the
ALGOL 68 error mechanism). The operators were designed with $nagsoft$ as
the user-friendly error-handling mechanism in mind.

parameter	message
1	VECTOR OPERAND OF $C06EXPSUM$ OF WRONG SIZE
	The size of the given array is smaller than zero; the result yielded is zero as value for the empty sum.

7. Auxiliary routines   None.

8. Timing

The time taken is proportional to n.

9. Storage   No auxiliary arrays are declared.

26

10. Accuracy

Let us denote by:

- **a** : the coefficient vector;

- θ : the angle;

- c_a : the condition number $\left(\sum\limits_{k=0}^{n} \max(1, |a_k|)\right) / \max(1, |f(\theta, \underline{a})|)$;

- c_θ : the condition number $\left| \max(1, |\theta|) \dfrac{\partial f}{\partial \theta} \right| / \max(1, |f(\theta, \underline{a})|)$;

- g_a : the growth factor (conjectured of order n);

- a : the machine representation of (the measured) $\underline{a}$;

- t : the machine representation of (the measured) θ;

- $\underline{\delta a}$: vector of componentswise errors:

$$\delta a_k = |a_k - a[LWB\ a + k]| / \max(1, |a_k|)$$

for all appropriate k;

- $\delta\theta$: $|t - \theta| / \max(1, |\theta|)$.

Error bounds (first order)

Formula	Propagated error	Generated error
t C06EXPSUM a	$c_a * \|\underline{\delta a}\|_\infty + c_\theta * \delta\theta$	$c_a * g *$ *small real*

11. Further comments   None.

12. Keywords

Exponential sum.

Clenshaw-Reinsch algorithm.

13. Examples

13.1 Program text.

```
'BEGIN'
    #
    AS AN ILLUSTRATION OF THE USE OF 'CO6EXPSUM' (DYADIC) THE PROGRAM
    CALCULATES APPROXIMATELY

       1
    SUM  A[K]*EXP(0'I'K*T)
    K=-1

    #
    'SCAL' ANGLE=PI/2;
    'VEC'APOS=(GENVEC(2):=(0.5,2.0))['AT'0],
         ANEG=(GENVEC(2):=(0.5,2.0))['AT'0];
    WRITEF(($28A,L,6A,Q-D.DD,L,34A,3(Q-D.DD),L,25A,Q-D.DDQI-D.DD$,
            "'CO6EXPSUM' EXAMPLE PROGRAM.",
            "ANGLE:",ANGLE,
            "COEFFICIENTS A[-1],A[0] AND A[1]:",
            ANEG[1],ANEG[0]+APOS[0],APOS[1],
            "VALUE OF EXPONENTIAL SUM:",
            -ANGLE'CO6EXPSUM'ANEG+ANGLE'CO6EXPSUM'APOS));

    'COVEC'APLS=(GENCOVEC(2):=(0.5'I'0.25,0.5'I'-0.5))['AT'0],
           AMIN=(GENCOVEC(2):=(0.5'I'0.25,0.5'I'-0.5))['AT'0];
    WRITEF(($2L,6A,Q-D.DD,L,34A,3(L,-D.DDQI-D.DD),L,25A,Q-D.DDQI-D.DD$,
            "ANGLE:",ANGLE,
            "COEFFICIENTS A[-1], A[0] AND A[1]:",
            AMIN[1],AMIN[0]+APLS[0],APLS[1],
            "VALUE OF EXPONENTIAL SUM:",
            -ANGLE'CO6EXPSUM'AMIN+ANGLE'CO6EXPSUM'APLS))
'END'#OF 'CO6EXPSUM' EXAMPLE PROGRAM#
```

13.2 Data for program. None.

13.3 Results.

```
'CO6EXPSUM' EXAMPLE PROGRAM.
ANGLE:  1.57
COEFFICIENTS A[-1], A[0] AND A[1]: 2.00 1.00 2.00
VALUE OF EXPONENTIAL SUM: 1.00 I 0.00

ANGLE:  1.57
COEFFICIENTS A[-1], A[0] AND A[1]:
 0.50 I-0.50
 1.00 I 0.50
 0.50 I-0.50
VALUE OF EXPONENTIAL SUM:  1.00 I 0.50
```

14. Source texts

```
OP    CO6EXPSUM =( SCAL t, COVEC c) COSCAL :

#purpose:              approximately calculated is the exponential sum

                       n
                       sum c[ LWB c+k]×z↑k
                       k=0

                       with z=exp(0 I t) and n= SIZE c-1.
input:                 left operand: angle t of  MODE    SCAL ; it is
                       advised to take t in the interval [-pi,pi).
                       right operand: coefficient vector c of  MODE
                       COVEC .
result:                the result of the above, possibly empty, sum of
                       MODE    COSCAL .
exception handling:if n<0 then c06fail is called and zero
                       is delivered.#

IF    SIZE c>0  THEN
   SCAL   sinserr,cosserr,sinseri,cosseri;
   (c06ser( re  OF c,cosserr,sinserr,t),
   c06ser( im  OF c,cosseri,sinseri,t));
   (cosserr-sinseri) I (sinserr+cosseri)
ELSE  c06fail(1,"expsumvector operand.of c06expsum of wrong size");
   WIDEN  WIDEN 0
FI  ,




OP    CO6EXPSUM =( SCAL t, VEC r) COSCAL :

#purpose:              approximately calculated is the exponential sum

                       n
                       sum r[ LWB r+k]×z↑k
                       k=0

                       with z=exp(0 I t) and n= SIZE r-1.
input:                 left operand: angle t of  MODE    SCAL ; it is
                       advised to take t in the interval [-pi,pi).
                       right operand: coefficient vector r of  MODE
                       VEC .
result:                the result of the above, possibly empty, sum of
                       MODE    COSCAL .
exception handling:if n<0 then c06fail is called and zero
                       is delivered.#

IF    SIZE r>0  THEN
   SCAL   sinser,cosser;
   c06ser(r,cosser,sinser,t);
   cosser I sinser
ELSE  c06fail(1,"expsumvector operand of c06expsum of wrong size");
   WIDEN  WIDEN 0
FI
```

II.3 *C06TRGSUM, C06COSSUM, C06SINSUM*

1. Purpose

The monadic operators

C06TRGSUM, C06COSSUM, C06SINSUM

evaluate the Discrete Harmonic Analysis (DHA) and Discrete Harmonic

Synthesis (DHS).

Advantage has been taken of zeros in the data in

C06COSSUM - sine coefficients are zero (a Discrete Cosine

Transform (DCT))

C06SINSUM - cosine coefficients are zero (a Discrete Sine

Transform (DST)).

IMPORTANT:....

2. Specification (Algol 68)

$$MODE\ SCAL\quad = REAL,\ COSCAL = COMPL;$$

$$MODE\ VEC\quad = REF[\ ]\ SCAL,$$

$$COVEC\quad = REF[\ ]COSCAL;$$

$$OP\ C06TRGSUM\ = (VEC\ r)\qquad COVEC:$$

$$OP\ C06TRGSUM\ = (COVEC\ ab)\quad VEC\quad:$$

$$OP\ C06COSSUM\ = (VEC\ a)\qquad VEC\quad:$$

$$OP\ C06SINSUM\ = (VEC\ b)\qquad VEC\quad:$$

3. Description

Given the relation

$$f_j = \sum_{k=0}^{m}{}'' (a_j * \cos(\pi*k*j/m) + b_j * \sin(\pi*k*j/m)),\ j=0,1,\ldots,n-1,\ n=2*m,$$

then the problems are characterized by:

DHS: obtain $\underline{f}$ from $\underline{a}$ and $\underline{b}$,

DHA: obtain $\underline{a}$ and $\underline{b}$ from $\underline{f}$,

DCT: obtain $\underline{f}$ from $\underline{a}$; $\underline{b}$ is zero,

DST: obtain $\underline{f}$ from $\underline{b}$; a is zero.

Used is the Cooley-Sande-Stockham algorithm for the DFT because the above relation is equivalent to

$$a_j + ib_j = 1/m \sum_{k=0}^{n-1} f_k * e^{2\pi ikj/n}, \quad j=0,1,\ldots,m.$$

4. References

See chapter introduction: [1,2].

5. Parameters

General:

. n is the size of the array of real variables f of even size; we use m to denote n/2.

. The lower bound of the result equals the lower bound of the operand.

. Only the size of the operands matters: the k-th element of a vector $\underline{c}$ is assumed to be represented by $c[LWB\ c + k]$, so the lower bound of the data representation of the vector does not matter and is free for choice.

. The operands are not preserved.

. The first and last element of b must contain zero.

Formula	Operand	Result
CO6TRGSUM f	a real array variable with n elements: $\underline{f}$.	a complex array variable with m+1 elements: the real part contains $\underline{a}$ and the imaginary part contains $\underline{b}$.
CO6TRGSUM ab	a complex array variable with m+1 elements: the real part contains $\underline{a}$ and the imaginary part contains $\underline{b}$.	a real array variable with n elements: $\underline{f}$.
CO6COSSUM a	a real array variable with m+1 elements: $\underline{a}$.	a real array variable with m+1 elements: the DCT of the operand.
CO6SINSUM b	a real array variable with m+1 elements: $\underline{b}$.	a real array variable with m+1 elements: the DST of the operand.

6. Error indicators

In the event of an error condition being detected the error routine: *c06fail* of mode *REF NAGFAIL*, is called with the parameters listed below. These are printed and in case the value of *c06fail* is *nagsoft* the execution is continued (see in Introduction of the NAG manual the document on the ALGOL 68 error mechanism). The operators were designed with *nagsoft* as the user-friendly error-handling mechanism in mind.

parameter	message
1	OPERAND OF <operator name> OF WRONG SIZE
	The given array is of too small size; the result yielded is the operand.
2	OPERAND OF <operator name> CONTAINS NONZERO FIRST AND/OR LAST SINE COEFFICIENT
	The calculation is performed with the nonzero elements in question overwritten with zeros.
3	OPERAND (REAL VECTOR) OF *CO6TRGSUM* IS OF ODD LENGTH
	The calculation is performed with an adapted operand; the smaller of the first and the last element is discarded.

7. Auxiliary routines

The used NAG library operators are given in the following table.

Formula	Used NAG library operators	Used Torrix operators/generators
CO6TRGSUM f	*CO6EXPSUM*	*gencoarrayl,/<, gencoarray* 1
CO6TRGSUM ab	*CO6EXPSUMHRM*	*genarrayl,/<, conj, genarray* 1, *widen*
CO6COSSUM a	*CO6EXPSUMHRM*	*/<*
CO6SINSUM b	*CO6EXPSUMANH r*	*/<*

8. Timing

The time taken is proportional to s*m, where s is the sum of the factors of m.

9. <u>Storage</u> No auxiliary arrays are declared.

10. <u>Accuracy</u>

The accuracy is determined by the DFT because of the second relation in the description.

Let us denote by

. c : the machine representation of a (measured) vector $\underline{c}$;

. $\underline{\Delta c}$: $\left| c - \underline{c} \right|$.

Error bounds (first order).

Formula	Propagated error	Generated error
C06TRGSUM f	$\\| \underline{\Delta f} \\|_2$ $/\max(1/\sqrt{n}, \\|\underline{f}\\|_2)$	$g_f *$ *small real*
C06TRGSUM ab	$\\| \underline{\Delta a} \\|_2$ $/\max(1/\sqrt{n}, \\|\underline{a}\\|_2) + \\| \underline{\Delta b} \\|_2 /\max(1/\sqrt{n}, \\|\underline{b}\\|_2)$	$(g_a + g_b) *$ *small real*
C06COSSUM a	$\\| \underline{\Delta a} \\|_2$ $/\max(1/\sqrt{n}, \\|\underline{a}\\|_2)$	$g_a *$ *small real*
C06SINSUM b	$\\| \underline{\Delta b} \\|_2$ $/\max(1/\sqrt{n}, \\|\underline{b}\\|_2)$	$g_b *$ *small real*

The growth factors $g_f, g_a,$ g_b are of the order of magnitude

$$\sim \sum_i p_i^{1.5}$$

with p_i the factors of m.

11. <u>Further comments</u> None.

12. <u>Keywords</u>

Discrete harmonic analysis.

Trigonometric interpolation.

Discrete harmonic synthesis.

Evaluation of trigonometric sums.

Fast Fourier Transform.

Cooley-Sande-Stockham algorithm.

13. Examples

C06TRGSUM

13.1 Program text.

```
'BEGIN'
    #
    AS AN ILLUSTRATION OF THE USE OF 'C06TRGSUM' THIS PROGRAM CALCULATES
    APPROXIMATELY THE DISCRETE HARMONIC ANALYSIS AND THE DISCRETE
    HARMONIC SYNTHESIS.
    #
    'VEC'F=GENVEC(4):=(1.0,0.0,3.0,4.0);
    WRITEF(($28A,L,28A,4(Q-D.DD),L,8A,L,42A,3(L,-D.DD17QIQ-D.DD)$,
            "'C06TRGSUM' EXAMPLE PROGRAM.",
            "DISCRETE FUNCTION, ON INPUT:",
            F,
            "RESULTS:",
            "COSINE COEFFICIENTS:    SINE COEFFICIENTS:",
            'C06TRGSUM'F));

    'COVEC'AB=GENCOVEC(5):=(2.0'I'0.0,0.0'I'1.0,0.0'I'2.0,2.0'I'3.0,
                4.0'I'0.0);

    WRITEF(($2L,42A,5(L,-D.DD17QIQ-D.DD),L,28A,8(Q-D.DD)$,
            "COSINE COEFFICIENTS:    SINE COEFFICIENTS:",
            AB,
            "RESULTING DISCRETE FUNCTION:",
            'C06TRGSUM'AB))
"END'#OF 'C06TRGSUM' EXAMPLE PROGRAM#
```

13.2 Data for program. None.

13.3 Results.

```
'C06TRGSUM' EXAMPLE PROGRAM.
DISCRETE FUNCTION, ON INPUT:  1.00 0.00 3.00 4.00
RESULTS:
COSINE COEFFICIENTS:    SINE COEFFICIENTS:
  4.00                I  0.00
 -1.00                I -2.00
  0.00                I  0.00

COSINE COEFFICIENTS:    SINE COEFFICIENTS: (ON INPUT)
  2.00                I  0.00
  0.00                I  1.00
  0.00                I  2.00
  2.00                I  3.00
  4.00                I  0.00
RESULTING DISCRETE FUNCTION:  5.00 2.41 1.00 1.24 1.00 -0.41 5.00 -7.24
```

C06COSSUM and C06SINSUM

13.1 Program text.

```
'BEGIN'
    #
    AS AN ILLUSTRATION OF THE USE OF 'C06COSSUM' AND 'C06SINSUM' THIS
    PROGRAM CALCULATES APPROXIMATELY THE DISCRETE COSINE TRANSFORM AND
    THE DISCRETE SINE TRANSFORM.
    #
    'VEC'A=GENVEC(5):=(2.0,0.0,0.0,2.0,4.0);
    WRITEF(($28A,L,16A,5(Q-D.DD),L,36A,5(Q-D.DD)$,
            "'C06COSSUM' EXAMPLE PROGRAM.",
            "VECTOR ON INPUT:",
            A,
            "RESULT OF DISCRETE COSINE TRANSFORM:",
            'C06COSSUM'A));

    'VEC'B=GENVEC(5):=(0.0,1.0,2.0,3.0,0.0);
    WRITEF(($2L,28A,L,16A,5(Q-D.DD),L,34A,5(Q-D.DD)$,
            "'C06SINSUM' EXAMPLE PROGRAM.",
            "VECTOR ON INPUT:",
            B,
            "RESULT OF DISCRETE SINE TRANSFORM:",
            'C06SINSUM'B))
'END'#OF 'C06COSSUM' AND 'C06SINSUM' EXAMPLE PROGRAM#
```

13.2 Data for program. None.

13.3 Results.

```
'C06COSSUM' EXAMPLE PROGRAM.
VECTOR ON INPUT:  2.00  0.00  0.00  2.00  4.00
RESULT OF DISCRETE COSINE TRANSFORM:  5.00  -2.41  3.00  0.41  1.00

'C06SINSUM' EXAMPLE PROGRAM.
VECTOR ON INPUT:  0.00  1.00  2.00  3.00  0.00
RESULT OF DISCRETE SINE TRANSFORM:  0.00  4.83  -2.00  0.83  0.00
```

14. Source texts

```
OP   CO6TRGSUM =( COVEC ab) VEC :

#purpose:           approximately calculated is

                     m
                    sum" a[k+ LWB ab]×cos(pi×k×j/m)
                    k=0


                    +


                     m
                    sum b[k+ LWB ab]×sin(pi×k×j/m)
                    k=0

                    for j=0,1,...,2×m-1 with m= SIZE ab-1.
        results:    the discrete harmonic synthesis i.e. the result of
                    the above formula for j=0,1, ... ,2×m-1 is
                    delivered as a  VEC  with bounds  LWB ab and
                    LWB ab+2×m-1. the vectors a and b must be given as
                    real and imaginary part of the  COVEC  ab.
exception handling:if m<1 then c06fail is called and the result is a
                    VEC  with bounds  LWB  ab and LWB ab+2×m-1.#

IF   INT 1= LWB ab,m= SIZE ab-1;m>0
THEN   SCAL zero= WIDEN 0;
    IF im OF ab[1]≠zero OR im OF ab[1+m]≠zero
    THEN c06fail(2,
        "trgsumoperand of  CO6TRGSUM  contains nonzero first and/or
        last element");
        im OF ab[1]:=im OF ab[1+m]:=zero
    FI ;
    ( CO6EXPSUMHRM  CONJ ab)/<2
ELSE c06fail(1,
    " TRGSUMOPERAND OF  c06trgsum  OF WRONG SIZE");
    genarray1(1,1+2×m-1)
FI ,
```

```
OP  CO6TRGSUM =( VEC f) COVEC :
```

#purpose: approximately calculated are

$$a[j]=1/m \sum_{k=0}^{n-1} f[k+ LWB \ f] \times \cos(pi \times k \times j/m),$$

and

$$b[j]=1/m \sum_{k=0}^{n-1} f[k+ LWB \ f] \times \sin(pi \times k \times j/m),$$

with $n=2 \times m=$ SIZE f.

input: see above formula. f is not preserved.

results: the coefficients a and b are delivered as real and imaginary part of a COVEC with bounds LWB f and LWB f+m.

exception handling:if n<1 then c06fail is called and the resulting COVEC has lower bound lwbf and upper bound LWB f+m. if n is odd then c06fail is called and the smaller of the first and the last element is discarded and CO6TRGSUM is called with the modified operand.#

```
IF  INT n= SIZE f,l= LWB f; INT m=n OVER 2;n>0
THEN  IF  NOT  ODD n
    THEN ( CO6EXPSUM f)[1:1+m AT 1]/<m
    ELSE c06fail(3,
        "trgsumoperand (real vector) of  CO6TRGSUM  is of odd length");
        INT u= UPB f;
        IF  ABS f[1]< ABS f[u]
        THEN  CO6TRGSUM f[l+1:u AT 1]
        ELSE  CO6TRGSUM f[1:u-1 AT 1]
        FI
    FI
ELSE c06fail(1,"trgsumoperand of  CO6TRGSUM OF WRONG SIZE");
    gencoarrayl(1,1+m)
FI ,
```

```
OP   CO6COSSUM =( VEC a) VEC :
#purpose:            approximately calculated is
                      m
                     sum" a[k+ LWB a]×cos(pi×k×j/m),  j=0,1, ... ,m
                     k=0

                     m= SIZE a-1.
input:               see above formula, a is not preserved.
results:             the cosine transform, i.e. the result of the
                     above formula is delivered as a  VEC  with bounds
                     similar to those of a.
exception handling:if m<1 then c06fail is called and the original
                     vector a is delivered.#


IF  INT ml= SIZE a;ml>1
THEN  CO6EXPSUMHRM (a/<2)
ELSE c06fail(1,"cossumoperand of  CO6COSSUM  of wrong size");a
FI ,
```

```
OP   CO6SINSUM =( VEC a) VEC :
#purpose:            approximately calculated is


                      m
                     sum a[k+ LWB a]×sin(pi×k×j/m),  j=0,1, ... ,m
                     k=0

                     m= SIZE a-1.
input:               see above formula, a is not preserved. note that:
                     the first and last element of a must be supplied
                     and filled with zero.
results:             the sine transform, i.e. the results of the
                     above formula is delivered as a  VEC  with bounds
                     similar to those of a. note that the first and last
                     element are zero again.
exception handling:if m<1 then c06fail is called and the original
                     vector a is delivered.#


IF  INT ml= SIZE a;ml>1
THEN  SCAL zero= WIDEN 0;
   IF a[ LWB a]≠zero OR a[ UPB a]≠zero
   THEN c06fail(2,
       "sinsumoperand of  CO6SINSUM  containes nonzero first and/or
       last sine coefficient");
       a[ LWB a]:=a[ UPB a]:=zero
   FI ;
   CO6EXPSUMANH (a/<2)
ELSE c06fail(1,"sinsumoperand of  CO6SINSUM  of wrong size");a
FI
```

II.4 *C06TRGSUM, C06COSSUM, C06SINSUM* (Dyadic)

1. Purpose

The dyadic operators

C06TRGSUM, C06COSSUM, C06SINSUM

evaluate trigonometric sums.

Advantage has been taken of zeros in the data in

C06COSSUM - sine coefficients are zero

C06SINSUM - cosine coefficients are zero.

IMPORTANT:....

2. Specification (Algol 68)

MODE SCAL	*= REAL, COSCAL = COMPL;*
MODE VEC	*= REF[ ]SCAL,*
COVEC	*= REF[ ]COSCAL;*
OP C06TRGSUM	*= (SCAL t, COVEC ab) SCAL:*
OP C06COSSUM	*= (SCAL t, VEC a) SCAL:*
OP C06SINSUM	*= (SCAL t, VEC b) SCAL:*
PRIO C06TRGSUM	*= 8, C06COSSUM = 8, C06SINSUM = 8.*

3. Description

The operators calculate

$$f(\theta, \underline{a}, \underline{b}) = \sum_{k=0}^{m} (a_k \cos k\theta + b_k \sin k\theta).$$

The algorithm used for evaluating a cosine sum and a sine sum is the Clenshaw algorithm with the modifications due to Reinsch.

4. References

[1] OLIVER , J.

An error analysis of the modified Clenshaw method for evaluating Chebyshev and Fourier series.

JIMA, vol. 20, 379-391. 1977.

5. Parameters

General:

. Both operands are preserved.

. The result is a real constant: the trigonometric sum.

. Only the size of the right operand matters: the k-th element of a vector
a is assumed to be represented by $a[LWB\ a + k]$, so the lower bound of the
data representation of the vector does not matter and is free for choice.

. m denotes the size of the coefficient vector minus 1.

. Left operand t: the angle θ; a real constant (it is advised to supply
a value within $[-\pi,\pi)$).

Formula	Right operand
t *C06TRGSUM ab*	a complex array with m+1 elements: the real part contains the cosine coefficients, a, and the imaginary part contains the sine coefficients, b. The first element of b must contain zero.
t *C06COSSUM a*	a real array with m+1 elements: the coefficients of the cosine sum, a.
t *C06SINSUM b*	a real array with m+1 elements: the coefficients of the sine sum, b. The first element must contain zero.

6. Error indicators

In the event of an error condition being detected, the error routine:
c06fail of mode *REF NAGFAIL*, is called with the parameters listed below.
These are printed and in case the value of *c06fail* is *nagsoft* the execution
is continued (see in Introduction of the NAG manual the document on the
ALGOL 68 error mechanism). The operators were designed with *nagsoft* as the
user-friendly error-handling mechanism in mind.

parameter	message
1	VECTOR OPERAND OF *<operator name>* OF WRONG SIZE
	The size of the given vector is smaller than one;

the result yielded is zero as value for the empty sum.

2   VECTOR OPERAND OF *<operator name>* CONTAINS NONZERO FIRST SINE COEFFICIENT

The calculation is performed with the first sine coefficient overwritten with zero.

7. Auxiliary routines   None.

8. Timing
The time taken is proportional to m.

9. Storage   No auxiliary arrays are declared.

10. Accuracy

Let us denote by:

. $\underline{a}$: the cosine coefficient vector;
. $\underline{b}$: the sine coefficient vector;
. θ : the angle;
. c_a, c_b : the condition numbers

$$c_a = \sum_{k=0}^{m} \max(1,|a_k|)/\max(1, |f(\theta,\underline{a},\underline{b})|),$$

$$c_b = \sum_{k=0}^{m} \max(1,|b_k|)/\max(1, |f(\theta,\underline{a},\underline{b})|);$$

. c_θ : the condition number

$$c_\theta = |\max(1,|\theta|)\frac{\partial f}{\partial \theta}|/\max(1, |f(\theta,\underline{a},\underline{b})|);$$

. g_a, g_b : growth factors (conjectured of order m);
. a : the machine representation of (the measured) $\underline{a}$;
. b : the machine representation of (the measured) $\underline{b}$;

. δa, δb : vectors of componentswise errors:

$$\delta a_k = |a_k - a[LWB\ a + k]|/\max(1, |a_k|)$$

$$\delta b_k = |b_k - b[LWB\ b + k]|/\max(1, |b_k|)$$

for all appropriate k;

. t : the machine representation of (the measured) θ;

. δθ : $|t - \theta|/\max(1, |\theta|)$.

Error bounds (first order)

Formula	Propagated error	Generated error
t C06TRGSUM ab	$c_a * \|\delta a\|_\infty + c_b * \|\delta b\|_\infty + c_\theta * \delta\theta$	$(c_a * g_a + c_b * g_b) * small\ real$
t C06COSSUM a	$c_a * \|\delta a\|_\infty + c_\theta * \delta\theta$	$c_a * g_a * small\ real$
t C06SINSUM b	$c_b * \|\delta b\|_\infty + c_\theta * \delta\theta$	$c_b * g_b * small\ real$

11. Further comments   None.

12. Keywords

Evaluation of trigonometric sum.

Clenshaw-Reinsch algorithm.

13. Examples

13.1 Program text.

```
'BEGIN'
    #
    AS AN ILLUSTRATION OF THE USE OF 'C06COSSUM', 'C06SINSUM' AND
    'C06TRGSUM' (DYADIC) THE PROGRAM CALCULATES APPROXIMATELY

     1
    SUM (A[K]*COS(K*T)+B[K]*SIN(K*T))
    K=0

    #
    'SCAL'ANGLE=PI/2;
    'COVEC'AB=(GENCOVEC(2):=(1.0'I'0.0,2.0'I'1.0))['AT'0];
    'VEC'A=RE'OF'AB,
         B=IM'OF'AB;
    WRITEF(($57A,L,6A,Q-D.DD,L,20A,2(Q-D.DD),L,18A,Q-D.DD,2L,18A,
             2(Q-D.DD),L,16A,Q-D.DD,2L,25A,Q-D.DD$,
             "'C06COSSUM', 'C06SINSUM' AND 'C06TRGSUM' EXAMPLE PROGRAM.",
             "ANGLE:",ANGLE,
             "COSINE COEFFICIENTS:",A,
             "RESULT COSINE SUM:",ANGLE'C06COSSUM'A,
             "SINE COEFFICIENTS:",B,
             "RESULT SINE SUM:",ANGLE'C06SINSUM'B,
             "RESULT TRIGONOMETRIC SUM:",ANGLE'C06TRGSUM'AB))
'END'#OF 'C06COSSUM', 'C06SINSUM' AND 'C06TRGSUM' EXAMPLE PROGRAM#
```

13.2 Data for program. None.

13.3 Results.

```
'C06COSSUM', 'C06SINSUM' AND 'C06TRGSUM' EXAMPLE PROGRAM.
ANGLE:  1.57
COSINE COEFFICIENTS:  1.00  2.00
RESULT COSINE SUM:  1.00

SINE COEFFICIENTS:  0.00  1.00
RESULT SINE SUM:  1.00

RESULT TRIGONOMETRIC SUM:  2.00
```

14. Source texts

```
OP  CO6TRGSUM =( SCAL t, COVEC ab) SCAL :
```

```
#purpose:              approximately calculated is
```

$$\sum_{k=0}^{m} (a[k]\times\cos(k\times t)+b[k]\times\sin(k\times t))$$

```
                       with m= SIZE ab-1.
input:                 the angle t of MODE  SCAL  must be given as left
                       operand. the right operand of MODE  COVEC
                       contains the cosine coefficients as real part and
                       the sine coefficients as imaginary part. the first
                       element of the imaginary part is supposed to refer
                       to zero.
results:               the trigonometric sum of MODE  SCAL .
exception handling:if m<0 then cO6fail is called and zero is
                       delivered as the value of the empty  sum. if the
                       first element of the imaginary part does not refer
                       to zero then cO6fail is called and the calculation
                       is performed after the element is overwritten with
                       zero.#
```

```
IF  SCAL zero= WIDEN 0; INT m= SIZE ab-1;m>-1
THEN  VEC a=re OF ab,b=im OF ab;
    IF b[ LWB b]≠zero
    THEN cO6fail(2,
        "trgsumvector operand of  CO6TRGSUM  contains nonzero first
            sine coefficient");
        b[ LWB b]:=zero
    FI ;
    SCAL sinsum,cossum;
    (cO6ser(a,cossum, NIL ,t),cO6ser(b, NIL ,sinsum,t));
    cossum+sinsum
ELSE cO6fail(1,"trgsumvector operand of  CO6TRGSUM  of wrong size");
    zero
FI ,
```

```
OP   CO6COSSUM =( SCAL t, VEC a) SCAL :

#purpose:              approximately calculated is


              m
              sum a[ LWB a+k]×cos(k×t)
              k=0


              with m= SIZE a-1.
input:              the cosine coefficients must be given as a   VEC
              with arbitrary first index.
result:              the cosine sum.
exception handling:if m<0 then c06fail is called and zero is
              delivered for the empty sum.#


IF   INT m= SIZE a-1;m>-1
THEN   SCAL cossum;
   c06ser(a,cossum, NIL ,t);cossum
ELSE c06fail(1,"cossumvector operand of  CO6COSSUM  of wrong size");
   WIDEN 0
FI ,
```



```
OP   CO6SINSUM =( SCAL t, VEC b) SCAL :

#purpose:              approximately calculated is


              m
              sum b[ LWB b+k]×sin(k×t)
              k=0


              with m= SIZE b-1.
input:              the sine coefficients must be given as a   VEC
              with arbitrary first index. the first element is
              supposed to refer to zero.
result:              the sine sum.
exception handling:if m<0 then c06fail is called and zero is
              delivered as the value of the empty sum. if
              b[ LWB b] does not refer to zero then c06fail is
              called and the calculation is performed after
              b[ LWB b] is overwritten with zero.#


IF   SCAL zero= WIDEN 0; INT m= SIZE b-1;m>-1
THEN   IF b[ LWB b]≠zero
   THEN c06fail(2,
       "sinsumvector operand of  CO6SINSUM  contains nonzero first
          sine coefficient");
       b[ LWB b]:=zero
   FI ;
   SCAL sinsum;
   c06ser(b, NIL ,sinsum,t);
   sinsum
ELSE c06fail(1,"sinsumvector operand of  CO6SINSUM  of wrong size");
   zero
FI
```

III. SOURCE TEXTS OF THE TECHNICAL ROUTINES

III.1 Hierarchy of the implementations (monadic operators)

CO6TRGSUM r   CO6TRGSUM c   CO6COSSUM   CO6SINSUM

CO6EXPSUM r   CO6EXPSUMHRM r ⟷ CO6EXPSUMANH r   CO6EXPSUMANH c

CO6EXPSUM   CO6EXPSUMHRM c

CO6INITW   CO6SINTWI

CO6FFT
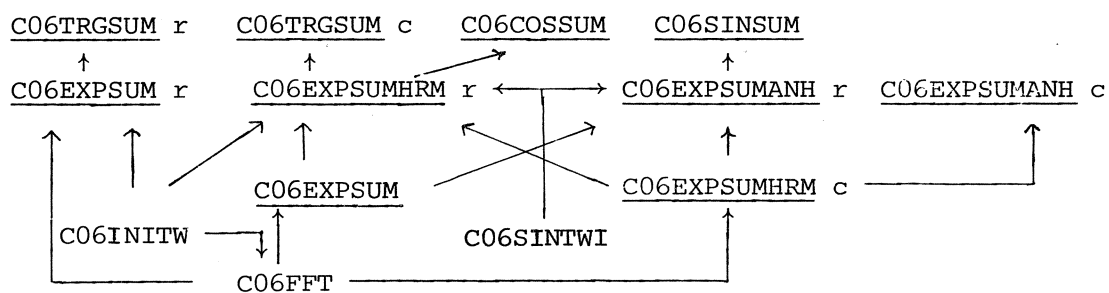
III.2 Hierarchy of the implementations (dyadic operators)

CO6EXPSUM   CO6TRGSUM   CO6COSSUM   CO6SINSUM

CO6SER
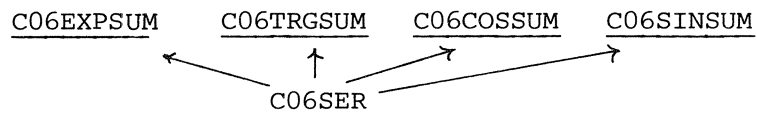
III.3 Source texts

```
PROC   cO6sintwi=( INT n, VEC s) VOID :

#purpose:the points sin(2×pi×j/n), j=0,1, ... , UPB s
         are calculated.
input:     VEC s  with  LWB s=0 and  UPB s>0;
           INT n  with n>0.
results: the points are delivered in s.#

IF  INT u= UPB s; LWB s=0  AND  u>0  AND  n>1
THEN  IF u<n
   THEN s[0]:= WIDEN 0;
      IF u>0
      THEN  SCAL spi=pi,two= WIDEN 2;
         SCAL  tpi=two×spi;
         SCAL  the=tpi/ WIDEN n;
         s[1]:=sin(the);
         INT ttp:=1; INT uu=u MIN (n OVER 2);
         WHILE ttp×:=2;ttp<uu
         DO s[ttp]:=sin(ttp×the) OD ;
         INT thp=ttp OVERAB 2;
         IF  INT l:=thp; l<uu
         THEN  WHILE (ttp OVERAB 2)>0
            DO  IF l+ttp<uu
               THEN l+:=ttp;s[l]:=sin(l×the)
               FI
            OD
         FI ;
         INT k:=thp OVER 2;
         WHILE  (k  OVERAB  2) > 1
         DO  IF  SCAL tct=two×cos(k×the); ABS tct> WIDEN 1
             THEN  FOR  j  FROM  3×k  BY  2×k  TO  uu-k
                DO s[j]:=(s[j-k]+s[j+k])/tct OD
             ELSE  FOR  j  FROM  3×k  BY  2×k  TO  uu-k
                DO s[j]:=tct×s[j-k]-s[j-2×k] OD
             FI
         OD ;
         FOR j FROM uu+1 TO u DO s[j]:=-s[n-j] OD
      FI
   ELSE cO6sintwi(n,s[0:n-1 AT 0]);
      FOR j FROM n TO u DO s[j]:=s[j-n] OD
   FI
ELSE   cO6fail(1,"sintwiwrong size and/or bounds in cO6sintwi")  FI ,
```

```
PROC  c06initw=( INT n, COVEC w) VOID :

#purpose:the twiddle factors exp(0 I 2×pi×j/n), j=0,1, ... ,    UPB w
          are calculated.
input:    COVEC w  with  LWB w=0 and  UPB w>0;
          INT n  with n>0.
results: the twiddle factors are delivered in w.#


IF   INT u= UPB w; LWB w=0  AND  u>0  AND  n>1
THEN  IF u<n
      THEN  SCAL scalpi=pi,two= WIDEN 2,one= WIDEN 1;
            SCAL  tpi=scalpi+scalpi;
            SCAL  kth:=(n=1! WIDEN 0!tpi/ WIDEN n),
            INT k:=1;
            INT uu=u MIN (n OVER 2);
            w[0]:= WIDEN one;
            WHILE k <uu
            DO w[k]:=(cos(kth),sin(kth));
               k× :=2;
               kth× :=two
            OD ;
            INT i=k OVERAB 2;k OVERAB 2;
            WHILE (k OVERAB 2) > 1
            DO  IF SCAL tct=two×re OF w[k]; ABS  tct >  one
                THEN  FOR  j  FROM  3×k  BY  2×k  TO  i-k
                   DO w[j]:=(w[j-k]+w[j+k])/tct OD
                ELSE  FOR  j  FROM  3×k  BY  2×k  TO  i-k
                   DO w[j]:=tct×w[j-k]-w[j-2×k] OD
                FI
            OD ;
            FOR  j  FROM  i+1  TO  uu DO w[j]:=w[i]×w[j-i] OD ;
            FOR  j  FROM  uu+1 TO  u DO w[j]:= CONJ w[n-j] OD
      ELSE c06initw(n,w[0:n-1 AT 0]);
            FOR  j  FROM  n  TO  u DO w[j]:=w[j-n] OD
      FI
ELSE  c06fail(1,"c06initw wrong size and/or bounds")
FI ,
```

```
PROC  c06fft=( COVEC xy) VOID :

#purpose:           approximately calculated is

                    n-1
                    sum w[j,k]×xy[k+ LWB xy],   j=0,1, ... ,n-1
                    k=0

                    with n= SIZE xy,  w[j,k]=exp(0 I j×k×2×pi/n).
input:              see above formula.
results:            the idft of xy is delivered with
                    bounds pair 0:n-1.
exception handling:if  NOT ( LWB xy=0 AND  SIZE xy>0)then
                    c06fail is called.#


IF  INT n= UPB xy+1;  LWB xy=0  AND  n>1  THEN
    IF  n>1  THEN

PROC fftri=( COVEC x,w, INT ri,rlriminl,ripluslrm,n) VOID :
CASE
    PROC xi=( INT i) COVEC :x[ AT -i×ripluslrm],
    COSCAL zero= WIDEN  WIDEN 0,
    INT rirm=ri×ripluslrm,noverri=n OVER ri,risubl=ri-1,
        ripluslrmsubl=ripluslrm-1;
    INT nsubrirm=n-rirm;
    risubl
IN
BEGIN
    COVEC x0=xi(0),x1=xi(1);
    FOR  k FROM  0 TO  ripluslrmsubl DO
        COSCAL  wn=w[k×rlriminl];
        FOR  j FROM  k BY  rirm TO  nsubrirm+k  DO
            REF  COSCAL x0j=x0[j],x1j=x1[j];
            COSCAL  a=x0j,b=x1j;
            (x0j:=a+b,x1j:=(a-b)×wn)
        OD
    OD
END ,
BEGIN
    COVEC x0=xi(0),x1=xi(1),x2=xi(2);
    COSCAL  ei120=w[noverri],ei240=w[2×noverri];
    FOR  k FROM  0 TO  ripluslrmsubl DO
        INT ind=k×rlriminl;
        COSCAL  w1n=w[ind],w2n=w[2×ind];
        FOR  j FROM  k BY  rirm TO  nsubrirm+k  DO
            REF  COSCAL x0j=x0[j],x1j=x1[j],x2j=x2[j];
            COSCAL  a=x0j,b=x1j,c=x2j;
            (x0j:=(a+b        +c      )    ,
             x1j:=(a+b×ei120+c×ei240)×w1n,
             x2j:=(a+b×ei240+c×ei120)×w2n)
        OD
    OD
END ,
```

```
BEGIN
    COVEC x0=xi(0),x1=xi(1),x2=xi(2),x3=xi(3),
    PROC ( COSCAL ) COSCAL ei90=
            ( COSCAL a) COSCAL :(- IM a, RE a) ;
    IF  noverri=rlriminl  THEN
        FOR  j  FROM  0  BY  rirm  TO  nsubrirm    DO
            REF   COSCAL x0j=x0[j],x1j=x1[j],x2j=x2[j],x3j=x3[j];
            COSCAL e=x0j+x2j,f=x0j-x2j,g=x1j+x3j,h=ei90(x1j-x3j);
            (x0j:=e+g,x2j:=e-g,x1j:=f+h,x3j:=f-h)
        OD
    ELSE
        FOR  k  FROM  0  TO  riplus1rmsubl  DO
            INT ind=k×rlriminl;
            COSCAL w1n=w[ind],w2n=w[2×ind],w3n=w[3×ind];
            FOR  j  FROM  k  BY  rirm  TO  nsubrirm+k  DO
                REF   COSCAL x0j=x0[j],x1j=x1[j],x2j=x2[j],x3j=x3[j];
                COSCAL e=x0j+x2j,f=x0j-x2j,g=x1j+x3j,h=ei90(x1j-x3j);
                (x0j:=e+g,x2j:=(e-g)×w2n,x1j:=(f+h)×w1n,x3j:=(f-h)×w3n)
            OD
        OD
    FI
END ,
BEGIN
    COVEC x0=xi(0),x1=xi(1),x2=xi(2),x3=xi(3),x4=xi(4);
    COSCAL  ei 72=w[  noverri],ei144=w[2×noverri],
            ei216=w[3×noverri],ei288=w[4×noverri];
    FOR  k  FROM  0  TO  riplus1rmsubl  DO
        INT ind=k×rlriminl;
        COSCAL  w1n=w[ind],w2n=w[2×ind],w3n=w[3×ind],w4n=w[4×ind];
        FOR  j  FROM  k  BY  rirm  TO  nsubrirm+k  DO
            REF   COSCAL x0j=x0[j],x1j=x1[j],x2j=x2[j],
                        x3j=x3[j],x4j=x4[j];
            COSCAL a=x0j,b=x1j,c=x2j,d=x3j,e=x4j;
            (x0j:=(a+b       +c       +d       +e       )     ,
             x1j:=(a+b×ei 72+c×ei144+d×ei216+e×ei288)×w1n,
             x2j:=(a+b×ei144+c×ei288+d×ei 72+e×ei216)×w2n,
             x3j:=(a+b×ei216+c×ei 72+d×ei288+e×ei144)×w3n,
             x4j:=(a+b×ei288+c×ei216+d×ei144+e×ei 72)×w4n)
        OD
    OD
END
OUT
IF  ri=n  THEN
    COVEC xp= COPY (x);
    BEGIN
        REF  COSCAL s=x[0]:=zero;
        FOR  p  FROM  0  TO  risubl  DO  s+:=xp[p]  OD
    END ;
    FOR  j  FROM  1  TO  risubl  DO
        INT  b:=j,
        REF  COSCAL s=x[j]:=xp[0];
        FOR  p  FROM  1  TO  risubl  DO
            s+:=xp[p]×w[b];
            ( (b+:=j) > n ! b-:=n )
        OD
    OD
```

```
ELSE
    [0:risubl] COVEC  xp;
    FOR  p  FROM  0  TO  risubl  DO  xp[p]:=xi(p)  OD ;
    FOR  k  FROM  0  TO  ripluslrmsubl  DO
        INT ind=k×rlriminl;
        FOR  j  FROM  k  BY  rirm  TO  nsubrirm+k  DO
            COVEC xpj=gencoarrayl(0,risubl);
            BEGIN
                COSCAL  s:=zero;
                FOR  p  FROM  0  TO  risubl  DO
                    s+:=(xpj[p]:=xp[p][j])
                OD ;
                xp[0][j]:=s
            END ;
            INT c:=0,d:=0;
            FOR  p  FROM  1  TO  risubl  DO
                (c+:=noverri,d+:=ind);
                INT b:=d,
                REF  COSCAL  s=xp[p][j]:=zero;
                FOR  q  FROM  0  TO  risubl  DO
                    s+:=xpj[q]×w[b];
                    ((b+:=c)>n!b-:=n)
                OD
            OD
        OD
    OD
FI
ESAC ,

MODE  R = STRUCT ( INT ri,rlriminl,ripluslrm),
MODE  L = STRUCT ( REF  R r, REF  L next),

PROC factor=( INT n, REF  REF  L 113,12,1123, REF  INT rimax) VOID :
BEGIN
    REF  L klb:= NIL ,kle,k2b:= NIL ,k2e,k3b:= NIL ,k3e,
          mlb:= NIL ,mle,m2b:= NIL ,m2e,m3b:= NIL ,m3e,
    PROC inlist=( INT ri) VOID :
    IF
        (ri>rimax!rimax:=ri);
        PROC list=( REF  REF  L lb,le, BOOL bef, REF  R r) VOID :
        IF  lb:=: REF  L ( NIL ) THEN
            lb:=le:= HEAP  L :=(r, NIL )
        ELIF  bef THEN
            lb:= HEAP  L :=(r,lb)
        ELSE
            le:=next OF le:= HEAP  L :=(r, NIL )
        FI ;
        lastri=0
    THEN
        lastri:=ri
    ELIF  lastri=ri  THEN
```

```
      HEAP  R  rl,r2;
      ri OF rl:=ri OF r2:=ri;
      list(klb,kle,  FALSE  ,rl);list(mlb,mle,  FALSE  ,rl);
      list(k3b,k3e,  TRUE   ,r2);list(m3b,m3e,  TRUE   ,r2);
      lastri:=0
ELSE
      HEAP  R  r;
      ri OF r:=lastri;
      list(k2b,k2e,  FALSE  ,r );list(m2b,m2e,  FALSE  ,r );
      lastri:=ri
FI ,
      INT npart:=n,lastri:=0;
      rimax:=0;

      WHILE npart  MOD  4=0  DO
          npart  OVERAB  4;inlist(4)
      OD ;
      IF  NOT  ODD  npart  THEN
          npart  OVERAB  2;inlist(2)
      FI ;
      INT div:=3;
      WHILE
          IF  npart  MOD  div =0  THEN
              npart  OVERAB  div;inlist(div);
              TRUE
          ELIF  npart  OVER  div>div  THEN
              div+:=2;
              TRUE
          ELSE  FALSE
          FI
      DO  SKIP  OD ;
      IF  npart>1  THEN  inlist(npart)  FI ;
      inlist(0);

      IF  klb:=:  REF  L  (  NIL  )  THEN
          ll3:=  NIL  ;l2:=ll23:=k2b
      ELIF  k2b:=:  REF  L  (  NIL  )  THEN
          l2:=  NIL  ;ll3:=ll23:=klb;next  OF  kle:=k3b
      ELSE
          l2:=k2b;ll3:=klb;next  OF  kle:=k3b;
          ll23:=mlb;next  OF  mle:=m2b;next  OF  m2e:=m3b
      FI ;

      INT rlriminl:=1,
      REF  L  l:=ll23;
      WHILE
          REF  R  r=r  OF  l;
          rlriminl  OF  r:=rlriminl ;
          ripluslrm  OF  r:=n  OVER  (rlriminl×:=ri  OF  r);
          (l:=next  OF  l):≠:  REF  L  (  NIL  )
      DO  SKIP  OD
END ,
```

```
PROC  revers=( COVEC x, REF  L 113,12, INT n) VOID :
IF
    PROC  perm=( REF  L 1, PROC ( INT , INT ) VOID pr) VOID :
    BEGIN
        PROC pp=( REF  L 1, INT j,k) VOID :
        IF  REF  R r=r OF 1;next OF 1:=: REF  L ( NIL ) THEN
            FOR  p FROM  0  TO  ri OF r-1  DO
                pr(j+p×rlriminl OF r,k+p×ripluslrm OF r)
            OD
        ELSE
            FOR  p FROM  0  TO  ri OF r-1  DO
                pp(next OF 1,j+p×rlriminl OF r,k+p×ripluslrm OF r)
            OD
        FI ;
        pp(1,0,0)
    END ;


    (113:≠: NIL ) AND  (12: =: NIL ) THEN
        perm(113,( INT j,k) VOID :(j<k!x[j]=:=x[k]))
ELIF (113: =: NIL ) AND  (12:≠: NIL ) THEN
    IF  next OF 12:≠: REF  L ( NIL ) THEN
        INDEX p=genintarray(0,n-1);
        perm(12,( INT j,k) VOID :p[j]:=k);
        FOR  j FROM  1  TO  n-2  DO
            IF  INT k:=p[j]; j≠k THEN
                COSCAL s=x[k];p[j]:=j;
                WHILE  INT l=p[k];l≠k  DO
                    x[k]:=x[l];p[k]:=k;k:=l
                OD ;
                x[k]:=s
            FI
        OD
    FI
ELIF (113:≠: NIL ) AND  (12:≠: NIL ) THEN
    IF  next OF 12: =: REF  L ( NIL ) THEN
        INT step=ripluslrm OF r OF 12;
        INT stepspan=step×(ri OF r OF 12-1);
        PROC  change=( INT j,k) VOID :
        IF  j < k THEN
            COVEC xj=x[ AT -j],xk=x[ AT -k];
            FOR  p FROM  0 BY  step TO  stepspan DO
                xj[p]=:=xk[p]
            OD
        FI ;
        perm(113,change)
    ELSE
        INT step=( REF  L 1:=12;  WHILE next OF 1:≠: REF  L ( NIL )
                DO  1:=next OF 1  OD ;ripluslrm OF r OF 1);
        INT span=n OVER (step×rlriminl OF r OF 12)-1;
        MODE  CYCLE = STRUCT ( INT no, REF  CYCLE next),
        [0:span] CYCLE p;
        perm(12,( INT j,k) VOID :p[j OVER step]:=(k OVER step, NIL ));
```

```
MODE  LIST = STRUCT ( REF  CYCLE start, REF  LIST next),
REF  LIST l:= NIL ,
INT j:=0;
WHILE
    IF  REF  CYCLE  t:=p[j];
          next  OF  t :=:  REF  CYCLE ( NIL ) THEN
        WHILE
           REF  CYCLE s=t; INT k=no OF s;
           t:=p[k];s:=(j×step,t); j:=k;
           next OF t :=:  REF  CYCLE ( NIL )
        DO  SKIP  OD ;
        l:= HEAP  LIST :=(t,1)
    FI ;
    j <  span
DO  j+:=1  OD ;


PROC  listperm=( INT j,k) VOID :
IF  j=k  THEN
    REF  LIST list:=l,
    COVEC xj=x[ AT -j];
    WHILE
        IF  REF  CYCLE start=start OF list;
              next  OF  start :≠: start  THEN
            REF  CYCLE  t:=start, INT no:=no OF start;
            COSCAL s=xj[no];
            WHILE
                INT n=no OF (t:=next OF t);
                xj[no]:=xj[n];no:=n;
                next  OF  t :≠: start
            DO  SKIP  OD ;
            xj[no]:=s
        FI ;
        (list:=next OF list):≠: REF  LIST ( NIL )
    DO  SKIP  OD
ELIF  j<k  THEN
REF  LIST list:=l,
COVEC xj=x[ AT -j],xk=x[ AT -k];
WHILE
    IF  REF  CYCLE start=start OF list;
          next  OF  start :=: start  THEN
        INT no=no OF start;
        xj[no]=:=xk[no]
    ELSE
        REF  CYCLE  t:=start, INT no:=no OF start;
        COSCAL s1=xj[no],s2=xk[no];
        WHILE
            INT n=no OF (t:=next OF t);
            (xj[no]:=xk[n],xk[no]:=xj[n]);no:=n;
            next  OF  t :≠: start
        DO  SKIP  OD ;
        (xk[no]:=s1,xj[no]:=s2)
    FI ;
```

```
                    (list:=next OF list):≠: REF   LIST ( NIL )
                DO   SKIP   OD
        FI ;

            perm(113,listperm)
        FI
    FI ,

            INT rimax,
            REF   L 113,12,1123;
            factor(n,113,12,1123,rimax);
            COVEC  w=gencoarray1(0,n-(rimax>5!1!n OVER rimax) );
            c06initw(n,w);
            WHILE
                REF   R r=r OF 1123;
                fftri(xy,w,ri OF r,rlriminl OF r,ripluslrm OF r,n);
                (1123:=next OF 1123):≠: REF   L ( NIL )
            DO   SKIP   OD ;
            revers(xy,113,12,n)
        FI
    ELSE   c06fail(1,"c06fft wrong size")  FI ,
```

```
PROC c06ser=( VEC a, REF  SCAL cosser,sinser, SCAL t) VOID :

#purpose: calculation of either or both of the trigonometric sums
      u
   sum aa[k]×cos(k×t)            (cosine sum)
   k=0


      u
   sum aa[k]×sin(k×t)           (sine sum  )
   k=1


   with aa[k]=a[ LWB a+k]  and  u= SIZE a-1.

input parameters:  SCAL t  angle of trigonometric sum,
                   VEC a   coefficients of trigonometric sum,
                   REF  SCAL cosser,sinser  when containing NIL  no
                   cosine and/or sine sum are desired otherwise
                   the cosine and/or the sine sum are desired.
output parameters:  REF  SCAL cosser,sinser  they will contain
                   the cosine and sine sum, provided they
                   were not pointing to NIL  on input.#


IF (cosser:=: NIL ! SIZE a>0! SIZE a>0)    THEN
    PROC sqr=( SCAL c) SCAL :c×c,
    SCAL c=cos(t),zero= WIDEN 0,
    one= WIDEN 1,two= WIDEN 2,four= WIDEN 4;
    SCAL half=one/two;
    INT ua= UPB a,lapl= LWB a+1;
    IF  c<-half  THEN
        SCAL lambda=four×sqr(cos(t/two)), SCAL un:=zero,dun:=zero;
        FOR  k  FROM  ua  BY  -1  TO  lapl
            DO  dun:=lambda×un-dun+a[k];un:=dun-un  OD ;
        IF  cosser:≠: NIL  THEN  cosser:=lambda/two×un-dun+a[lapl-1]
        FI ;
        IF  sinser:≠: NIL  THEN  sinser:=un ×sin(t)            FI
    ELIF  c>half  THEN
        SCAL lambda=-four×sqr(sin(t/two)), SCAL un:=zero,dun:=zero;
        FOR  k  FROM  ua  BY  -1  TO  lapl
            DO  dun:=lambda×un+dun+a[k];un:=dun+un  OD ;
        IF  cosser:≠: NIL  THEN  cosser:=lambda/two×un+dun+a[lapl-1]
        FI ;
        IF  sinser:≠: NIL  THEN  sinser:=un ×sin(t)            FI
    ELSE
        SCAL cc=c+c , SCAL un1:=zero,un2:=zero;
        FOR  k  FROM  ua  BY  -1  TO  lapl
            DO  SCAL h=cc×un1-un2+a[k];un2:=un1;un1:=h  OD ;
        IF  cosser:≠: NIL  THEN  cosser:=un1×c-un2+a[lapl i]
        FI ;
        IF  sinser:≠: NIL  THEN  sinser:=un1×sin(t)            FI
    FI
ELSE  c06fail(1,"c06ser wrong size")  FI
```

IV. TESTING

Apart from the example test programs given in the documentation units
we considered for the stringent tests the cases:

. problems with known exact results (model problem),

. verification of relation of Parseval,

. verification of the pair: transformation and its inverse.

The dyadic operators are applied to those argument values implicit
in the monadic operators. The resulting formulas of the above cases
were published (in Dutch) in MC Syllabus 29.1b p. 227-231.

V . FUTURE PLANS

For the near future implementations are considered for

. general summation (V.1)

. summation of Chebyshev sums (V.2)

. summation of sums of orthogonal polynomials (V.3)

. two-dimensional IDFT (V.4)

. operators for special matrix-times-vector products (V.5)

. Winograd technique for the improvement of the DFT (V.6)

Anyone who likes to contribute with respect to the above items - or has suggestions with respect to any other item within the C6 chapter - is encouraged to contact the author.

When appropriate we refer to the NUMAL library of the Mathematical Centre for ALGOL 60 implementations.

58

V.1 General summation

Within this context we have for problem (1.1)

$$\sum_{k=\ell}^{\infty} f_k .$$

For a slowly convergent series an Euler transformation with van Wijngaarden strategy can be used; when the terms of the series have the same sign a preliminary transformation (due to Van Wijngaarden) can be applied to transform the series into an alternating one. Apart from those linear transformations a lot of non-linear techniques are available. For a recent survey see Brezinski (1978) with implementations as FORTRAN programs. It requires more research in order to make a more detailed proposal.

Remarks

. A routine Euler is provided in the RR of ALGOL 68.

. In NUMAL implementations are available for an alternating series and for a series with terms of the same sign.

Literature

Brezinski, C. (1978): Algorithmes d'accélération de la convergence. Étude numérique, Paris.

Daniel, J.W. (1969): Summation of a series of positive terms by condensation transformations. Math. Comp., 23, 91-96.

Van Wijngaarden, A. (1965): Course Scientific computing B; process analysis (Dutch). Mathematisch Centrum CR-18.

V.2 The summation of Chebyshev sums

Within this context we have for problem (1.1)

$$S(x) = \sum_{k=0}^{n} a_k T_k(x), \qquad x \in [-1,1]$$

with $T_k(x)$ the Chebyshev polynomial of the first kind of degree k.
A well-known algorithm for the evaluation of this sum is the Clenshaw algorithm, which can easily be understood from

$$S(x) = (1,-x) \sum_{k=0}^{n} \begin{pmatrix} 2x & -1 \\ 1 & 0 \end{pmatrix}^{k} \begin{pmatrix} a_k \\ 0 \end{pmatrix}$$

by applying Horner's rule to the matrix polynomial.

As a special case we have the odd Chebyshev sum

$$S_o(x) = \sum_{k=0}^{n} a_k T_{2k+1}(x).$$

The Clenshaw algorithm for the evaluation of the above sum is easily obtained from

$$S_o(x) = x(1,-1) \sum_{k=0}^{n} \begin{pmatrix} 2T_2(x) & -1 \\ 1 & 0 \end{pmatrix}^{k} \begin{pmatrix} a_k \\ 0 \end{pmatrix}$$

by again applying Horner's rule to the matrix polynomial.

The even Chebyshev sum

$$S_e(x) = \sum_{k=0}^{n} a_k T_{2k}(x)$$

can be reduced to the calculation of S because

$$T_{2k}(x) = T_k(T_2(x))$$

and therefore

$$S_e(x) = S(T_2(x)).$$

The summation of the shifted Chebyshev sum

$$S^*(x) = \sum_k a_k T_k^*(x), \qquad\qquad x \in [0,1]$$

can be reduced to the summation of S because

$$T_k^*(x) = T_k(2x-1),$$

and therefore

$$S^*(x) = S(2x-1).$$

For the above problems it is proposed to implement the dyadic operators with left operand <u>scal</u> x and right operand <u>vec</u> a:

C06CHESUM	for $S(x)$
C06ODDCHESUM	for $S_o(x)$
C06EVECHESUM	for $S_e(x)$
C06SHTCHESUM	for $S^*(x)$.

<u>Remarks.</u>

. Implementations in ALGOL 60 are provided in NUMAL.

. We agree with the remark of Curtiss that the modifications due to Reinsch of the Clenshaw-algorithm are only useful, if we have ϕ available instead of x, with x = cos ϕ. The implementation of Cox and Hayes (1974) is therefore not necessary.

. It has been observed by Newbery (1974), that if the coefficients of the equivalent power sum representation are of the same sign or strictly alternating, then the power sum representation can be used instead of the Chebyshev sum representation, without loss of accuracy and with gain in evaluation speed. If we define the sensitivity for the perturbations in the coefficients $\{a_k\}$ of $S(x)$ by

$$K(S_a(x)) = \sum_k \left| \frac{a_k}{S} \frac{\partial S(x)}{\partial a_k} \right|$$

and $K(S_b)$ analogous for S in the Chebyshev sum representation, then we have under the conditions mentioned by Newbery:

$$\max_{x \in [-1,1]} K(S_a(x)) = \max_{x \in [-1,1]} K(S_b(x)).$$

(An implementation for the transformation of a power sum into a Chebyshev sum or vice versa is provided in NUMAL).

<u>Literature.</u>

Cox, M.G. & J.G. Hayes (1974): Curve fitting: a guide and suite of algorithms for the non-specialist user. NAC Report 26. National Physical Laboratory.

Newbery, A.C.R. (1974): Error analysis for polynomial evaluation. Math. Comp., 28, 789-793.

V.3   The summation of sums of orthogonal polynomials

Within this context we have for problem (1.1)

$$S(x) = \sum_{k=0}^{n} a_k f_k(x)$$

with $f_k(x)$ a function obeying a second order homogeneous recurrence relation. As an important special case we have the summation of orthogonal polynomials. The names of classical orthogonal polynomials and some of their properties and interrelations are given in chapter 22 of Abramowitz and Stegun. Correlated with the names is the standardization. In our opinion it is advantageous to provide, at first, implementations for:

a. a sum of orthogonal polynomials each normalized with the coefficient of the highest power of x equals 1;

b. sums of diverse orthogonal polynomials, named and normalised according to Abramowitz and Stegun.

The implementation under a) is general in the sense, that the appropriate recurrence coefficients, besides the argument x and coefficients, $\{a_k\}$, must be provided by the user; the implementations under b) are recognised by their names and only the argument, the coefficients $\{a_k\}$ and appropriate parameters must be provided by the user.

The algorithms are essential due to Clenshaw, because, if

$$f_{k+1} = \alpha_k(x) f_k + \beta_k f_{k-1}, \qquad k = 1,2,\ldots$$

with initial values f_0 and f_1, are given, then

$$S(x) = (f_0, f_1) \sum_{k=0}^{n} \prod_{j=0}^{k-1} \begin{pmatrix} \alpha_j(x) & 1 \\ \beta_j & 0 \end{pmatrix} \begin{pmatrix} a_k \\ 0 \end{pmatrix}$$

is easily obtained from the Horner-like rule

$$(f_0, f_1) \left(\begin{pmatrix} a_0 \\ 0 \end{pmatrix} + A_0 \left(\begin{pmatrix} a_1 \\ 0 \end{pmatrix} + \ldots + A_{n-2} \left(\begin{pmatrix} a_{n-1} \\ 0 \end{pmatrix} + A_{n-1} \begin{pmatrix} a_n \\ 0 \end{pmatrix} \right) \ldots \right) \right)$$

with

$$A_j = \begin{pmatrix} \alpha_j(x) & 1 \\ \beta_j & 0 \end{pmatrix} .$$

This 'Horner-scheme' can be viewed as an inhomogeneous second order recursion.

For the implementation under a) we propose the dyadic operators

name	left operand	right operand
CO6SUMORTPOL	struct (vec b,c, scal x)	vec a
CO6SUMORTPOL	struct (vec c, scal x)	vec a

The recurrence coefficients b,c are related to those given in table 22.7 of Abramowitz and Stegun by

$$b_k = -a2_k/a3_k \quad , \quad k = 0,1,..$$

$$c_k = a4_k * a1_{k-1}/(a3_k * a3_{k-1}) , \quad k = 1,2,... .$$

For the well-known classical orthogonal polynomials we have

polynomial kind	recurrence coefficients	
	b_k	c_k
Chebyshev (1st kind)	0	$1/2, \quad k = 1$ $1/4, \quad k > 1$
Chebyshev (2nd kind)	0	$1/4$
Legendre	0	$k^2/(4k^2-1)$
Jacobi	$\dfrac{(\beta^2-\alpha^2)}{(\alpha+\beta+2k)(\alpha+\beta+2(k+1))}$	$\dfrac{4(\alpha+1)(\beta+1)}{(\alpha+\beta+2k)^2(\alpha+\beta+3)} , \quad k = 1$ $\dfrac{4k(\alpha+k)(\beta+k)(\alpha+\beta+k)}{(\alpha+\beta+2k)^2((\alpha+\beta+2k)^2-1)} , k>1$
Laguerre	$\alpha +2k + 1$	$(\alpha+k)k$
Hermite	0	$k/2$

For the sum $S(x)$ we have

$$S(x) = \sum_{k=0}^{n} a_k f_k(x) = (b,c,x) \ \underline{C06SUMORTPOL} \ a'$$

with $f_k(x)$ as defined by Abramowitz and Stegun in table 22.7 and

$$a'_k = \prod_{j=0}^{k-1} (a3_j/a1_j) a_k, \qquad k = 0,1,\ldots n.$$

<u>Remarks.</u>

. The summation of polynomials with $b_k = 0$, is catered for in a simple fashion.

. In NUMAL an implementation, heavily based on Gautschi (1968), is provided for obtaining the recurrence coefficients of a general orthogonal polynomial with a positive weight function.

. The used techniques may be applied to the summation of functions, which obey a three-term recurrence relation. However, the numerical stability must be ascertained for every particular case: if either of the solutions of the homogeneous recurrence dominate the solution of the inhomogeneous recurrence, then the problem is unstable, and a modification of the problem by eliminating the dominant solution is necessary; a criterium in terms of the eigenvalues of the matrices A_j and the coefficients $\underline{a}$ is not yet provided.

For the implementations under b) we propose the dyadic operators with right operand <u>vec</u> a and

name	left operand	$f_k(x)$
C06JACSUM	struct (<u>scal</u> α,β,x)	$P_k^{(\alpha,\beta)}$
C06GEGSUM	struct (scal α,x)	$C_k^{(\alpha)}$
C06CHESUM		(see previous paragraph)
C06TSJSUM	<u>scal</u> x	U_k
C06LEGSUM	<u>scal</u> x	P_k
C06LAGSUM	struct (<u>scal</u> α,x)	$L_k^{(\alpha)}$
C06HERSUM	<u>scal</u> x	H_k

Literature

Gautschi, W. (1968): Algorithm 331, Gaussian Quadrature formulas, Comm. ACM, 11.

V.4 Two dimensional IDFT

The computational problem is

$$\sum_{\ell=0}^{n-1} \sum_{k=0}^{n-1} a_{k\ell} \, w_n^{kp} \, w_n^{\ell q}, \qquad p,q = 0,1,\ldots,n-1$$

which can be dealt with:

> for q to n
>
> do   C06EXPSUM a[,q]od;
>
> for p to n
>
> do   C06EXPSUM a[p,]od

where we assumed

$$a_{k\ell} = a[1+k,1+\ell].$$

Remarks.

. Henrici (1979;see introduction) mentions, that application of the DFT-idea direct to a multi-dimensional structure is more efficient. For the two-dimensional case it is not clear, whether it is worth the possible more complicated and more time-consuming bookkeeping. Within the context of the applications an improvement of 10 to 20% in speed is worthwhile.

. Various authors mention the storage problems for large n.

. Two dimensional Fourier transforms with data provided equidistant in the the r,ϕ-plane are desired.

V.5 Operators for special matrix-times-vector products

In this paragraph we consider representations of circulant and Toeplitz matrices.

These representations are used to form matrix-time-vector products fast; and when we can handle the multiplication fast, it is worthwhile to consider the solution of linear equations with Toeplitz matrices. So far we have not found DFT methods for solving general Toeplitz systems of equations although we are aware of the work of Trench (Zohar (1969)), Zohar (1974), Farder (1977), Kailath T. c.s. (1978), and de Meersman (1975).

5.1 Conventions, notations and relations

. We restrict ourselves to $n*n$-matrices

. Circulant matrix

$$C(a) = \begin{pmatrix} a_0 & a_{n-1} & \cdots & a_1 \\ a_1 & a_0 & \cdots & a_2 \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ a_{n-1} & a_{n-2} & \cdots & a_0 \end{pmatrix} \quad , \text{with } a = \begin{pmatrix} a_0 \\ a_1 \\ \cdot \\ \cdot \\ \cdot \\ a_{n-1} \end{pmatrix}$$

. Toeplitz matrix

$$T(a) = \begin{pmatrix} a_0 & a_{-1} & \cdots & & & a_{-(n-1)} \\ a_1 & a_0 & & & & \\ \cdot & & \cdot & & & \\ \cdot & & & \cdot & & \\ \cdot & & & & a_{-1} \\ a_{n-1} & \cdots & & a_1 & a_0 \end{pmatrix} \quad , \text{with } a = \begin{pmatrix} a_0 \\ a_1 \\ \cdot \\ \cdot \\ a_{n-1} \end{pmatrix}$$

$$\underline{a} = \begin{pmatrix} a_0 \\ a_{-1} \\ \cdot \\ \cdot \\ \cdot \\ a_{-(n-1)} \end{pmatrix} .$$

. Hankel matrix

$$H(a) = \begin{pmatrix} a_0 a_1 & \cdots & a_{n-1} \\ a_1 & & \ddots & \\ \vdots & & & \\ \vdots & & & \\ a_{n-1} & \cdots & a_{2n-1} \end{pmatrix}$$

. upper triangular Toeplitz matrix

$$\nabla(a) = \begin{pmatrix} a_0 & a_{n-1} & \cdots & a_1 \\ & a_0 & \ddots & \vdots \\ & & \ddots & a_{n-1} \\ 0 & & & a_0 \end{pmatrix}$$

. lower triangular Toeplitz matrix

$$\Delta(a) = \begin{pmatrix} a_0 & & & 0 \\ a_1 & a_0 & & \\ \vdots & \ddots & \ddots & \\ a_{n-1} & \cdots & a_1 & a_0 \end{pmatrix}$$

. S-matrix

$$
\begin{pmatrix}
& & \cdot & 1 \\
& & \cdot & \\
1 & \cdot & & \\
\end{pmatrix}
$$

. E-matrix

$$
\begin{pmatrix}
0 & 1 & & & \\
& 0 & 1 & & \\
& & \cdot & \cdot & \\
& & & \cdot & 1 \\
1 & & & & 0 \\
\end{pmatrix}
$$
, with E:
$$
\begin{pmatrix}
a_0 \\
a_1 \\
\cdot \\
\cdot \\
\cdot \\
a_{n-1}
\end{pmatrix}
\rightarrow
\begin{pmatrix}
a_1 \\
a_2 \\
\cdot \\
\cdot \\
a_{n-1} \\
a_0
\end{pmatrix}
$$.

. E^--matrix

$$
\begin{pmatrix}
0 & & & & 1 \\
1 & 0 & & & \\
& \cdot & \cdot & & \\
& & \cdot & \cdot & \\
& & & 1 & 0 \\
\end{pmatrix}
$$
, with E^-:
$$
\begin{pmatrix}
a_0 \\
a_1 \\
\cdot \\
\cdot \\
\cdot \\
a_{n-1}
\end{pmatrix}
\rightarrow
\begin{pmatrix}
a_{n-1} \\
a_0 \\
a_1 \\
\cdot \\
\cdot \\
a_{n-2}
\end{pmatrix}
$$.

. W-matrix (or IDFT-matrix)

$$
\begin{pmatrix}
1 & 1 & \cdots & 1 \\
1 & w & \cdots & w^{n-1} \\
\cdot & \cdot & & \cdot \\
\cdot & \cdot & & \cdot \\
\cdot & \cdot & & \cdot \\
1 & w^{n-1} & \cdots & w^{(n-1)^2} \\
\end{pmatrix}
$$
, with $w = e^{2\pi i/n}$.

. D(w)-matrix

$$
\begin{pmatrix}
1 & & & & 0 \\
& w & & & \\
& & \cdot & \cdot & \\
& & & \cdot & \\
0 & & & & w^{n-1}
\end{pmatrix}
$$

$D = D(e^{2\pi i/n})$

$D' = D(e^{\pi i/n})$.

. $\Lambda(a)$-matrix: a diagonal matrix with diagonal elements

$$\Lambda(a)_{ii} = (\overline{W}a)_i$$

. $W\overline{W} = nI$

$WS = \overline{DW}$

$SW = \overline{WD}$

$DWS = \overline{W}$

$WE^{-} = DW$

$EW = WD$

$WSEq = \overline{Wq},$ q a vector

$WE = \overline{DW}$

$E^{-}W = W\overline{D}$

$\Lambda(a) = S\nabla(SEa)S$

$$
a' = \begin{pmatrix}
& 1 & & & \\
\cdot & \cdot & \overline{-1} & & \\
& & \cdot & \cdot & \\
& & & \cdot & \cdot \\
& & & & -1
\end{pmatrix}\overline{D}'a.
$$

. The number of operations of a W-matrix-times-vector is $O(n*\Sigma p_i)$, with

n = $\pi(p_i)$ if we apply FFT-like algorithms.

5.2 Matrix-times-vector products

The representation of a circulant in terms of W, and Toeplitz matrices in terms of circulants is given. Indicated is how the DFT can be used with respect to the matrix-times-vector products. Applications to the multiplication of polynomials are treated, where the treatment of the multiplication of two polynomials in the Chebyshev sum representation is possibly new. No implementations are proposed, because the treatment of the solution of linear systems with Toeplitz matrices is not yet clarified. (For the matrix-times-vector products a set of dyadic operators is a realistic possibility).

LEMMA 1. (Eigensystem of circulant)

$$C(c) = n^{-1} W \Lambda(c) \bar{W}.$$

PROOF. Multiplication of the eigensystem equation

$$Cv = \lambda v$$

by $\bar{W}$, yields for the j-th component

$$(\bar{W}Cv)_j = (\bar{W}c)_j \, (\bar{W}v)_j = \lambda (\bar{W}v)_j.$$

The solution of the transformed equation is

$$\lambda = (\bar{W}c)_j$$

$$\bar{W}v = e_j \qquad \text{(the j-th unit vector)}.$$

The equations for the eigensystem

$$CW = W\Lambda$$

give the factorization

$$C = n^{-1} W \Lambda \bar{W}.$$

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

70

THEOREM 1. (Circulant times vector)

The product can be represented by

$$C(c)b = n^{-1}(W(\Lambda(c)(\bar{W}b))).$$

PROOF. Apply the decomposition given in lemma 1.

Remarks.

. A product of a rectangular circulant times a vector can be obtained by the above formula by padding the shorter of b and c with zeros.

. A circulant times vector is also called a circular convolution and is often denoted by

$$\sum_{i=0}^{n-1} c_{j-i}b_i, \quad j = 0,1,\dots \text{ and } c_k = c_{n+k}, \quad \text{all integer } k.$$

LEMMA 2. (Representation of an upper Triangular Toeplitz matrix as a sum of a circulant and a diagonal similarity transformation of a circulant.)

$$\triangledown (a) = \tfrac{1}{2}\{C(a) + D'C(a')\bar{D}'\}.$$

PROOF.

$$\triangledown(a) = \tfrac{1}{2} \left\{ C(a) + \begin{pmatrix} a_0 & & \cdots & a_1 \\ -a_1 & a_0 & \cdots & a_2 \\ \vdots & & \ddots & \vdots \\ & & & \\ -a_{n-1} & \cdots & -a_1 & a_0 \end{pmatrix} \right\}$$

with the decomposition of the second term

$$
\begin{pmatrix}
a_0 & a_{n-1} & \cdots & a_1 \\
-a_1 & a_0 & \cdots & a_2 \\
\cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot \\
-a_{n-1} & & -a_1 & a_0
\end{pmatrix}
=
$$

$$
\begin{pmatrix}
1 & & & 0 \\
& w & & \\
& & \cdot & \\
0 & & & w^{n-1}
\end{pmatrix}
\begin{pmatrix}
a_0 & wa_{n-1} & \cdots & w^{n-1}a_1 \\
w^{n-1}a_1 & a_0 & & \cdot \\
\cdot & & \cdot & \cdot \\
\cdot & & & \cdot \\
wa_{n-1} & \cdots & & a_0
\end{pmatrix}
\begin{pmatrix}
1 & & & 0 \\
& \bar{w} & & \\
& & \cdot & \\
0 & & & \bar{w}^{(n-1)}
\end{pmatrix}
$$

with $w = e^{\pi i/n}$ and $a' = \begin{pmatrix} a_0 \\ w^{n-1}a_1 \\ \cdot \\ \cdot \\ \cdot \\ wa_{n-1} \end{pmatrix}$.

The result is obtained if we use the notation of 5.1.

Remark. The proof given above is constructive. Once the result is known, a more direct proof can be given by evaluating the sum of the circulants.

THEOREM 2. (Upper triangular Toeplitz matrix times a vector)

$$
\nabla(a)\, b = \tfrac{1}{2}\{C(a) + D'C(a')\bar{D}'\}b
$$

$$
= \{W\Lambda(a)\bar{W}b + D'W\Lambda(a')\overline{WD}'b\}/2n) .
$$

PROOF. Apply the decomposition given in lemma 2.

Remark. For the calculation we need

$\overline{W}b$

$\overline{WD}'b$

$\overline{W}a$ (for $\Lambda(a)$)

$\overline{W}a'$ (for $\Lambda(a')$).

COROLLARY 1. (The coefficients of the product of two (balanced) polynomials in power sum representation).

Let

$$P_{n-1}(x) = \sum_{k=0}^{n-1} a_k x^k \text{ and } Q_{n-1}(x) = \sum_{k=0}^{n-1} b_k x^k$$

then

$$R_{2(n-1)}(x) = P_{n-1}(x) \, Q_{n-1}(x) = \sum_{j=0}^{2(n-1)} c_j x^j$$

with

$$\begin{pmatrix} c_0 \\ c_1 \\ \cdot \\ \cdot \\ \cdot \\ c_{n-1} \\ c_n \\ \cdot \\ \cdot \\ \cdot \\ c_{2n-2} \end{pmatrix} = \begin{pmatrix} a_0 & & & & 0 \\ a_1 & a_0 & & & \\ \cdot & \cdot & \cdot & & \\ \cdot & & \cdot & & \\ \cdot & & & \cdot & \\ a_{n-1} & \cdots & & a_1 & a_0 \\ & a_{n-1} & & & a_1 \\ & & \cdot & & \cdot \\ & & & \cdot & \cdot \\ 0 & & & & \cdot \\ & & & & a_{n-1} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ \cdot \\ \cdot \\ \cdot \\ b_{n-1} \end{pmatrix}$$

or

$$c_j = \sum_{k=0}^{n-1} b_k a^*_{j-k}, \qquad j = 0,1,\ldots,2(n-1)$$

with $a^*_j = a_j$, $j = 0,1,\ldots,n-1$ and $a^*_j = 0$ for $j < 0$, $j \geq n$.

The calculation can be reduced to the upper triangular Toeplitz matrix times vector products

$$\{c_k\}_{n-1}^{2(n-1)} = \nabla (E^- a)\, b$$

$$\{c_k\}_0^{n-1} = S \nabla (SEa)\, (Sb).$$

The vectors in these products are related

$$E^- a \quad \text{with } E^- Sa \qquad (E^- S = SE)$$

and

$$b \qquad \text{with } Sb.$$

For the calculation with $a, b \in \mathbb{R}^n$ some products with W can be written as

$$WE^- a = DWa;$$
$$WE^- Sa = DWSa = \overline{W}a = \overline{Wa}, \qquad \text{for } a \in \mathbb{R}^n;$$
$$Wb;$$
$$WSb = \overline{DWb} = \overline{DWb}, \qquad \text{for } b \in \mathbb{R}^n.$$

COROLLARY 2. (The coefficients of the product of two (balanced) polynomials in Chebyshev sum representation).

Let

$$A_{n-1}(x) = \sum_{k=0}^{n-1} p_k T_k(x) \quad \text{and} \quad B_{n-1}(x) = \sum_{k=0}^{n-1} q_k T_k(x)$$

then

$$C_{2(n-1)}(x) = \sum_{k=0}^{2(n-1)} r_k T_k(x) = \tfrac{1}{2}\{ \sum_{k=0}^{2(n-1)} c_k T_k(x) + \sum_{k=0}^{n-1}{}' (a_k + b_k) T_k(x) \}$$

with

$$
\begin{pmatrix} c_0 \\ c_1 \\ \cdot \\ \cdot \\ \cdot \\ c_{n-1} \\ c_n \\ \cdot \\ \cdot \\ \cdot \\ c_{2(n-1)} \end{pmatrix}
=
\begin{pmatrix}
q_0 & & & & & \\
q_1 & q_0 & & & & \\
\cdot & & \cdot & & & \\
\cdot & & & \cdot & & \\
\cdot & & & & \cdot & \\
q_{n-1} & & & q_1 & q_0 \\
& q_{n-1} & & & q_1 \\
& & \cdot & & \\
& & & \cdot & \\
& & & & q_{n-1}
\end{pmatrix}
\begin{pmatrix} p_0 \\ p_1 \\ \cdot \\ \cdot \\ \cdot \\ p_{n-1} \end{pmatrix}
$$

or $c_j = \sum\limits_{k=0}^{n-1} q^*_{j-k} p_k,$ $\quad j = 0,1,\ldots,2(n-1),$

$$
\begin{pmatrix} a_0 \\ a_1 \\ \cdot \\ \cdot \\ \cdot \\ a_{n-1} \end{pmatrix}
=
\begin{pmatrix}
q_{n-1} & \cdots & q_0 \\
& q_{n-1} & & \cdot \\
& & \cdot & \cdot \\
& & & \cdot \\
& & & q_{n-1}
\end{pmatrix}
\begin{pmatrix} p_0 \\ p_1 \\ \cdot \\ \cdot \\ \cdot \\ p_{n-1} \end{pmatrix}
$$

or $a_j = \sum\limits_{k=0}^{n-1} q^*_{j+k} p_k,$ $\quad j = 0,1,\ldots,n-1,$

$$
\begin{pmatrix} b_0 \\ b_1 \\ \cdot \\ \cdot \\ \cdot \\ b_{n-1} \end{pmatrix}
=
\begin{pmatrix}
q_0 & q_1 & \cdots & q_{n-1} \\
& q_0 & & \cdot \\
& & \cdot & \cdot \\
& & & q_1 \\
& & & q_0
\end{pmatrix}
\begin{pmatrix} p_0 \\ p_1 \\ \cdot \\ \cdot \\ \cdot \\ p_{n-1} \end{pmatrix}
$$

or $b_j = \sum_{k=0}^{n-1} q^*_{k-j} p_k \ (= \sum_{k=0}^{n-1} p^*_{j+k} q_k), \qquad j = 0,1,\ldots,n-1;$

the vectors with asterisk denote the vectors without asterisk padded with zeroes.

<u>PROOF.</u>

From

$$\sum_{k=0}^{n-1} p_k T_k(x) \sum_{k=0}^{n-1} q_k T_k(x) = \tfrac{1}{2} \sum_{k=0}^{n-1} \sum_{\ell=0}^{n-1} p_k q_\ell \{T_{k+\ell}(x) + T_{|k-\ell|}(x)\}$$

$$= \tfrac{1}{2}\{\sum_{k=0}^{n-1} \sum_{\ell=0}^{n-1} p_k q_\ell T_{k+\ell}(x) + \sum_{k=0}^{n-1} \sum_{\ell=k}^{n-1}{}' p_k q_\ell T_{\ell-k}(x) +$$

$$\sum_{\ell=0}^{n-1} \sum_{k=\ell}^{n-1}{}' p_k q_\ell T_{k-\ell}(x)\}.$$

The first term can be represented by substitution of $j = k+\ell$ as

$$\sum_{j=0}^{2(n-1)} (\sum_{k=0}^{n-1} q^*_{j-k} p_k) T_j(x).$$

The second term can be represented by substitution of $j = \ell-k$ as

$$\sum_{j=0}^{n-1}{}' (\sum_{k=0}^{n-1} q^*_{j+k} p_k) T_j(x) \text{ or } \sum_{j=0}^{n-1}{}' (\sum_{k=0}^{n-1} q^*_{j+n-1-k} p_{n-1-k}) T_j(x).$$

The third term can be represented by substitution of $j = k-\ell$ as

$$\sum_{j=0}^{n-1}{}' (\sum_{\ell=0}^{n-1} p^*_{j+\ell} q_\ell) T_j(x) \text{ or } \sum_{j=0}^{n-1} (\sum_{\ell=0}^{n-1} q^*_{\ell-j} p_\ell) T_j(x).$$

<u>Remark.</u>

The calculation can be reduced to

$$\{c_k\}^{2(n-1)}_{n-1} = \nabla(E^- q) p$$

$$\{c_k\}^{n-1}_0 = S\nabla(SEq) Sp$$

$$a = \nabla(E^- q) Sp$$

$$b = \nabla(SEq) p$$

Again the vectors in these products are related

$$E^-q \text{ with } SEq$$

and

$$p \quad \text{with } Sp.$$

For the calculation with $p,q \in \mathbb{R}^n$ some products with W can be written as

$$WE^-q = DWq$$
$$WSEq = \overline{DWEq} = \overline{Wq}$$
$$WSq = \overline{DWq}.$$

<u>LEMMA 3.</u> (representation of a Toeplitz matrix as a difference of a circulant and a diagonal similarity transformation of a circulant

$$T(t) = C((t+E^- St_-)/2) - \dot{D}'C((t-E^- St_-)'/2)\bar{D}'.$$

<u>PROOF.</u>

$$T(t) = \begin{pmatrix} t_0 & t_{-1} & \cdots & t_{n-1} \\ t_1 & t_0 & & \\ \cdot & & \cdot & \\ \cdot & & & \cdot \\ \cdot & & & t_{-1} \\ t_{n-1} & \cdots & & t_0 \end{pmatrix} =$$

$$= C(t) + \begin{pmatrix} t_0 & t_{-1} & \cdots & t_{-(n-1)} \\ & t_0 & \cdot & \cdot \\ & & \cdot & \cdot \\ & & & t_{-1} \\ & & & t_0 \end{pmatrix} - \begin{pmatrix} t_0 & t_{n-1} & \cdots & t_1 \\ & t_0 & \cdot & \cdot \\ & & \cdot & \cdot \\ & & & t_{n-1} \\ & & & t_0 \end{pmatrix}$$

$$= C(t) + \nabla(u) - \nabla(t)$$

with

$$u = \begin{pmatrix} t_0 \\ t_{-(n-1)} \\ \cdot \\ \cdot \\ \cdot \\ t_{-1} \end{pmatrix} \quad = \bar{E} St_- .$$

From the representation of an upper triangular Toeplitz matrix in lemma 2, we have

$$C(t) - \nabla(t) = C(t) - \tfrac{1}{2}\{C(t) + D'C(t')\bar{D}'\}$$

$$= \tfrac{1}{2}\{C(t) - D'C(t')\bar{D}'\}.$$

Therefore,

$$T(t) = \tfrac{1}{2}\{C(t) - D'C(t')\bar{D}'\} +$$

$$\tfrac{1}{2}\{C(E^{-1} St_-) + D'C((\bar{E} St_-)')\bar{D}'\}$$

$$= C((t+\bar{E} St_-)/2) - D'C((t-\bar{E} St_-)'/2)\bar{D}'.$$

Remark. A band Toeplitz matrix with u upper and ℓ lower codiagonals ($\ell+u<n$) can be represented by the sum of a circulant of order n and a lower triangular Toeplitz matrix of order u and an upper triangular Toeplitz matrix of order ℓ.

THEOREM 3. (Toeplitz matrix times vector)

$$T(t)b = \{W\Lambda(t+\bar{E} St_-)\bar{W}b - D'W\Lambda((t-\bar{E} St_{-1})')\bar{W}\bar{D}'b\}/(2n).$$

PROOF.

Apply the factorization of a circulant as given in lemma 1 to the circulants in the representation of a Toeplitz matrix as given in lemma 3.

Remarks.

. For the calculation we need

$$\overline{W}b = \overline{\overline{W}b} \qquad \text{for } b \in \mathbb{R}^n$$

$$\overline{W}D'b = \overline{\overline{W}D'b}, \text{ for } b \in \mathbb{R}^n$$

$$\overline{W}(t+E^-St_-) = \overline{\overline{W}(t+E^-St_-)}, \qquad \text{for } t, \ t_- \in \mathbb{R}^n$$

$$\overline{W}((t-E^-St_-)') = \overline{\overline{W}((t-E^-St_-)')}.$$

. A Hankel matrix times vector can be reduced to the above case because

$$H = ST.$$

. High speed convolution (correlation) is merely a Toeplitz(Hankel)-matrix-times-vector product via the above factorizations.

5.3 Solution of linear systems

The theorems in this paragraph handle the possibility to obtain the solution fast; exceptional cases and algorithmic details are not yet available.

THEOREM 4. (Solution of a linear system with a circulant as matrix)

$$C(c)x = b \iff x = n^{-1}W(\Lambda^{-1}(c)(\overline{W}b)).$$

provided $\Lambda(c)$ is not singular.

PROOF. From theorem 1 we have

$$C(c)x = n^{-1}W\Lambda(c)\overline{W}x = b$$

and therefore

$$x = n^{-1}W\Lambda^{-1}(c)\overline{W}b.$$

Remarks

. Berg (1975) proposed to use the easy solution of a linear system with a circulant matrix C, for a general linear system with matrix A, by splitting

$$A = C-D.$$

$$Ax = b \iff x_{k+1} = C^{-1}b + C^{-1}Dx_k, \qquad k = 0,1,\ldots .$$

with

$$\|C^{-1}D\| < 1.$$

The iteration can be modified for singular C.

. According to regular splittings a wealth of literature has emerged, see e.g. Berman & Plemmons (1974) and Neumann (1976).

THEOREM 5. (Solution of a linear system with an upper triangular Toeplitz matrix).

The solution of a linear system

$$\nabla(a)x = b \iff \begin{pmatrix} \nabla & T \\ & \nabla \end{pmatrix} \begin{pmatrix} x_1 \\ \cdots \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ \cdots \\ b_2 \end{pmatrix}$$

can recursively be reduced to the solution of smaller systems of the same structure and Toeplitz-matrix-times-vector products.

PROOF. Partitioning of ∇ yields the result.

Remarks.

. The normal back substitution

$$x_{n-1} = b_{n-1}/a_0$$

$$x_{n-k} = b_{n-k} - \sum_{j=1}^{k} a_j x_{n-k+j}, \qquad k = 2,3,\ldots,n-1$$

takes

$$O(n^2) \text{ operations.}$$

. The problem of theorem 5 occurs by polynomial division as can be seen from corollary 1 (and 2 with a rank one update)

determine b, say, from

$$\nabla(E^-a) \; b \; = \; \{c_k\}_{n-1}^{2(n-1)} \; .$$

. The problem of theorem 5 occurs by the determination of the inverse of an upper triangular Toeplitz matrix, because the inverse is again an upper triangular Toeplitz matrix (Kung (1973; theorem 2.3), Derr (1971))

$$\nabla(a)x \; = \; e^{(n)} \; \Longleftrightarrow \; \nabla(a) \; \nabla(E^-x) \; = \; I.$$

. Theorem 5 gives the general idea of how to solve this system of equations; the details of the implementation are not yet worked out.

THEOREM 6. (Solution of a linear system with a band Toeplitz matrix with ℓ lower and u upper codiagonals, Berg (1975))
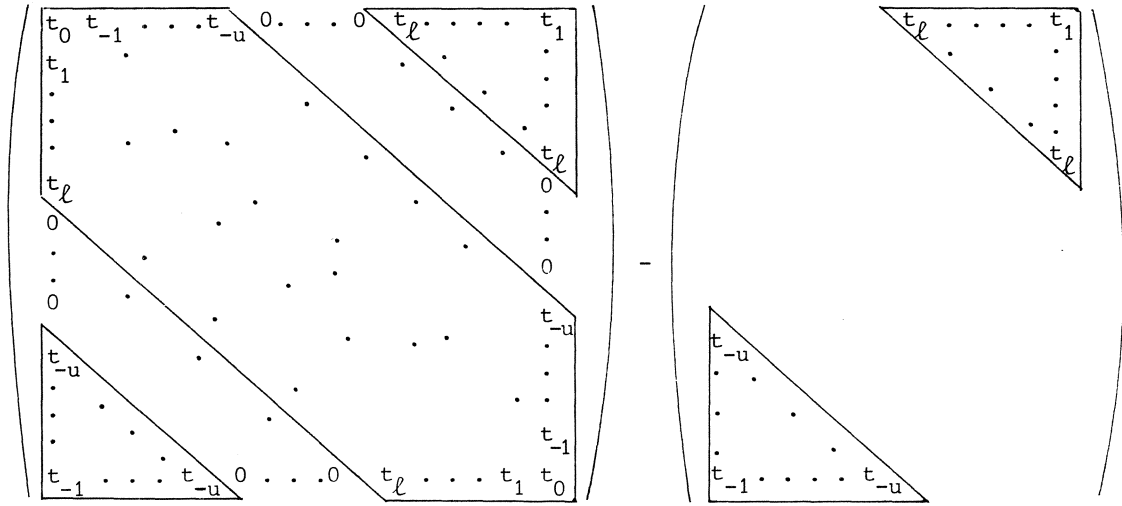
The solution of a band Toeplitz system of equations with ℓ lower codiagonals and u upper codiagonals ($\ell+u<n$) can be reduced to the solution of a system of $\ell+u$ equations, provided the circulant is regular.

PROOF.

The band Toeplitz matrix

$$T = \begin{pmatrix} t_0 & t_{-1} & \cdots & t_{-u} & 0 & \cdots & 0 \\ t_1 & t_0 & & & & & \\ & & & & & & 0 \\ & & & & & & \\ t_\ell & & & & & t_{-u} \\ 0 & & & & & \\ & & & & t_0 & t_{-1} \\ 0 & \cdots & t_\ell & \cdots & t_1 & t_0 \end{pmatrix}$$

can be split into

$$\left(\begin{array}{c} t_0 \; t_{-1} \cdots t_{-u} \; 0 \cdots 0 \; t_\ell \cdots t_1 \\ t_1 \\ \vdots \\ t_\ell \\ 0 \\ \vdots \\ 0 \\ t_{-u} \\ t_{-1} \cdots t_{-u} \; 0 \cdots 0 \; t_\ell \cdots t_1 \; t_0 \end{array}\right) - \left(\begin{array}{c} t_\ell \cdots t_1 \\ t_\ell \\ \\ t_{-u} \\ t_{-1} \cdots t_{-u} \end{array}\right)$$

The linear equations

$$Tx = b$$

can be split into

$$(C - (\; {\textstyle \triangle}^{\triangledown}\;))x = b \; ,$$

and for regular C — the above given circulant —

$$x - C^{-1}(\; {\textstyle \triangle}^{\triangledown}\;)x = C^{-1}b.$$

If we call

$$\hat{x} = (\; {\textstyle \triangle}^{\triangledown}\;)x \qquad \text{(a function of } x_1, \ldots, x_u, x_{n-\ell+1}, \ldots x_n)$$

then we arrive at the $\ell + u$ system of equations

$$x_k - (C^{-1}\hat{x})_k = (C^{-1}b)_k, \qquad k = 1, \ldots, u, \text{ and } n-\ell+1, \ldots, n.$$

The resulting components are obtained from

$$x_k = (C^{-1}(b+\hat{x}))_k, \qquad k = u+1,\ldots,n-\ell.$$

Remarks.

. The above technique is of considerable importance for block band Toeplitz matrices, as arise from discretization of partial differential equations, e.g. the Poisson equation.

. Henrici (1979) considers a more dimensional circular convolution, and its properties under the DFT analogous to the one-dimensional convolution, as the central point. Two-dimensional recurrence relations with constant coefficients and block band Toeplitz matrices are particular cases of a two-dimensional convolution, and by a proper extension circular convolutions (this last aspect simplifies the problem of inversion and is "given" by the problem).

On the other hand it is interesting to consider a block Toeplitz matrix and to think of solutions of a Toeplitz system of equations where the elements are again Toeplitz matrices, so the multiplication and 'division' are performed on operands of type Toeplitz matrix and vector.

Literature

Berg, L. (1975):
Solution of large linear systems with help of circulant matrices. ZAMM, 55, 439-441.

Berman, A. & R.J. Plemmons (1974):
Cones and iterative methods for best least squares solutions of linear systems. SIAM J. Numer. Anal., 11, 145-154.

Derr, L.J. (1971):
Triangular matrices with the isoclinal property. Pac. J. Math., 37, 41-43.

Farden, D.C. (1977):
The solution of a special set of Hermitian Toeplitz linear equations. TOMS, 3, 159-163.

Kailath, T., A. Vieira, M. Morf (1978):

Inversion of Toeplitz operators, innovations and orthogonal polynomials.
SIAM Rev., 20.1, 106-119.

Kung, H.F. (1974);
Fast evaluation and interpolation. Report Carnegie-Mellon University.

Meersman, R. de (1975):
A method for the least squares solution of systems with a cyclic rectangular
coefficient matrix. J. Comp. & Appl. Math., 1, 51-54.

Neumann, M. (1976):
Subproper splitting for rectangular matrices. Lin. Alg. appl., 14, 41-51.

Zohar, S. (1969):
Toeplitz matrices inversion: the algorithm of W.F. French. J. ACM, 16,
592-560.

Zohar, S. (1974):
The solution of a Toeplitz set of linear equations. J. ACM, 21, 272-276.

V.6 Consideration of the Winograd technique for the improvement of the DFT.

The calculation of

$$A_k = \sum_{\ell=0}^{n-1} a_\ell \exp(-2\pi i k\ell/n), \qquad k = 0,1,\ldots,n-1$$

with n prime, can be reduced to

$$A_k - a_0 = \sum_{\ell=1}^{n-1} a_\ell \exp(-2\pi i k\ell/n), \quad k = 1,2,\ldots,n-1.$$

Because n is prime, the numbers $1,2,\ldots,n-1$ form a cyclic group with g as
primitive root, say (Abramowitz & Stegun, p. 827). Therefore a permutation
in the summation, with notation $((x)) = x$ modulo n,

$$\ell \to ((g^\ell)),$$
$$k \to ((g^k))$$

yields

$$A_{((g^k))} - a_0 = \sum_{\ell=0}^{n-2} a_{((g^\ell))} \exp(-2\pi i g^{\ell+k}/n)$$

(Note that $((g^{n-1})) = g^0$.)

This summation is a circular correlation of size n-1; this circulant-times-vector can be calculated via theorem 1 (with Λ precomputed).

Although we consider this the principle of the Winograd technique it requires some more detailed study, whether this is the Winograd technique or not; the necessity of modification of C06FFT depends on the relation between increase of the bookkeeping and lower intrinsic computational complexity.

Literature

Winograd, S. (1978):

On Computing the Discrete Fourier transform. Math. Comp. 32, 179-199.

McClellan, J.H., H. Nawab (1979):

Complex general-n Winograd Fourier Transform Algorithm (WFTA) In: Programs for digital signal processing, IEEE-press, 1.7-1 - 1.7-22.

Silverman, H.F. (1977):

An introduction to programming the Winograd Fourier Transform algorithm. IEEE Trans Acoust. Speech and Signal Processing, vol ASSP-25, 152-165.