STICHTING
MATHEMATISCH CENTRUM
2e BOERHAAVESTRAAT 49
AMSTERDAM

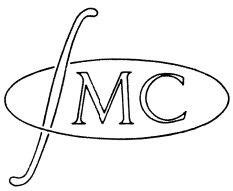
AFDELING TOEGEPASTE WISKUNDE

TW 96

Algebraic operations in ALGOL 60
(a second order problem)

by

R.P. van de Riet



march 1965

Printed at the Mathematical Centre at Amsterdam,49,2nd Boerhaavestraat.
The Netherlands.

The Mathematical Centre, founded the 11th of February 1946, is a non - profit institution aiming at the promotion of pure mathematics and its applications, and is sponsored by the Netherlands Government through the Netherlands Organization for Pure Scientific Research (Z.W.O.) and the Central National Council for Applied Scientific Research in the Netherlands (T.N.O.), by the Municipality of Amsterdam and by several industries.

Introduction

Nowadays almost all time-consuming numerical analysis is done by aid of the fast and accurate computer technique.

Sometimes however, one has to do a considerable amount of work, with the risk of making errors, in elementary algebraic operations with formulae, before one can construct an (ALGOL) program to be used for obtaining numerical results.

In this report we give an account of the investigations about a physico-chemical problem, which will be described in section 1.

In turns out that the analysis of the so-called second order effect results in a fast amount of elementary algebraic operations, which can be done by the computer. The different stages to be followed in the analysis, will be examined in section 2, which results in a calculation scheme.

In section 3, the system is given, by which formulae can be stored within the computer. Moreover the way of input of the formulae will be considered.

The ALGOL procedures based upon the calculation scheme will be illucidated in section 4 and 5.
The best description of them are of course given by their definition in the ALGOL program, called the "Second order program", reproduced in section 6.

The "Second order program" analyses the problem algebraically and gives output in the form of ALGOL statements, punched on a tape. This tape is directly used for the construction of another program, the "Calculation program" (also reproduced in section 6), which is used to obtain numerical results.

Since the variables and constants of the problem are complex quantities (i.e. not real), the form of the statements in the "Calculation program", for calculating these quantities and thus the form of the desired output of the "Second order program" is not obvious, a special section (5) is devoted to these questions.

The main object in publishing this report is to give a demonstration about the way in which a computer can be instructed to do analytical work, by means of a program written in ALGOL 60, a language, which turned out to be extremely suited for this purpose, due to the possibility to construct recursive procedures.

Since we expect that there exists more problems, likely to this particular physico-chemical problem, for which the same method can be used, we described the problem and both programs in a very detailed form, although these programs can not be used for any other problem.

We remark that in [1] a description is given of a program based on an entirely different problem, namely the derivation of a series expansion for a solution of a differential equation, but using the same technique, which is discussed here.

The two ALGOL programs in this report were run on the Electrologica X1 computer of the Mathematical Centre.

1. The Second order problem

In the (x,y,z) coordinate system we study the motion of a fluid. We assume that the phenomena to be considered are independent of the z coordinate.

Let the y coordinate axis be vertically and the x-coordinate axis be horizontally directed.

Let the surface of the fluid be given by $y = \zeta(x)$ and let the fluid be infinitely deep. (i.e. $y = -\infty$)

The fluid and the surface are set in motion by some harmonically vibrating (with period $\frac{2\pi}{\omega}$) oscillator.

The ripples of the surface are damped in the x direction, primarily by the presence of surface active material (surfactant), adsorbed at the surface and dissolved in the bulk fluid.

The interaction between the adsorbed and the dissolved surfactant is due to diffusion.

If we denote the horizontal velocity by u, the vertical velocity by v, the pressure by p and the concentration of the surfactant in the bulk fluid by c, then the motion and diffusion(with diffusion coefficient D) are governed by the following differential equations:

$$(1) \qquad \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial x} + \frac{\eta}{\rho}\Delta u$$

$$(2) \qquad \frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial y} - g + \frac{\eta}{\rho}\Delta v$$

$$(3) \qquad \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0$$

and

$$(4) \qquad \frac{\partial c}{\partial t} = D\,\Delta c - u\frac{\partial c}{\partial x} - v\frac{\partial c}{\partial y}\,.$$

The boundary conditions for $y = -\infty$ are very simple: u and v must be zero and c must be equal to the constant c_0. The boundary condition for the surface $y = \zeta(x,t)$ are very complicated. We define

$$\sigma = \sigma_w - R\,T\,\Gamma_\infty \ln\left(1 + \frac{c}{a}\right) \text{ and } L = \left(1 + \left(\frac{\partial \zeta}{\partial x}\right)^2\right)^{\frac{1}{2}}\,.$$

Let the quantities $\eta, \rho, g, D, \Gamma_\infty, a, \sigma_w, R, T$ and c_0 be measurable constants determining the physico-chemical state, then the boundary conditions are

$$(5) \qquad \frac{\partial \zeta}{\partial t} + u\frac{\partial \zeta}{\partial x} = v$$

$$(6) \qquad -p + \frac{2\eta}{L^2}\left\{\frac{\partial v}{\partial y} - \frac{\partial \zeta}{\partial x}\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial x}\right) + \left(\frac{\partial \zeta}{\partial x}\right)^2 \frac{\partial u}{\partial x}\right\} = \sigma\,\frac{\partial^2 \zeta}{\partial x^2}\,L^{-3/2}$$

$$(7) \qquad 2\eta\frac{\partial \zeta}{\partial x}\left(\frac{\partial v}{\partial y} - \frac{\partial u}{\partial x}\right) + \eta\left(1 - \left(\frac{\partial \zeta}{\partial x}\right)^2\right)\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) = L\left(\frac{\partial \sigma}{\partial x} + \frac{\partial \sigma}{\partial y}\cdot\frac{\partial \zeta}{\partial x}\right)$$

$$(8) \qquad a\,\Gamma_\infty\left(\frac{\partial c}{\partial t} + u\frac{\partial c}{\partial x} + v\frac{\partial c}{\partial y}\right) = -\frac{\Gamma_\infty}{L^2}\,c(c+a)\left\{\frac{\partial u}{\partial x} + \frac{\partial \zeta}{\partial x}\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) + \left(\frac{\partial \zeta}{\partial x}\right)^2 \frac{\partial v}{\partial g}\right\}$$
$$- \frac{D}{L}\left[\frac{\partial c}{\partial y} - \frac{\partial c}{\partial x}\frac{\partial \zeta}{\partial x}\right](c+a)^2$$

For the derivation of these formulae we refer the reader to [2].
The solution of this problem is sought for in the form of Fourier series for the unknown functions u, v, p, c, ζ and σ.

e.g. $u = \sum_{n=0}^{\infty} c_n(x,y) e^{in\omega t}$, when the functions $c_n(x,y)$ should be chosen properly.

Insertion of these series into the differential equations and the boundary conditions and assembling corresponding coefficients of $e^{in\omega t}$, furnishes a method for obtaining the unknown functions c_n and the corresponding coefficients in the series of v, p, c, ζ and σ.

For n = 0 we get the trivial result $\zeta_0 = U_0 = V_0 = 0$.

$u_0 =$, $p_0 = -\rho g y$, $c_0 = c_0$ and $\sigma_0 = \sigma_w$.

For n = 1 the substitution is relatively simple to do and the result is:

$$u_1 = (-ik_1 A_1 e^{k_1 y} - m_1 B_1 e^{m_1 y}) e^{i(\omega t + k_1 x)} \quad \text{with } m_1^2 = k_1^2 + \frac{i\omega\rho}{\eta} \tag{9}$$

$$v_1 = (-k_1 A_1 e^{k_1 y} + ik_1 B_1 e^{m_1 y}) e^{i(\omega t + k_1 x)} \tag{10}$$

$$p_1 = i\rho\omega A_1 e^{k_1 y} e^{i(\omega t + k_1 x)} \tag{11}$$

$$c_1 = E_1 e^{\mu_1 y} e^{i(\omega t + k_1 v)} \quad \text{with } \mu_1^2 = k_1^2 + \frac{i\omega}{D} \tag{12}$$

$$\zeta_1 = \frac{k_1}{\omega} (iA_1 + B_1) e^{i(\omega t + k_1 x)} \tag{13}$$

$$\sigma_1 = -\frac{R\,T\,\Gamma_\infty}{a + c_0} c_1 \tag{14}$$

The quantity k_1 can be determined by the condition that the determinant of the three homogeneous equations for A_1, B_1 and E_1, obtained from the equations 6, 7 and 8, should vanish. A_1, B_1 and E_1 can then be expressed in terms of k_1.

From the differential equations we get for u = 2 in a not too complicated way the following results:

$$(15)\qquad u_2 = (-2ik_1A_2e^{2k_1y} - m_2B_2e^{m_2y} - (m_1+k_1)\psi\, e^{(k_1+m_1)y})e^{2i(\omega t+k_1x)}$$

$$(16)\qquad v_2 = (-2k_1A_2e^{2k_1y} + 2ik_1B_2e^{m_1y} + 2ik_1\psi e^{(k_1+m_1)y})\, e^{2i(\omega t+k_1x)}$$

$$(17)\qquad p_2 = (2i\omega\rho A_2\, e^{2k_1y} + P^{**}\, e^{(k_1+m_1)y})\, e^{2i(\omega t+k_1x)}$$

$$(18)\qquad c_2 = (E_2\, e^{\mu_2 y} + j_1\, e^{(k_1+\mu_1)y} + j_2\, e^{(m_1+\mu_1)y})e^{2i(\omega t+k_1x)}$$

with $P^{**}\psi$, j_1 and j_2 known expressions in A_1, B_1 and E_1 and $m_2^2 =$
$= 4k_1^2 + \frac{2i\omega\rho}{\eta}$ and $\mu_2^2 = 4k_1^2 + \frac{2i\omega}{D}$

$$(19)\qquad \sigma_2 = -\frac{R\,T\,\Gamma_\infty}{a+c_0}\left(c_2 - \frac{1}{2(a+c_0)}\, c_1^2\right)$$

$$(20)\qquad \zeta_2 = Z_2\, e^{2i(\omega t+k_1x)}$$

From the four boundary conditions, the four unknown constants A_2, B_2, E_2 and Z_2 can be determined.
Obviously this is a painstaking job; even the boundary itself in an unknown function.
At this point the fast and accurate computer technique can be introduced.

2. The Calculation Scheme

Before we describe how the computer technique can be used we want to analyse the process for solving this problem.

This amounts in short to: inserting the formulae (9) to (10) into the boundary conditions (5) until (8).
For each boundary condition the coefficients of the several unknown quantities must be assembled and the resulting linear equations in these quantities should be formed.

However, it turned out during the execution of some test programs, that this device can not be used, without saying. The origin of the difficulties is the restriction of the memory capacity of the used X1 computer.

We are forced to split up the boundary conditions in several parts, so that the operations can be done for the different parts seperately. The results obtained from these different parts are assembled afterwards for each boundary condition.
It is obvious that this splitting up gives extra difficulties for the administration.

To clear up the situation we will treat in more detail the first and easiest boundary condition (5).
Although the splitting up is not neccessary for this boundary condition it will be done by way of example.

Let us formulate our purpose:

> Calculate Z_2 as a linear combination of A_2 and B_2.
> (Note that E_2 does not occur in this boundary condition)

We examine the following parts of the boundary conditions

$$B1 : \frac{\partial \zeta}{\partial t} \; ; \; C1 : -v \quad \text{and} \quad D1 : u \frac{\partial \zeta}{\partial x}$$

The "Second order program" which also treats this boundary condition in this way, gives six intermediate results of the calculation in an ordinary notation. These results, marked with the symbol comment are reproduced in the Calculation program after the label CALCULATION .

The process which is followed in the "Second order program" is illustrated by the following Calculation Scheme. (we use a short hand notation: "→" meaning: so that we obtain the formula)

1: Insert the formulae 9 to 20 into B1 → B2

2: Remove the brackets in B2 → B3, so that B3 is a sum of terms, and each term is a product of known or unknown quantities

3: Collect the terms in B3 which are, apart from a numerical factor, the same → B4

(Thus $k_1\omega + i\,\omega k_1 + A_1 = (1+i)k_1\omega + A_1$)

4: Remove the terms in B4 which contain more than two factors $e^{i(\omega t+k_1x)}$ → B5. This intermediate result is produced in the first comment part.

5: Insert the boundary $y = \zeta(x)$ in B5 → B6

(Thus $e^{k_1y} = e^{k_1\zeta} = 1 + k_1\zeta + \frac{k_1^2\,\zeta^2}{2} + \ldots$

the series should be continued as far as is necessary)

6: Remove the brackets in B6 → B7

7: Collect the terms of B7 which are, apart from a numerical factor, the same → B8

8: Remove the terms of B8 which contain more than two factors $e^{i(\omega t+k_1x)}$ → B9. This intermediate result is reproduced in the second comment part

9: Collect the coefficients (if any) of $Z_2\,e^{2i(\omega t+k_1x)}$

10 " " " " " $A_2\,e^{2i(\omega t+k_1x)}$

11 " " " " " $B_2\,e^{2i(\omega t+k_1x)}$

12 Collect the remaining second order terms.

From the results of 9, 10, 11 and 12 the program defines output in the form of ALGOL statements, so that we are actually able to calculate the coefficients of Z_2, A_2, B_2 and the remaining second order terms.

The specific form of these statements will be described later on.

The whole scheme is of course repeated for the remaining two parts of the boundary conditions: C2 and D2.

We get in the same way two sequences of formulae C1,...,C9 and D1,...,D9 The third until the sixth above mentioned comment parts correspond to the formulae C5, C9, D5 and D9 respectively.

Since the process is straight forward and very easy to follow for this simple boundary condition we will not show it in more detail; the reader can easily check the mentioned results given in the Calculation program.

Now, that the problem is decribed and the calculation scheme is given we turn over to the ALGOL program.

3. The Storage and Input of formulae in ALGOL 60

Let us investigate the form of the formulae described above. These formulae are build up with a set of quantities partly known (e.g. ω or k_1) and partly unknown (e.g. A_2), a set of exponentials and a set of complex numbers. The operations which occur in these formulae are the sum, the subtraction, the product, the division and the differentiation with respect to x, y and t.

The subtraction of a quantity will be treated as the sum of the product of this quantity with -1. The division occurs only as the division by the known real quantities ω and $a + c_0$, thus if we introduce the quantities $1/\omega$ and $1/(a+c_0)$ then the division can be turned over into a product.

If the factors L, $\frac{1}{L}$ and $L^{-3/2}$ in formulae 6, 7 and 8 are expanded in power series in $\frac{\partial \zeta}{\partial x}$, then these divisions are also eliminated.

We see therefore that the only essential operations are the sum and the product, assuming that the differentiation is directly carried out.

We can describe this situation by saying that the formulae which occur in our problem lie in an algebraic ring. This ring is composed of the complex number field, to which a set of letters (in ALGOL notation identifiers) is adjungated. These letters are given in the procedure body of AP of the "Second order program" as the first actual parameter of the procedure PONS. There are 44 of them beginning with "zero" and ending with "eiotplkx". The physical meaning of these letters can be inspected from the second actual parameter of PONS, which is a string; as string quotes the Math. Centre versions ¢ and ¤ are used.

Thus e.g. the identifier RTGGAPC stands for the physical quantity RT * GAMMA/(a+c0) or $\frac{R.T.\Gamma^{\infty}}{a+c_0}$; the identifier em1y stands for e↑m1y or $e^{m_1 y}$.

We will call henceforth these 44 letters, simple terms.

The representation of a formula in the ring is of course not unique. The original (physical) formula may be written in any way, in which also derivatives occur.

The set of procedures for storing a formula is constructed in such a way that first, the operations product and sum are binary operations,

second, the brackets around a sum occurring as a multiplicant in a formula are removed and third, a possibly occurring derivative is replaced by a formula equal to this derivative (in other words: the differentiation is carried out).

Example: suppose $\frac{da}{dx} = b+c$ then the formula $a.\frac{da}{dx}$ is stored as ((ab)+(ac)), we placed the brackets to indicate that the operations are binary.

Conclusion: the form of the formulae to be stored may be quite general; the differentiation operator may occur in it but the restriction is that the operations product and sum are used as binary operations.

From the formulae of the before-going sections we see that the complex numbers occur only in a product with a formula which contains at least one simple term. We may also say that the formulae which we consider are elements of the ring but not of the complex number field.
According to the above remarks, a stored formula is a sum of products of simple terms.
To each such product we may attach a complex number, so that we obtain a set of formulae large enough to include the formulae of sections 1 and 2.
The formulae of this set have a form which can be defined in the following way, using Backus notation.

```
< simple term    >::= < identifier >
< complex number >::= (< number > + i < number >)
< simple product > ::= (< simple term >*< simple terms >)|
                       (< simple term >*< simple product >)|
                       (< simple product >*< simple product)>)
< formula > ::= (< formula > + < formula> )|
                < simple term >|
                < simple product >|
                (< complex number >*< simple products >)
```

Let us now describe how the storage is actually performed. Assume that to every formula and to every occurring complex number a non-negative integer called the index is associated.
Let us moreover introduce the two negative integers called PLUS and ST

(in the "Second order program" respectively equal to -1 and -2).
We see that each formula can be characterized by three numbers. One number defining the type (sum, product or simple term) and two numbers defining the two formulae of which the considered formula is built up, if it is not a simple term.
It is therefore not surprising, that we use for the storage of the formulae an integer array which has three array elements in each row. This array will be called H and in the program the declaration of H is H[0:k last, 1:3], where k last must be chosen large enough (see section 6).

For the storage of the complex numbers we use the real array HC, declared by HC[1:kc last, 1:2]; where kc last must also be chosen large enough. A complex number R+iI, with R and I real, which has the index n is stored such that HC[n,1]=R and HC[n,2]=I.

We shall now show, by means of the following table, how a formula with index k is stored in the several cases.

Assume the formulae a and b have indexes i_a and i_b and the complex number C has index n (>1)

	H[k,1]	H[k,2]	H[k,3]	
f = a + b	i_a	PLUS	i_b	
f = a * b	i_a	1	i_b	a and b must be simple products or simple terms HC[n,1] = R(C), HC[n,2] = = I(C).
f = C*a*b	i_a	n	i_b	
f = a	undefined	ST	undefined	a must be a simple term

From this table we see that if H[k,2] is positive then the formula is automatically a simple product to which a complex number with index n is attached in the case H[k,2] > 1, but if we define HC[1,1] = 1 and HC[1,2] = 0, then we can say that in the case H[k,2] = 1, the complex number 1 is attached to the simple product.
A consequence of attaching a complex number to a simple product is that we can not multiply a complex number with a simple term, say s. This difficulty is however easily overcome when we use the simple term "one" which is introduced especially for this difficulty. "one" is used as a unit element in the ring, so that we may form the product of s with "one"

and attach to this product the complex number.
Some times however, the same procedure is also followed if we want to multiply a simple product say s, with a complex number C. This is done in the case that s may not be altered (since it is also used elsewhere in the program), then a new simple product is formed of s and "one" and to this simple product a new complex number is attached. This new complex number is the product of C with the originally to s attached complex number. Note, that we adopt the convention that there is only one complex number attached to a simple product. Thus if we multiply 10(a.b) with 20(c.d) then the result is 200((a.b).(c.d)) and not (10(a.b)).(20(c.d)). It is easily seen however, that it is not allowed to remove the two numbers 10 and 20 from the system, since this can damage other formulae using the simple product 10(a.b), so that we actually store the formula

$$200((\overset{10}{\downarrow}a.b).(\overset{20}{\downarrow}c.d))$$

in which the arrows have an obvious meaning: the numbers 10 and 20 remain in the system but they will not be used in this formula.

The simple term "zero" is used within the differentiating procedure, where the derivative of a constant is set equal to the simple term "zero". For convenience sake the indexes of "zero" and "one" are chosen to be equal to 0 and 1 respectively.

Let us illustrate the description given until so far with an example. We want to store the formula $f = u_1+2i\omega$, where u_1 is given in (9), thus f is equal to

$$-ik_1A_1\, e^{k_1y}\, e^{i(\omega t+k_1x)} - m_1B_1\, e^{m_1y}\, e^{i(\omega t+k_1x)} + 2i\omega.$$

In the "Second order program" the occurring simple terms have the following indexes (see the procedure AP). ω:2, k_1:4, m_1:5, A_1:9, B_1:10, e^{k_1y}:37, e^{m_1y}:38 and $e^{i(\omega t+k_1x)}$:43.
Assume that the complex numbers -i, -1 and 2i have the indexes 2, 3 and 4, then

HC[2,1] = 0, HC[2,2] = -1,
HC[3,1] = -1, HC[3,2] = 0,
HC[4,1] = 0, HC[4,2] = 2.

The formula f may then be stored according to the following table:

k	H[k,1]	H[k,2]	H[k,3]	
44	9	1	37	$A_1 e^{k_1 y}$
45	4	2	44	$-ik_1 A_1 e^{k_1 y}$
46	10	1	38	$B_1 e^{m_1 y}$
47	5	3	46	$-m_1 B_1 e^{m_1 y}$
48	45	2	43	$-ik_1 A_1 e^{k_1 y} e^{i(\omega t + k_1 x)}$
49	47	3	43	$-m_1 B_1 e^{k_1 y} e^{i(\omega t + k_1 x)}$
50	48	PLUS	49	$= u_1$
51	1	4	2	$= 2i\omega$
52	50	PLUS	51	= f

and the index of f is equal to 52.

By aid of this example we can also show the use of the input procedures P(i,j), S(i,j) and PC(a1,a2,i).

P and S store the product respectively the sum of two formulae with indexes i and j and PC multiplies the complex number with real part a1 and imaginary part a2 to a formula with index i. ü1 and f are stored by the following statements

```
ü1: = P(S(PC(0,-1, P(k1, P(A1, ek1y))),
         PC(-1,0, P(m1, P(B1, em1y)))), eiotplkx);
f: = S(ü1, PC(0,2, OMEGA));
```

The correspondence between these statements and the formulae for u_1 and f is easily seen.

It is of great importance that this correspondence is direct, since the translation of the formula written in ordinary notation into statements of the above kind, is a source of errors.

We shall now describe in more detail the procedures P, S and PC.

It should be remarked that the non-local integers k and kc are used as pointers for the arrays H and HC indicating the next free places in these arrays.

The integer procedure P(i,j) stores the product of two formulae with indexes i and j. The index of the stored result is assigned to P itself. Notice that we said: P stores the product of two formulae and not: two simple products. In storing a formula, P removes the brackets around a sum, occurring as a factor. If i and j are indexes of simple products then P attaches the product of the two complex numbers attached to the multiplicants, to the newly formed simple product. However, the already attached complex numbers are not removed since as is already mentioned above, it is possible that one or both simple products are used somewhere else in the program.

Of course, attaching the product of the two complex numbers to the product of i and j is not necessary, but our definition of a formula requires it and it has the advantage that we can directly get the attached complex number of a simple product.

The integer procedure S(i,j) stores the sum of the formulae with indexes i and j. The index of the result is assigned to S itself.

The integer procedure PC(a1, a2, i) stores the product of the complex number, with real part a1 and imaginery part a2, with the formula with index i. The index of the stored formula is assigned to P itself. If i corresponds to a sum, then the complex number is multiplied with both summands and the resulting formula is a new sum with an index unequal to i. However, it is possible that the formula with index i occurs only once in the process, so that we may change it without disturbing the future calculation; and this means saving of storage space. We may ask therefore, when it is not allowed to change a formula and how this can be seen from the index. The answer to this question is very easy as we will see. The set of formulae which we encounter in our system can be divided in two sets. One set consists of the formulae such as u_1, u_2, v_1, v_2, etc., which are used in all the boundary conditions, these formulae will be called basic. It is therefore obvious, that these formulae are stored earlier then the formulae corresponding to the boundary conditions. The way in which the

storage of the formulae is performed, is such that the index of consecutively stored formulae, increases. Thus, the indexes of the basic formulae are lower than the indexes of the other formulae and we can indicate a sharp point, for which the non-local integer K is used, below which a formula is basic; i.e. if $i \leq K$ then the formula with index i is basic, and may not be changed.

PC investigates, by means of this device, if a formula is basic or not, if it is basic a completely new formula is constructed and if it is not basic the formula is changed itself (i.e. the formula keeps the same index i, but the data given in H[i,1], H[i,2] and H[i,3] are properly changed).

Both procedures P and PC use the integer procedure CP(i,j) which calculates the product of two complex numbers with indexes i and j. The result is stored in HC, the index of this result is assigned to CP itself.

We shall now deal with the representation of the simple terms". The indexes of them are of course a priori arbitrary. But for short-cutting several procedures we order the values of the integers ek1y, em1y, emu1y, ek2y, em2y, emu2y and eiotplkx such, that this sequence is increasing with succeeding integers. The same restriction is posed on the sequence k1, m1, mu1, m2 and mu2. Moreover the integer eiotplkx is the largest of all indexes of simple terms.

It is perhaps superfluous to say that the two mentioned sequences of integers correspond to the indexes of the simple terms $e^{k_1 y}, \ldots, e^{i(\omega t + k_1 x)}$ and $k_1, \ldots, \mu_2$.

We have two kinds of simple terms in our systems namely, a simple term corresponding to a real quantity (such as OMEGA or ω) and a simple term corresponding to a complex quantity (such as k1).

The information about the two kinds is stored in H[k,1] where k is the index of the simple term. When the simple term is real then H[k,1] = -2 else H[k,1] = 1.

The procedure AP(i) defines, when $i < 0$ the values of the indexes of the simple terms. It is easily seen that this is done in correspondence with the above remarks about the ordering.

When however i $\geq$ 0, then AP defines the output form of the simple term i, this output is given between the string quotes in the procedure PONS.

The M.C. standard procedure PUTEXT1(st) cares for the punching of the string st in a paper tape.

4. Procedures for more detailed investigations

Until so far the description of the system was rather general. There are much more problems for which we can use the same procedures P, S, PC and AP (with other simple terms of course).

The representation of a formula in the array H is such, that the input procedures can be made very simple. A construction of a procedure for differentiating a formula can also be easily based upon this representation; we do this for the procedure DIFF(n,i), but using also, for shortness sake, the special ordering of the simple terms as described before. The integer procedure DIFF(n,i) differentiates a formula with index i with respect to x, y or t dependent on whether n equals 1, 2 or 3 resp. The index of the stored formula is assigned to DIFF itself.

The representation in the array H is very suited for the input and differentiation procedures, but not for more detailed investigations of a formula, therefore we introduce another representation.

As we have seen, each formula is a sum of, say, l_0 simple products. Let these be ordered from 1 to l_0. The i-th simple product is a product of, say, l_i simple terms. Let the indexes of these simple terms be given by

$$a_{i,1}, \ldots, a_{i,l_i} \qquad \text{with } i = 1, \ldots, l_0.$$

We can store the numbers $l_0, l_1, \ldots, l_{l_0}$ in the integer array L[0:N] (where N has to be chosen large enough, in our case N = 50 is sufficient). We set $L[i] = l_i$ ($i = 0, \ldots, l_0$). The index $a_{i,j}$ can be stored in the integer array a, declared by a[1:L[0],0:Max] with Max = maximum L[i], and we set $a[i,j] = a_{i,j}$, for $i > 0$. Moreover the index of the complex number, attached to the i-th simple product, is assigned to a[i,0].

The transfer of a formula with index i from the representation in the array H to the representation in the arrays a and L is governed by the procedure BT(i,a,L,fi).

This procedure should be called twice, the first time with the Boolean fi = false (then the arraybounds of a i.e. the array L is calculated) and the second time with fi = true (then the array a is calculated).

The redundant factors "one" and terms "zero" are removed by the procedure BT. It is used within the procedure SUBCALC.

We shall now discuss the other procedures in more or less detail.

4.1. The procedure OUTPUT1(L,a) punches a formula represented in the arrays a and L, in the ordinary notation, in the output paper tape.

4.2. The procedure SUBCALC(i,c) regulates the different stages of the calculation scheme of section 2.
The stages 3, 4, 7 and 8 are carried out by the procedure CAL, the stage 5 is carried out by the procedure SUBSTITUTE and the stages 9 to 12 are carried out by the procedure PUN.

4.3. The procedure CAL(a,L) rearranges the array in such a way that $a[i,j] \leq a[i,j+1]$, for $j = 1, \ldots, L[i]-1$.
Simple products consisting of the same simple terms, apart from a numerical factor, are summed.
The array a is rearranged another time, such that simple products with the attached complex number equal to zero, or containing more than two factors $e^{i(\omega t+k_1 x)}$, are dropped.

4.4. The procedure SUBSTITUTE(fi,a,L) substitutes, when fi = true the boundary $y = \zeta(x)$ in the formula, represented in the array a and L. When fi = false the formula is not changed.
The possibly new formula is stored in the array H and possibly new introduced brackets are removed. With fi = false SUBSTITUTE transfers the formula from the representation in a and L into the representation in H.

4.5. The description of the procedures PUN and PROD is given in the course of the following section.

5. Computation with complex numbers

Before we can describe the procedures PUN and PROD, we have to consider the way in which the output is desired.

Since we are interested in numerical results, an ALGOL program (the "Calculation program" reproduced in section 6), based upon the formulae derived with the here-described method, has to be constructed.

These formulae are built up with complex numbers, it is therefore not obvious how to treat them in ALGOL. Let us, by way of example, want to calculate x from the formula $x = \frac{(a + b)c}{d}$.
If a, b, c and d are real numbers this calculation is carried out by the statement x := (a + b) * c/d, assuming that a, b, c, d and x are declared real and that a, b, c and d have already got values.
Now however a, b, c and d are complex numbers.

We constructed the procedures P, Q, S, T, J, PRC and U to be able to write down one statement which effectuates the calculation of x.

In this example the statement would be U(x,Q(P(S(T(a),T(b)),T(c)),T(d))) in which the complex numbers constituting the statement must be stored in the real arrays x, a, b, c, d [1:2].
The real and imaginary parts are stored in the array elements with index 1 and 2 respectively.

The analogy with the expression for real numbers becomes apparent, if we remark that the procedures P, Q and S are used for the calculation of a product, a quotient and a sum respectively and the procedure U for assigning the calculated result to x in this case. The role of T will be described later.

For the calculation we use the real array H[1:N,1:2], where N should be chosen large enough (it can be seen from the following that N+1 should be chosen equal to the maximum number of right-handed brackets, placed one after another in the relevant statements).

We have chosen the letter H for the array which is perhaps somewhat confusing, since the same letter is used for the array in which formulae are stored. It has however the advantage that the correspondence of both

types of operations, storage of formulae and computations with complex numbers, is showed very well.

In the array H the intermediate results of the calculations are stored and the final results are stored, with the aid of the procedure U, in the particular arrays representing the several complex numbers (x in the example above). This is in contrast to the use of the array H for representing formulae; where the results themselves are stored in H.

Before we can calculate with the complex numbers they have to be brought in the array H. This is done by the integer procedure T(a). T stores the contents of the array a (representing the complex number a, such that a[1] and a[2] are equal to the real and imaginary part respectively), into the next free places of the array H, say H[k,1] and H[k,2]; the integer k is assigned to T itself.
The mentioned integer k is a non-local integer of the program and indicates at any time the next free places of H. At the beginning of the program k must be set equal to zero. During the execution of a computation, k augments and diminishes automatically, so that at the end of a computation k becomes equal to zero again.

The definition of the integer procedures P, Q, S, PRC and J in the "Calculation program", is such that the index of the place in H, where the result of the calculation is stored, is assigned to the procedure identifiers themselves.

We remark that the same way is followed for the integer procedures P, S, PC and DIFF of the "second order program". This is of course a consequence from the fact that the administrations of the two processes, storage of formulae and computation with complex numbers, are identical.

We shall now give the meaning of the procedures PRC(a,i) and J(a1,a2); PRC multiplies the complex number, stored in H[i,1] and H[i,2], with the real number a; J stores the complex number a1 + ia2 in the array H (where a1 and a2 are real numbers).

Besides the procedures mentioned we use the procedure FLOPC, defining the output of a complex number.

The procedures U1 and Sum are especially made for the connexion with the "Second order program".

Before we can describe these procedures we have to return to the "Second order program".

In section 2 we discussed the way in which the boundary conditions are split up. Each part of it gives a formula, via the stages 1 to 8 of the Calculation scheme, from which the desired output can be obtained.

Evidently this output (ALGOL statements) is given in the form discussed above.

Let us take a look, for instance, at the coefficient of

$$Z_2 e^{(\omega t + k_1 x)} = Z2 \quad e\uparrow i(\text{OMEGA } t + k1x)e\uparrow i(\text{OMEGA } t + k1x)$$

in formula B9 (see the second comment part behind the label CALCULATION in the "Calculation program") this coefficient is $(0 + 2i)$OMEGA.

The numerical value of this coefficient is calculated by the statement

PRC(OMEGA, J(0,2)).

(Note that OMEGA is a real number).

The result of the calculation is stored in the array H, but since H is used for intermediate results only, we have to extract this result from H and store it somewhere else. For this purpose we use the array elements of H with index from 51 to 100 (this is possible since the first 50 places of H are enough for the calculation).

Thus the complex number is restored in H and gets an index between 51 and 100. Of course this index should be stored itself somewhere, since we have to remember that the calculated number is a coefficient of Z_2.

Therefore we introduce the integer arrays

CZ2, CE2[1:1], CA2, CB2[1:4], C2[1:28]

and we store the index of the coefficient of Z_2 in the array element CZ2[1]. (It is the first coefficient of Z_2 which we encounter).

Evidently, indexes of coefficients of E_2, A_2 and B_2 are stored in the arrays CE2, CA2 and CB2 resp., and the index of a remaining second order term is stored in C2.

The transfer of the intermediate result of the calculation (stored in H with index lower than 51) to the array elements of H with indexes greater than 50, and the storage of the index into one of the arrays CZ2, CE2, CA2, CB2 or C2 is done by the procedure U1 (in the "Calculation program" of course).

In the example above, the procedures PUN and PROD (of the "Second order program") define the following output

U1(CZ2[1], PRC(OMEGA, J(+0, $+.200_{10}$ + 1)));

(in which $+.200_{10}$ + 1 means simply +2).
For the sake of illustration we give the following tabel, from which the effect of this statement can be inspected, if we assume that ω = OMEGA = 300.

	k	H[k,1]	H[k,2]	
J	1	0	2	
PRC	1	0	600	
U1	51	0	600	and CZ2[1] = 51.

When the "Second order program" comes across another coefficient of Z2 then it should produce output of the form U1(CZ2[2], ... Therefore we have to administrate how many times a coefficient of Z2 did occur already in a boundary condition. this is done with the help of the integer array c occuring as a formal parameter in the procedures SUBCALC and PUN. The actual parameter is the integer array cc[1:8]. From the definition of the procedure PUN it can be seen that c[1], ..., c[8] correspond to the coefficients of A1, B1, E1, A2, B2, Z2, E2 and the remaining second order terms respectively.
(Remark: The "Second order program" was also used to check the first order solution, therefore A1, B1 and E1 do also occur here. For shortness sake, however, we did not reproduce the output of the first order terms and the reader may thus assume that XEEN(2) ≠ 2 (7-th line of the procedure PUN)).

The administration is very simple.
First all c[i] are set equal to zero and when the procedure PUN comes

across a second order term, the corresponding array element of c is augmented by one; in the above case concerning a coefficient of Z2, c[6] is augmented by one.

We are now able to describe the procedures PUN and PROD of the "Second order program" more accurately.

PUN(c,a,L) investigates a formula represented in the arrays a and L according to stages 9 to 12 of the Calculation scheme of section 2. (Of course the letter Z2 can be changed in the letter E2).

Each row of the array a is examined with respect to the type of it. The relevant output is given i.e. the "heading" of an ALGOL statement (in the above example: "U1(CZ2[1],").

Then the further output is defined by the procedure PROD(i,a,L). This procedure gives the output of the i-th row of the array a, this row is already slightly altered by the procedure PUN, which has set equal to one the array elements, representing simple terms which may not occur in the output, e.g. the simple terms A2 or eiotplkx.

PROD investigates, if the simple term represented by an array element, is possibly a real quantity, then output is given in theform: "PRC(" followed by the output of this simple term and a comma.

When the simple term is a complex quantity then PROD firstly produces the output "P(T(", secondly it investigates if this simple term does occur more than once. If this is the case, then the output of the simple term is followed by a number, indicating the number of times this simple term occurs.

When the simple term occurs only once, then the output is given of this simple term only.

Finally, PROD gives the output of the complex number attached to the simple product and stored in the array elements HC[a[i,0],1] and HC[a[i,0],2] in the form "J(...,...)", where the dots must be replaced by the relevant numbers.

Example: let the i-th row of the array a be given by

a[i,0] = 2, a[i,1] = 2 (OMEGA), a[i,2] = 4(k1), a[i,3] = 5 (m1), a[i,4] = 5 and a[i,5] = 1 with HC[2,1] = 1 and HC[2,2] = .5 (representing the complex number 1 + 1/2 i), then PROD produces with the aid of the

procedure AP the following output:

"PRC(OMEGA, P(T(k1), P(T(m1 2), J(+.100$_{10}$ + 1, +.500$_{10}$ + 0))))".

When the considered row is e.g. a 3rd coefficient of A2 then the procedure PUN has already given the preceeding output "U1(CA2[3]," and it closes the output with ");".

A consequence of doing things as described above is that we have to introduce in the "Calculation program" besides the array m1 also the array m12, m13, k12, k13, k14, etc.
These arrays representing integral powers of m1, k1, etc. should be calculated beforehand in the "Calculation program". The advantage is obvious: the "Calculation program" becomes shorter and less time-consuming.

We have seen, how the relevant output of a formula is produced. This formula is a part of the boundary condition.
When the output of all the different parts of the boundary condition is given, the several results should be assembled.
Let us see how this is done for the first boundary condition (5), which is split up in three parts. The first part gives rise to one CZ2 term, the second part to one CA2, one CB2 and five C2 terms and the third part to another four C2 terms.
The procedures PUN and PROD produced the output by which the "Calculation program" can calculate these several terms. The results of these calculations are (indirectly) stored in the arrays CZ2, CA2, CB2 and C2. These results must be summed, so that we can solve the equation for Z2.
This equation is

$$Z2 = CZ2A2 * A2 + CZ2B2 * B2 + COZ2$$

where the coefficients CZ2A2, CZ2B2 and COZ2 can be calculated from the already stored results.

COZ2 is firstly set equal to the sum of the terms belonging to CZ2. The "Second order program" gives therefore the following output: U(COZ2, Sum(1, CZ2)); in which the, until now, undefined procedure Sum occurs. The meaning of Sum is now evident: Sum calculates the sum of the

complex numbers whose indexes are stored in CZ2, and via the procedure U this sum is stored in the array COZ2.
Generally, Sum(i,a) stores into the next free places of the array H, the sum of the complex numbers with indexes given by a[1], ..., a[i].
The coefficients CZ2A2, CZ2B2 and COZ2 are now calculated by the statements right after the label ASSEMBLAGE in the "Calculation program".
These statements are produced by the "Second order program" with the help of the procedure PU.

One final remark should conclude this section. After the investigations of the first boundary condition, output is given in such a form that the "Calculation program" can calculate the coefficients CZ2A2, CZ2B2 and COZ2. Therefore we can set henceforth Z2 equal to the formula CZ2A2 * A2 + CZ2B2 * B2 + COZ2 so that in the calculation of the following boundary conditions Z2 does not occur anymore.
The same procedure is followed for the second boundary condition (6) from which E2 can be calculated in terms of A2 and B2.
In this way the boundary conditions (7) and (8) constitute two linear equations in A2 and B2. These equations are solved in the "Calculation program" after the last comment. The ALGOL statements for solving these equations are not produced by the "Second order program".

6. The ALGOL programs

In this section we give the "Second order program" and the "Calculation program".
In both programs a set of procedure identifiers is used which are not declared. This set belongs to the set of standard functions for the Mathematical Centre ALGOL system.
We shall describe them shortly.

XEEN(i) : an integer procedure assigning to its identifier a number which can be brought into the machine by the console.

PUTEXT1(string) : a procedure, punching the actual string on the output paper tape.
the symbols ≠ and ≠ are the M.C. representations of string quotes.

PUTEXT1(string) : is most easily described by:
procedure PUTEXT1(string); string string;
begin PUTEXT1(≠'≠); PUTEXT1(string); PUTEXT1(≠'≠) end

PU7BIT(i) : a procedure punching the value of i ($0 \leq i \leq 127$) as a heptade on the output paper tape.

PUNLCR : a procedure punching a new line carriage return symbol on the output paper tape.

PUSPACE(i) : a procedure punching n space symbols on the output paper tape.

RUNOUT : a procedure punching a piece of blank output paper tape.

FLOP(i,j,n) : a procedure punching on the output paper tape the number n in floating point representation; i significant decimals behind the comma and j decimals of the exponential part.

ABSFIXP(i,j,n) : a procedure punching on the output paper tape the absolute value of the number n in fixed point representation; i decimals before and j decimals behind the comma.

read : a real procedure assigning to its identifier the value of a number punched on the input paper tape.

Besides these procedures the following symbol is used : for $\frac{a}{b}$.
Some data are perhaps of interest.
The calculation times for the "Second order program" and the "Calculation program" were about one hour and $1\frac{1}{2}$ minute respectively. Both programs had to be cut in two parts.
For the "Second order program" we chose k last and kc last (the lenghths of the arrays H and HC) equal to 352 and 96 resp. It turned out that the first 310 and 60 places of H and HC were actually needed.
The number of unused X1 storage words was about 300 (i.e. about 3 percent of the total).

We may conclude therefore that the memory capacity was rather critical.

```
begin comment Second order program;
  integer kclast,klast; kclast:= XEEN(1024x1023)/1024; klast:= XEEN(1023);
  begin integer k, kc, K, KC, u1, v1, u2, v2, u, v, k1, m1, mu1, m2, mu2,
    CZ2A2, CZ2B2, ek1y, em1y, emu1y, em2y, ek2y, emu2y, eiotplkx, A1, B1,
    E1, A2, B2, E2, Z2, Z1, Z, zero, one, ST, PLUS, OMEGA, EGO, PSI,
    PST, CST1, CST2, COZ2, CE2A2, CE2B2, COE2, RHO, RTGGAPC,
    EGAPC, ETA, D, g, CNPA, GCCPA, GTCPA, AG, ekm2iok, p1, p2, cst,
    c1, sigma0, sigma1, sigma2, z1, z2, c2;
    array HC[1:kclast,1:2]; integer array H[0:klast,1:3];
    procedure AP (i); value i; integer i;
    begin switch S:= S0, S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12,
      S13, S14, S15, S16, S17, S18, S19, S20, S21, S22, S23, S24, S25, S26,
      S27, S28, S29, S30, S31, S32, S33, S34, S35, S36, S37, S38, S39, S40,
      S41, S42, S43; procedure PONS (j,st); integer j; string st;
      if i < 0 then
      begin j:= k:= k + 1; H[k,1]:= 1; H[k,2]:= ST end else
      begin PUTEXT1(st); goto END end;
      if i < 0 then
      begin k:= -1; kc:= 1; K:= 0; KC:= 1; PLUS:= -1; ST:= -2; HC[1,1]:= 1;
        HC[1,2]:= 0; goto S0
      end else goto S[i+1]; S0: PONS(zero,<0>);
      S1: PONS(one,<1>);                S2: PONS(OMEGA,<OMEGA>);
      S3: PONS(EGO,<OMEGA^(-1)>);       S4: PONS(k1,<k1>);
      S5: PONS(m1,<m1>);                S6: PONS(mu1,<mu1>);
      S7: PONS(m2,<m2>);                S8: PONS(mu2,<mu2>);
      S9: PONS(A1,<A1>);                S10: PONS(B1,<B1>);
      S11: PONS(E1,<E1>);               S12: PONS(A2,<A2>);
      S13: PONS(B2,<B2>);               S14: PONS(E2,<E2>);
      S15: PONS(Z2,<Z2>);               S16: PONS(CZ2A2,<CZ2A2>);
      S17: PONS(CZ2B2,<CZ2B2>);         S18: PONS(PSI,<PSI>);
      S19: PONS(PST,<PST>);             S20: PONS(CST1,<CST1>);
      S21: PONS(CST2,<CST2>);           S22: PONS(COZ2,<COZ2>);
      S23: PONS(CE2A2,<CE2A2>);         S24: PONS(CE2B2,<CE2B2>);
      S25: PONS(COE2,<COE2>);           S26: PONS(RHO,<RHO>);
      S27: PONS(RTGGAPC,<RTxGAMMA/(a+c0)>);
      S28: PONS(EGAPC,<1/(a+c0)>);      S29: PONS(ETA,<ETA>);
      S30: PONS(g,<g>);                 S31: PONS(CNPA,<(a+c0)>);
      S32: PONS(GCCPA,<GAMMAxc0x(a+c0)>);
      S33: PONS(GTCPA,<GAMMAx(2xc0+a)>); S34: PONS(AG,<GAMMAxa>);
      S35: PONS(D,<D>);                 S36: PONS(sigma0,<SIGMA>);
      S37: PONS(ek1y,<e^k1y>);          S38: PONS(em1y,<e^m1y>);
      S39: PONS(emu1y,<e^mu1y>);        S40: PONS(ek2y,<e^k2y>);
      S41: PONS(em2y,<e^m2y>);          S42: PONS(emu2y,<e^mu2y>);
      S43: PONS(eiotplkx,<e^i(OMEGA t + k1 x)>);
    END: end;
    procedure comment;
    begin integer i; for i:= 122,14,115,14,70,14,84,14,84,14,117,14,69,14,35,16
      do PU7BIT(i)
    end;
    procedure PS; begin PUNLCR; PUSPACE(8) end;
```

```
comment Continuation of Second order program;
integer procedure P (i,j); value i,j; integer i,j;
if i = 0 ∨ j = 0 then P:= 0 else
if H[i,2] = PLUS then P:= S(P(H[i,1],j),P(H[i,3],j)) else
if H[j,2] = PLUS then P:= S(P(i,H[j,1]),P(i,H[j,3])) else
begin P:= k:= k + 1; H[k,1]:= i; H[k,3]:= j; H[k,2]:= if H[i,2] < 1 then
  (if H[j,2] < 1 then 1 else H[j,2]) else if H[j,2] < 1 then H[i,2] else
  CP(H[i,2],H[j,2])
end;
integer procedure S(i,j); value i,j; integer i,j;
begin S:= k:= k + 1; H[k,1]:= i; H[k,2]:= PLUS; H[k,3]:= j end;
integer procedure CP(i,j); value i,j; integer i,j;
begin real a; CP:= kc:= kc + 1; a:= HC[i,1] × HC[j,1] - HC[i,2] × HC[j,2];
  HC[kc,2]:= HC[i,1] × HC[j,2] + HC[i,2] × HC[j,1]; HC[kc,1]:= a
end;
integer procedure PC (a1,a2,i); value i; real a1, a2; integer i;
if H[i,2] = PLUS then
begin if i < K then PC:= S(PC(a1,a2,H[i,1]),PC(a1,a2,H[i,3])) else
  begin H[i,1]:= PC(a1,a2,H[i,1]); H[i,3]:= PC(a1,a2,H[i,3]); PC:= i end
end else
begin if i < K ∨ H[i,2] = ST then i:= P(one,i); PC:= i; if i = 0 then
  goto END; kc:= kc + 1; HC[kc,1]:= a1; HC[kc,2]:= a2; if H[i,2] > 1 then
  begin kc:= kc - 1; H[i,2]:= CP(kc+1,H[i,2]) end else H[i,2]:= kc;
END: end;
integer procedure DIFF (n,i); value n,i; integer n,i;
begin integer j; integer array diff[1:2];
  procedure DIFFP(i); value i; integer i;
  if H[i,2] = ST then
  begin if i ≥ ek1y ∧ (n ≠ 2 = i = eiotplkx) then
    begin j:= j + 1; diff[j]:= i end
  end else begin DIFFP(H[i,1]); DIFFP(H[i,3]) end;
  if H[i,2] = PLUS then DIFF:= S(DIFF(n,H[i,1]),DIFF(n,H[i,3])) else
  begin j:= 0; DIFFP(i); DIFF:= P(if j = 0 then 0 else if n = 2 then
    (if j = 2 then S(diff[1] - ek1y + k1, diff[2] - ek1y + k1) else
    if diff[1] = ek2y then PC(2,0,k1) else diff[1] - ek1y + k1 - diff[1];ek2y)
    else if n = 1 then (if j = 1 then PC(0,1,k1) else PC(0,2,k1))
    else (if j = 1 then PC(0,1,OMEGA) else PC(0,2,OMEGA)),i)
end end;
procedure OUTPUT1(L, a); integer array L, a;
begin integer i, j; for i:= 1 step 1 until L[0] do
  begin PS; if a[i,0] > 1 then
    begin PUTEXT1(≮(≯); FLOP(3,1,HC[a[i,0],1]); PUTEXT1(≮ + i×≯);
      FLOP(3,1,HC[a[i,0],2]); PUTEXT1(≮)≯)
    end; for j:= 1 step 1 until L[i] do
    begin AP(a[i,j]); PUTEXT1(≮ ≯) end;
    if i ≠ L[0] then PUTEXT1(≮+≯)
  end; if L[0] = 0 then PUTEXT1(≮0≯)
end;
```

```
comment Continuation of Second order program;
procedure BT (i,a,L,fi); value i; integer i; Boolean fi; integer array a,L;
begin integer l, l1; procedure BT1 (i); value i; integer i;
  if H[i,2] ≠ PLUS then
  begin l:= l + 1; l1:= 0; OT(i); if l1 > 0 then
    begin L[l]:= l1; if fi then a[l,0]:= if H[i,2] ≤ 0 then 1 else H[i,2] end
  end else begin BT1(H[i,1]); BT1(H[i,3]) end;
  procedure OT (i); integer i;
  begin procedure OT1(i); integer i;
    begin if H[i,2] = ST ∧ i ≠ one then
      begin if i = 0 then goto END; l1:= l1 + 1; if fi then a[l,l1]:= i end
      else if i ≠ one then begin OT1(H[i,1]); OT1(H[i,3]) end
    end; OT1(i); if l1 = 0 then begin l1:= 1; if fi then a[l,l1]:= one end;
    goto A; END: l:= l - 1; l1:= 0;
  A: end; l:= 0; BT1(i); L[0]:= l
end;
procedure SUBCALC(i,c); integer i; integer array c;
begin integer n,j,i1,i2; integer array L[0:50]; i1:= i; i2:= 0;
A: BT(i1,H,L, false); n:= if L[0] = 0 then 0 else L[1]; for j:= 2 step 1
  until L[0] do begin if n < L[j] then n:= L[j] end;
  begin integer array a[1:L[0],0:n]; BT(i1,a,L,true);
    k:= K; CAL(a,L); if i2 = 1 then goto B; i2:= 1;
    i1:= SUBSTITUTE(true,a,L); goto A; B: PUN(c,a,L);
  END: end
end;
procedure CAL (a, L); integer array a, L;
begin integer i, i1, j; procedure COM;
  begin integer k; if a[j,0] = 0 then goto END;
    if L[i] ≠ L[j] then goto END; for k:= 1 step 1 until L[i] do
    begin if a[i,k] ≠ a[j,k] then goto END end; if a[i,0] > KC then
    i1:= a[i,0] else i1:= kc:= kc + 1; HC[i1,1]:= HC[a[i,0],1] + HC[a[j,0],1];
    HC[i1,2]:= HC[a[i,0],2] + HC[a[j,0],2]; a[i,0]:= i1; a[j,0]:= 0;
    if abs(HC[i1,1])+abs(HC[i1,2]) < 10^-10 then a[i,0]:= 0;
  END: end;
  for i:= 1 step 1 until L[0] do
  begin A: for i1:= 1 step 1 until L[i] - 1 do
    begin if a[i,i1] > a[i,i1+1] then
      begin j:= a[i,i1]; a[i,i1]:= a[i,i1+1]; a[i,i1+1]:= j; goto A end
  end end; for i:= 1 step 1 until L[0] - 1 do for j:= i+1 step 1 until L[0]
  do COM; for i:= 1 step 1 until L[0] do
  begin if L[i] > 3 then
    begin if a[i,L[i]-2] = eiotplkx then a[i,0]:= 0 end
  end; i1:= 1; for i:= 1 step 1 until L[0] do
  begin if i1 ≠ i ∧ a[i,0] ≠ 0 then
    begin for j:= 0 step 1 until L[i] do a[i1,j]:= a[i,j]; L[i1]:= L[i];
      i1:= i1 + 1
    end else if a[i,0] ≠ 0 then i1:= i1 + 1
  end; L[0]:= i1 - 1; i1:= kc; j:= 0; for i:= 1 step 1 until L[0] do
  begin if a[i,0] > KC then
    begin kc:= kc + 1; HC[kc,1]:= HC[a[i,0],1]; HC[kc,2]:= HC[a[i,0],2] end
  end;
```

```
comment Continuation of Second order program;
  kc:= KC; for i:= 1 step 1 until L[0] do
  begin if a[i,0] > KC then
    begin j:= j+1; a[i,0]:= kc:= kc+1; HC[kc,1]:= HC[i1+j,1];
      HC[kc,2]:= HC[i1+j,2]
    end
  end; if XEEN(1024) = 1024 then
  begin RUNOUT; PS; comment; OUTPUT1(L,a); PUTEXT1(‹;›); RUNOUT
end end;
integer procedure SUBSTITUTE(fi,a,L); Boolean fi; integer array a, L;
begin integer i,j; integer procedure prod(i); integer i;
  begin integer j,p,e,b; p:= one; e:= -1; b:= if a[i,L[i]-1] = eiotplkx then
    1 else 0; for j:= 1 step 1 until L[i] do
    begin if fi then
      begin if a[i,j] ≥ ek1y ∧ a[i,j] ≤ emu1y ∧ b = 0 then
        e:= (ek1y - a[i,j] - k1) × e else if a[i,j] < ek1y ∨ a[i,j] > emu2y
        then p:= P(p,a[i,j])
      end else p:= P(p,a[i,j])
    end; if a[i,0] > 1 then p:= PC(HC[a[i,0],1],HC[a[i,0],2],p);
    if e > 0 then p:= P(p,S(one,P(e,z1))); prod:= p
  end; if L[0] = 0 then begin SUBSTITUTE:= 0; goto END end; j:= prod(1);
  for i:= 2 step 1 until L[0] do j:= S(j,prod(i)); SUBSTITUTE:= j;
END: end;
procedure PUN(c,a,L); integer array c,a,L;
begin integer i,j; procedure PU(n,st); integer n; string st;
  begin a[i,j]:= +1; c[n]:= c[n] + 1; PS; PUTEXT1(st); ABSFIXP(2,0,c[n]);
    PUTEXT1(‹],›); PROD(i,a,L); PUTEXT1(‹);›); j:= L[i]
  end; for i:= 1 step 1 until L[0] do
  begin a[i,L[i]]:= 1; if a[i,L[i]-1] ≠ eiotplkx then
    begin if XEEN(2) = 2 then
      begin for j:= 1 step 1 until L[i] - 1 do
        begin if a[i,j] = A1 then PU(1,‹U1(CA1[›) else
              if a[i,j] = B1 then PU(2,‹U1(CB1[›) else
              if a[i,j] = E1 then PU(3,‹U1(CE1[›)
    end end end else
    begin a[i,L[i]-1]:= 1; for j:= 1 step 1 until L[i] - 2 do
      begin if a[i,j] = A2 then PU(4,‹U1(CA2[›) else
            if a[i,j] = B2 then PU(5,‹U1(CB2[›) else
            if a[i,j] = Z2 then PU(6,‹U1(CZ2[›) else
            if a[i,j] = E2 then PU(7,‹U1(CE2[›) else
        if j = L[i] - 2 then begin j:= j + 1; PU(8,‹U1(C2[›) end
end end end end;
procedure PROD(i,a,L); value i; integer i; integer array a, L;
begin integer j, r0, r1, p, p0, q; integer array R0, R1[1:L[i]];
  r0:= r1:= p:= p0:= q:= 0; for j:= 1 step 1 until L[i] do
  begin if a[i,j] ≠ 1 then
    begin if H[a[i,j],1] = -2 then begin r0:= r0 + 1; R0[r0]:= a[i,j] end
      else begin r1:= r1 + 1; R1[r1]:= a[i,j] end
  end end; if r0 > 0 then
  begin PUTEXT1(‹PRC(›); AP(R0[1]);
```

```
comment Continuation of Second order program;
    for j:= 2 step 1 until r0 do
    begin PUTEXT1(‹ × ›); AP(R0[j]) end; PUTEXT1(‹,›)
  end; for j:= 1 step 1 until r1 do
  begin if R1[j] = p0 then p:= p + 1 else
    begin if p > 1 then ABSFIXP(1,0,p); if j ≠ 1 then PUTEXT1(‹),›);
      PUTEXT1(‹P(T(›); AP(R1[j]); q:= q + 1; p0:= R1[j]; p:= 1
  end end; if p > 1 then ABSFIXP(1,0,p); if r1 > 0 then PUTEXT1(‹),›);
  PUTEXT1(‹J(›); if a[i,0] = 1 then PUTEXT1(‹1,0›) else
  begin FLOP(3,1,HC[a[i,0],1]); PUTEXT1(‹,›); FLOP(3,1,HC[a[i,0],2]) end;
  for j:= (if r0 > 0 then -1 else 0) step 1 until q do PUTEXT1(‹)›)
end;

BEGIN of PROGRAM: RUNOUT; AP(-1); H[OMEGA,1]:= H[EGO,1]:=
  H[RHO,1]:= H[RTGGAPC,1]:= H[EGAPC,1]:= H[sigma0,1]:=H[ETA,1]:=
  H[g,1]:= H[CNPA,1]:= H[GCCPA,1]:= H[GTCPA,1]:= H[D,1]:= H[AG,1]:= -2;
  u1:= P(S(PC(0,-1,P(k1,P(A1,ek1y))),PC(-1,0,P(m1,P(B1,em1y)))),eiotplkx);
  v1:= P(S(PC(-1,0,P(k1,P(A1,ek1y))),PC(0,1,P(k1,P(B1,em1y)))),eiotplkx);
  ekm2iok:= P(ek1y,P(em1y,P(eiotplkx,eiotplkx)));
  u2:= S(P(S(PC(0,-2,P(k1,P(A2,ek2y))),PC(-1,0,P(m2,P(B2,em2y)))),
   P(eiotplkx,eiotplkx)),PC(-1,0,P(S(m1,k1),P(PSI,ekm2iok))));
  v2:= S(P(S(PC(-2,0,P(k1,P(A2,ek2y))),PC(0,2,P(k1,P(B2,em2y)))),
   P(eiotplkx,eiotplkx)),PC(0,2,P(k1,P(PSI,ekm2iok))));
  z1:= P(PC(0,-1,P(EGO,S(PC(-1,0,P(k1,A1)),PC(0,1,P(k1,B1))))),eiotplkx);
  p1:= PC(0,1,P(RHO,P(OMEGA,P(A1,P(ek1y,eiotplkx)))));
  p2:= S(PC(0,2,P(RHO,P(OMEGA,P(A2,P(ek2y,P(eiotplkx,eiotplkx)))))),
   P(PST,ekm2iok)); v:= S(v1,v2); u:= S(u1,u2);
  cst:= P(S(P(CST1,P(ek1y,emu1y)),P(CST2,P(em1y,emu1y))),
   P(eiotplkx,eiotplkx)); c1:= P(E1,P(emu1y,eiotplkx));
  sigma1:= PC(-1,0,P(RTGGAPC,c1)); K:= k; KC:= kc;
  begin integer i, K1, KC1; integer array cc[1:8];
    procedure SI (i); integer i; begin SUBCALC(i,cc); k:= K; kc:= KC end;
    procedure PU (st1,n,st2,st3); integer n; string st1,st2,st3;
    begin PS; PUTEXT1(st1); if cc[n] = 0 then
      begin PUTEXT1(‹J(0,0)›); goto END end; PUTEXT1(‹Sum(›);
      ABSFIXP(2,0,cc[n]); PUTEXT1(‹,›); PUTEXT1(st2); PUTEXT1(‹)›);
      END: PUTEXT1(st3)
    end;
    for i:= 1,2,3,4,5,6,7,8 do cc[i]:= 0;
  BC1: SI(DIFF(3,S(z1,P(Z2,P(eiotplkx,eiotplkx))))); SI(PC(-1,0,v));
    SI(P(u1,DIFF(1,z1))); PU(‹U(COZ2,›,6,‹CZ2›,‹)›);
  ASSEMBLAGE:
    PU(‹U(CZ2A2,Q(PRC(-1,›,4,‹CA2›,‹),T(COZ2)))›);
    PU(‹U(CZ2B2,Q(PRC(-1,›,5,‹CB2›,‹),T(COZ2)))›);
    PU(‹U(COZ2,Q(PRC(-1,›,8,‹C2›,‹),T(COZ2)))›);
    z2:= P(S(P(CZ2A2,A2),S(P(CZ2B2,B2),COZ2)),P(eiotplkx,eiotplkx));
    K1:= k; KC1:= kc; for i:= 1,2,3,4,5,6,7,8 do cc[i]:= 0; i:= 1;
  AA: c2:= S(P(E2,P(emu2y,P(eiotplkx,eiotplkx))),cst);
    sigma2:= PC(-1,0,P(RTGGAPC,S(c2,PC(-.5,0,P(EGAPC,P(c1,c1))))));
```

```
comment Continuation of Second order program;
  K:= k; KC:= kc;
BC2:
  SI(P(ETA,PC(2,0,P(DIFF(1,z1),S(DIFF(2,v1),PC(-1,0,DIFF(1,u1)))))));
  SI(P(ETA,S(DIFF(2,u),DIFF(1,v)))); SI(PC(-1,0,DIFF(1,S(sigma1,sigma2))));
  SI(PC(-1,0,P(DIFF(1,z1),DIFF(2,sigma1))));
  PU(‹U(COE2,›,7,‹CE2›,‹));
  PU(‹U(CE2A2,Q(PRC(-1,›,4,‹CA2›,‹),T(COE2)));
  PU(‹U(CE2B2,Q(PRC(-1,›,5,‹CB2›,‹),T(COE2)));
  PU(‹U(COE2,Q(PRC(-1,›,8,‹C2›,‹),T(COE2)));
  for i:= 1,2,3,4,5,6,7,8 do cc[i]:= 0; k:= K1; kc:= KC1; i:= E2;
  E2:= S(P(CE2A2,A2),S(P(CE2B2,B2),COE2)); goto AA; E2:= i;
BC3: SI(S(P(RHO,P(g,S(z1,z2))),PC(-1,0,S(p1,p2))));
  SI(PC(2,0,P(ETA,DIFF(2,v))));
  SI(PC(-2,0,P(ETA,P(DIFF(1,z1),S(DIFF(2,u1),DIFF(1,v1))))));
  SI(PC(-1,0,P(sigma1,DIFF(1,DIFF(1,z1)))));
  SI(PC(-1,0,P(sigma0,DIFF(1,DIFF(1,S(z1,z2))))));
  PU(‹U(coeff11,›,4,‹CA2›,‹));
  PU(‹U(coeff12,›,5,‹CB2›,‹));
  PU(‹U(coeff13,PRC(-1,›,8,‹C2›,‹));
  for i:= 1,2,3,4,5,6,7,8 do cc[i]:= 0;
BC4:
  SI(P(D,P(CNPA,P(CNPA,S(DIFF(2,S(c1,c2)),PC(-1,0,P(DIFF(1,z1),
  DIFF(1,c1)))))));  SI(P(D,PC(2,0,P(CNPA,P(c1,DIFF(2,c1))))));
  SI(P(GCCPA,S(DIFF(1,u),P(DIFF(1,z1),S(DIFF(2,u1),DIFF(1,v1))))));
  SI(P(GTCPA,P(c1,DIFF(1,u1))));
  SI(P(AG,S(DIFF(3,S(c1,c2)),S(P(u1,DIFF(1,c1)),P(v1,DIFF(2,c1))))));
  PU(‹U(coeff21,›,4,‹CA2›,‹));
  PU(‹U(coeff22,›,5,‹CB2›,‹));
  PU(‹U(coeff23,PRC(-1,›,8,‹C2›,‹)); RUNOUT
end end end
```

```
begin comment Calculation program;
  real NU, c0, a, D, GAMMA, SIGMA, SIGMAO, RT, ETA, RHO, g, OMEGA;
  integer k, ka; array k1, k12, k13, k14, m1, m12, m13, mu1, mu12, mu13,
  m2, m22, mu2, mu22, A1, A12, B1, B12, E1, E12, A2, B2, CZ2A2,
  CZ2B2, COZ2, CE2A2, CE2B2, COE2, PSI, PST, CST1, CST2, coeff11,
  coeff12, coeff13, coeff21, coeff22, coeff23[1:2], H[1:100,1:2];
  integer array CZ2,CE2[1:1],CA2,CB2[1:4],C2[1:28];
  integer procedure T(a); array a;
  begin T:= k:= k + 1; H[k,1]:= a[1]; H[k,2]:= a[2] end;
  integer procedure P(i,j); value i,j; integer i,j;
  begin real a; P:= k:= k - 1; a:= H[i,1] × H[j,1] - H[i,2] × H[j,2];
    H[k,2]:= H[i,1] × H[j,2] + H[i,2] × H[j,1]; H[k,1]:= a
  end;
  integer procedure Q(i,j); value i,j; integer i,j;
  begin real a,b; Q:= k:= k - 1; b:= H[j,1]↑2 + H[j,2]↑2; a:= (H[i,1] × H[j,1] +
    H[i,2] × H[j,2])/b; H[k,2]:= (H[i,2] × H[j,1] - H[i,1] × H[j,2])/b; H[k,1]:= a
  end;
  integer procedure S(i,j); value i,j; integer i,j;
  begin S:= k:= k - 1; H[k,1]:= H[i,1] + H[j,1]; H[k,2]:= H[i,2] + H[j,2] end;
  procedure U(R, i); value i; integer i; array R;
  begin R[1]:= H[i,1]; R[2]:= H[i,2]; k:= k - 1 end;
  integer procedure PRC(a,i); value a,i; real a; integer i;
  begin PRC:= k; H[k,1]:= a × H[i,1]; H[k,2]:= a × H[i,2] end;
  integer procedure J(a1,a2); real a1, a2;
  begin J:= k:= k + 1; H[k,1]:= a1; H[k,2]:= a2 end;
  procedure FLOPC (STRING, a); string STRING; array a;
  begin ka:= 50; PUNLCR; PUTEXT(STRING); PUTEXT1(‹  ›);
    FLOP(10,3,a[1]); PUTEXT(‹+ i.(›); FLOP(10,3,a[2]); PUTEXT(‹)›)
  end;
  procedure U1 (i,j); value j; integer i,j;
  begin i:= ka:= ka + 1; H[ka,1]:= H[j,1]; H[ka,2]:= H[j,2]; k:= k - 1 end;
  integer procedure Sum(i,a); value i; integer i; integer array a;
  begin integer j; Sum:= k:= k + 1; H[k,1]:= H[k,2]:= 0; for j:= 1 step 1
    until i do begin H[k,1]:= H[k,1] + H[a[j],1]; H[k,2]:= H[k,2] + H[a[j],2] end
  end;
  procedure RE(a); array a; begin a[1]:= read; a[2]:= read end;

BEGIN of CALCULATION:
  RUNOUT; PUNLCR; SIGMAO:= read; RT:= read; ETA:= read; RHO:= read;
  g:= read; NU:= read; D:= read; GAMMA:= read; c0:= read; a:= read;
  OMEGA:= NU × 6.2831853071794; SIGMA:= SIGMAO - RT × GAMMA ×
  ln (1 + c0/a); k:= 0; ka:= 50; RE(k1); U(k12,P(T(k1),T(k1)));
  U(k13,P(T(k1),T(k12))); U(k14,P(T(k1),T(k13))); RE(mu1); RE(m1); RE(mu2);
  RE(m2); U(m12,P(T(m1),T(m1))); U(m13,P(T(m1),T(m12))); U(mu12,
  P(T(mu1),T(mu1))); U(mu13,P(T(mu1),T(mu12))); U(m22,P(T(m2),T(m2)));
  U(mu22,P(T(mu2),T(mu2))); RE(A1); U(A12,P(T(A1),T(A1))); RE(B1);
  U(B12,P(T(B1),T(B1))); RE(E1); U(E12,P(T(E1),T(E1))); RE(PSI); RE(PST);
  RE(CST1); RE(CST2);
```

```
  comment Continuation of Calculation program;
CALCULATION: k:= k;
  comment
  (-.100[10]+1  + ix+   0     )OMEGA OMEGA/\(-1) k1 A1 e/\i(OMEGA t + k1 x) +
  (+   0      + ix+.100[10]+1 )OMEGA OMEGA/\(-1) k1 B1 e/\i(OMEGA t + k1 x) +
  (+   0      + ix+.200[10]+1 )OMEGA Z2 e/\i(OMEGA t + k1 x)
   e/\i(OMEGA t + k1 x) ;
  comment
  (-.100[10]+1  + ix+   0     )OMEGA OMEGA/\(-1) k1 A1 e/\i(OMEGA t + k1 x) +
  (+   0      + ix+.100[10]+1 )OMEGA OMEGA/\(-1) k1 B1 e/\i(OMEGA t + k1 x) +
  (+   0      + ix+.200[10]+1 )OMEGA Z2 e/\i(OMEGA t + k1 x)
   e/\i(OMEGA t + k1 x) ;
  U1(CZ2[  1 ],PRC(OMEGA,J(+   0     ,+.200[10]+1 )));
  comment
  (+.100[10]+1  + ix-   0     )k1 A1 e/\k1y e/\i(OMEGA t + k1 x) +
  (-   0      + ix-.100[10]+1 )k1 B1 e/\m1y e/\i(OMEGA t + k1 x) +
  (+.200[10]+1  + ix-   0     )k1 A2 e/\k2y e/\i(OMEGA t + k1 x)
   e/\i(OMEGA t + k1 x) +
  (-   0      + ix-.200[10]+1 )k1 B2 e/\m2y e/\i(OMEGA t + k1 x)
   e/\i(OMEGA t + k1 x) +
  (-   0      + ix-.200[10]+1 )k1 PSI e/\k1y e/\m1y e/\i(OMEGA t + k1 x)
   e/\i(OMEGA t + k1 x) ;
  comment
  (+.100[10]+1  + ix-   0     )k1 A1 e/\i(OMEGA t + k1 x) +
  (+   0      + ix+.100[10]+1 )OMEGA/\(-1) k1 k1 k1 A1 A1
   e/\i(OMEGA t + k1 x) e/\i(OMEGA t + k1 x) +
  (+.100[10]+1  + ix-   0     )OMEGA/\(-1) k1 k1 k1 A1 B1
   e/\i(OMEGA t + k1 x) e/\i(OMEGA t + k1 x) +
  (-   0      + ix-.100[10]+1 )k1 B1 e/\i(OMEGA t + k1 x) +
  (+.100[10]+1  + ix-   0     )OMEGA/\(-1) k1 k1 m1 A1 B1
   e/\i(OMEGA t + k1 x) e/\i(OMEGA t + k1 x) +
  (-   0      + ix-.100[10]+1 )OMEGA/\(-1) k1 k1 m1 B1 B1
   e/\i(OMEGA t + k1 x) e/\i(OMEGA t + k1 x) +
  (+.200[10]+1  + ix-   0     )k1 A2 e/\i(OMEGA t + k1 x)
   e/\i(OMEGA t + k1 x) +
  (-   0      + ix-.200[10]+1 )k1 B2 e/\i(OMEGA t + k1 x)
   e/\i(OMEGA t + k1 x) +
  (-   0      + ix-.200[10]+1 )k1 PSI e/\i(OMEGA t + k1 x)
   e/\i(OMEGA t + k1 x) ;
  U1(C2[  1 ],PRC(OMEGA/\(-1),P(T(k1 3 ),P(T(A1 2 ),
   J(+   0     ,+.100[10]+1 ))));
  U1(C2[  2 ],PRC(OMEGA/\(-1),P(T(k1 3 ),P(T(A1),P(T(B1),
   J(+.100[10]+1 ,-   0     ))));
  U1(C2[  3 ],PRC(OMEGA/\(-1),P(T(k1 2 ),P(T(m1),P(T(A1),P(T(B1),
   J(+.100[10]+1 ,-   0     )));
  U1(C2[  4 ],PRC(OMEGA/\(-1),P(T(k1 2 ),P(T(m1),P(T(B1 2 ),
   J(-   0     ,-.100[10]+1 ))));
  U1(CA2[  1 ],P(T(k1),J(+.200[10]+1 ,-   0     )));
  U1(CB2[  1 ],P(T(k1),J(-   0     ,-.200[10]+1 )));
  U1(C2[  5 ],P(T(k1),P(T(PSI),J(-   0     ,-.200[10]+1 ))));
```

```
comment Continuation of Calculation program
(+   0     + ix+.100₁₀+1 )OMEGA↑(-1) k1 k1 k1 A1 A1 e↑k1y
 e↑i(OMEGA t + k1 x) e↑i(OMEGA t + k1 x) +
(+.100₁₀+1  + ix+   0     )OMEGA↑(-1) k1 k1 k1 A1 B1 e↑k1y
 e↑i(OMEGA t + k1 x) e↑i(OMEGA t + k1 x) +
(+.100₁₀+1  + ix-   0     )OMEGA↑(-1) k1 k1 m1 A1 B1 e↑m1y
e↑i(OMEGA t + k1 x) e↑i(OMEGA t + k1 x) +
(-   0     + ix-.100₁₀+1 )OMEGA↑(-1) k1 k1 m1 B1 B1 e↑m1y
 e↑i(OMEGA t + k1 x) e↑i(OMEGA t + k1 x) ;
comment
(+   0     + ix+.100₁₀+1 )OMEGA↑(-1) k1 k1 k1 A1 A1
 e↑i(OMEGA t + k1 x) e↑i(OMEGA t + k1 x) +
(+.100₁₀+1  + ix+   0     )OMEGA↑(-1) k1 k1 k1 A1 B1
 e↑i(OMEGA t + k1 x) e↑i(OMEGA t + k1 x) +
(+.100₁₀+1  + ix-   0     )OMEGA↑(-1) k1 k1 m1 A1 B1
 e↑i(OMEGA t + k1 x) e↑i(OMEGA t + k1 x) +
(-   0     + ix-.100₁₀+1 )OMEGA↑(-1) k1 k1 m1 B1 B1
 e↑i(OMEGA t + k1 x) e↑i(OMEGA t + k1 x) ;
U1(C2[  6 ],PRC(OMEGA↑(-1),P(T(k1 3 ),P(T(A1 2 ),
 J(+   0     ,+.100₁₀+1 )));
U1(C2[  7 ],PRC(OMEGA↑(-1),P(T(k1 3 ),P(T(A1),P(T(B1),
 J(+.100₁₀+1 ,+   0     )));
U1(C2[  8 ],PRC(OMEGA↑(-1),P(T(k1 2 ),P(T(m1),P(T(A1),P(T(B1),
 J(+.100₁₀+1 ,-   0     )));
U1(C2[  9 ],PRC(OMEGA↑(-1),P(T(k1 2 ),P(T(m1),P(T(B1 2 ),
 J(-   0     ,-.100₁₀+1 )));
ASSEMBLAGE:
 U(COZ2,Sum(  1 ,CZ2));
 U(CZ2A2,Q(PRC(-1,Sum(  1 ,CA2)),T(COZ2)));
 U(CZ2B2,Q(PRC(-1,Sum(  1 ,CB2)),T(COZ2)));
 U(COZ2,Q(PRC(-1,Sum(  9 ,C2)),T(COZ2))); FLOPC(↓CZ2A2 =↓,CZ2A2);
 FLOPC(↓CZ2B2 =↓,CZ2B2); FLOPC(↓COZ2 =↓,COZ2);
 comment We removed from the following the comment parts;
 U1(C2[  1 ],PRC(OMEGA↑(-1) × ETA,P(T(k1 4 ),P(T(A1 2 ),
  J(+.400₁₀+1 ,-   0     )));
 U1(C2[  2 ],PRC(OMEGA↑(-1) × ETA,P(T(k1 3 ),P(T(m1),P(T(A1),P(T(B1),
  J(-   0     ,-.400₁₀+1 )));
 U1(C2[  3 ],PRC(OMEGA↑(-1) × ETA,P(T(k1 4 ),P(T(A1),P(T(B1),
  J(-   0     ,-.400₁₀+1 )));
 U1(C2[  4 ],PRC(OMEGA↑(-1) × ETA,P(T(k1 3 ),P(T(m1),P(T(B1 2 ),
  J(-.400₁₀+1 ,+   0     )));
 U1(C2[  5 ],PRC(OMEGA↑(-1) × ETA,P(T(k1 4 ),P(T(A1 2 ),
  J(+.200₁₀+1 ,-   0     )));
 U1(C2[  6 ],PRC(OMEGA↑(-1) × ETA,P(T(k1 4 ),P(T(A1),P(T(B1),
  J(+   0     ,-.200₁₀+1 )));
 U1(C2[  7 ],PRC(OMEGA↑(-1) × ETA,P(T(k1),P(T(m1 3 ),P(T(A1),P(T(B1),
  J(-   0     ,-.100₁₀+1 )));
 U1(C2[  8 ],PRC(OMEGA↑(-1) × ETA,P(T(k1),P(T(m1 3 ),P(T(B1 2 ),
  J(-.100₁₀+1 ,+   0     )));
```

```
comment Continuation of Calculation program;
U1(CA2[  1 ],PRC(ETA,P(T(k1 2 ),J(+    0     ,-.800₁₀+1 ))));
U1(CB2[  1 ],PRC(ETA,P(T(m2 2 ),J(-.100₁₀+1 ,+    0     ))));
U1(C2[  9 ],PRC(ETA,P(T(k1),P(T(m1),P(T(PSI),J(-.200₁₀+1 ,+    0     ))))));
U1(C2[ 10 ],PRC(ETA,P(T(m1 2 ),P(T(PSI),J(-.100₁₀+1 ,+    0     )))));
U1(C2[ 11 ],PRC(ETA,P(T(k1 2 ),P(T(PSI),J(-.500₁₀+1 ,+    0     )))));
U1(C2[ 12 ],PRC(OMEGA↑(-1) × ETA,P(T(k1 3 ),P(T(m1),P(T(A1),
 P(T(B1),J(-    0     ,-.100₁₀+1 )))))));
U1(C2[ 13 ],PRC(OMEGA↑(-1) × ETA,P(T(k1 3 ),P(T(m1),P(T(B1 2 ),
 J(-.100₁₀+1 ,+    0     ))))));
U1(CB2[  2 ],PRC(ETA,P(T(k1 2 ),J(-.400₁₀+1 ,+    0     ))));
U1(C2[ 14 ],PRC(OMEGA↑(-1) × RT×GAMMA/(a+c0),P(T(k1 2 ),P(T(mu1),
 P(T(A1),P(T(E1),J(-.100₁₀+1 ,+    0     )))))));
U1(C2[ 15 ],PRC(OMEGA↑(-1) × RT×GAMMA/(a+c0),P(T(k1 2 ),P(T(mu1),
 P(T(B1),P(T(E1),J(+    0     ,+.100₁₀+1 )))))));
U1(CE2[  1 ],PRC(RT×GAMMA/(a+c0),P(T(k1),J(+    0     ,+.200₁₀+1 ))));
U1(C2[ 16 ],PRC(RT×GAMMA/(a+c0),P(T(k1),P(T(CST1),
 J(    0     ,+.200₁₀+1 )))));
U1(C2[ 17 ],PRC(RT×GAMMA/(a+c0),P(T(k1),P(T(CST2),
 J(+    0     ,+.200₁₀+1 )))));
U1(C2[ 18 ],PRC(RT×GAMMA/(a+c0) × 1/(a+c0),P(T(k1),P(T(E1 2 ),
 J(-    0     ,-.100₁₀+1 )))));
U1(C2[ 19 ],PRC(OMEGA↑(-1) × RT×GAMMA/(a+c0),P(T(k1 2 ),P(T(mu1),
 P(T(A1),P(T(E1),J(-.100₁₀+1 ,+    0     )))))));
U1(C2[ 20 ],PRC(OMEGA↑(-1) × RT×GAMMA/(a+c0),P(T(k1 2 ),P(T(mu1),
 P(T(B1),P(T(E1),J(+    0     ,+.100₁₀+1 )))))));
U(COE2,Sum(  1 ,CE2));
U(CE2A2,Q(PRC(-1,Sum(  1 ,CA2)),T(COE2)));
U(CE2B2,Q(PRC(-1,Sum(  2 ,CB2)),T(COE2)));
U(COE2,Q(PRC(-1,Sum( 20 ,C2)),T(COE2)));
FLOPC(⊰CE2A2 =⊱,CE2A2); FLOPC(⊰CE2B2 =⊱,CE2B2);
FLOPC(⊰COE2 =⊱,COE2);
U1(CA2[  1 ],PRC(RHO × g,P(T(CZ2A2),J(1,0))));
U1(CB2[  1 ],PRC(RHO × g,P(T(CZ2B2),J(1,0))));
U1(C2[  1 ],PRC(RHO × g,P(T(COZ2),J(1,0))));
U1(C2[  2 ],PRC(OMEGA × OMEGA↑(-1) × RHO,P(T(k1 2 ),
 P(T(A1 2 ),J(+.100₁₀+1 ,-    0     )))));
U1(C2[  3 ],PRC(OMEGA × OMEGA↑(-1) × RHO,P(T(k1 2 ),
 P(T(A1),P(T(B1),J(-    0     ,-.100₁₀+1 ))))));
U1(CA2[  2 ],PRC(OMEGA × RHO,J(-    0     ,-.200₁₀+1 )));
U1(C2[  4 ],P(T(PST),J(-.100₁₀+1 ,+    0     )));
U1(C2[  5 ],PRC(OMEGA↑(-1) × ETA,P(T(k1 4 ),P(T(A1 2 ),
 J(-    0     ,-.200₁₀+1 )))));
U1(C2[  6 ],PRC(OMEGA↑(-1) × ETA,P(T(k1 4 ),P(T(A1),
 P(T(B1),J(-.200₁₀+1 ,-    0     ))))));
U1(C2[  7 ],PRC(OMEGA↑(-1) × ETA,P(T(k1 2 ),P(T(m1 2 ),
 P(T(A1),P(T(B1),J(-.200₁₀+1 ,+    0     )))))));
U1(C2[  8 ],PRC(OMEGA↑(-1) × ETA,P(T(k1 2 ),P(T(m1 2 ),
 P(T(B1 2 ),J(-    0     ,+.200₁₀+1 ))))));
U1(CA2[  3 ],PRC(ETA,P(T(k1 2 ),J(-.800₁₀+1 ,-    0     ))));
U1(CB2[  2 ],PRC(ETA,P(T(k1),P(T(m2),J(+    0     ,+.400₁₀+1 )))));
```

```
comment Continuation of Calculation program;
U1(C2[  9 ],PRC(ETA,P(T(k1 2 ),P(T(PSI),J(+    0     ,+.400[10]+1 )))));
U1(C2[ 10 ],PRC(ETA,P(T(k1),P(T(m1),P(T(PSI),J(+    0     ,+.400[10]+1 ))))));
U1(C2[ 11 ],PRC(OMEGA↑(-1) × ETA,P(T(k1 4 ),P(T(A1 2 ),
 J(-    0     ,-.400[10]+1 )))));
U1(C2[ 12 ],PRC(OMEGA↑(-1) × ETA,P(T(k1 2 ),P(T(m1 2 ),P(T(A1),
 P(T(B1),J(-.200[10]+1 ,+    0     )))))));
U1(C2[ 13 ],PRC(OMEGA↑(-1) × ETA,P(T(k1 4 ),P(T(A1),P(T(B1),
 J(-.600[10]+1 ,+    0     ))))));
U1(C2[ 14 ],PRC(OMEGA↑(-1) × ETA,P(T(k1 2 ),P(T(m1 2 ),P(T(B1 2 ),
 J(+    0     ,+.200[10]+1 ))))));
U1(C2[ 15 ],PRC(OMEGA↑(-1) × ETA,P(T(k1 4 ),P(T(B1 2 ),
 J(+    0     ,+.200[10]+1 )))));
U1(C2[ 16 ],PRC(OMEGA↑(-1) × RT×GAMMA/(a+c0),P(T(k1 3 ),P(T(A1),
 P(T(E1),J(-    0     ,-.100[10]+1 ))))));
U1(C2[ 17 ],PRC(OMEGA↑(-1) × RT×GAMMA/(a+c0),P(T(k1 3 ),P(T(B1),
 P(T(E1),J(-.100[10]+1 ,+    0     ))))));
U1(CA2[  4 ],PRC(SIGMA,P(T(k1 2 ),P(T(CZ2A2),J(+.400[10]+1 ,-    0     )))));
U1(CB2[  3 ],PRC(SIGMA,P(T(k1 2 ),P(T(CZ2B2),J(+.400[10]+1 ,-    0     )))));
U1(C2[ 18 ],PRC(SIGMA,P(T(k1 2 ),P(T(COZ2),J(+.400[10]+1 ,-    0     )))));
U(coeff11,Sum(  4 ,CA2));
U(coeff12,Sum(  3 ,CB2));
U(coeff13,PRC(-1,Sum( 18 ,C2)));
FLOPC(↓coeff11 =↓,coeff11); FLOPC(↓coeff12 =↓,coeff12);
FLOPC(↓coeff13 =↓,coeff13);
U1(C2[  1 ],PRC(OMEGA↑(-1) × (a+c0) × (a+c0) × D,P(T(k1),P(T(mu1 2 ),
 P(T(A1),P(T(E1),J(+    0     ,+.100[10]+1 )))))));
U1(C2[  2 ],PRC(OMEGA↑(-1) × (a+c0) × (a+c0) × D,P(T(k1),P(T(mu1 2 ),
 P(T(B1),P(T(E1),J(+.100[10]+1 ,+    0     )))))));
U1(CA2[  1 ],PRC((a+c0) × (a+c0) × D,P(T(mu2),P(T(CE2A2),J(1,0)))));
U1(CB2[  1 ],PRC((a+c0) × (a+c0) × D,P(T(mu2),P(T(CE2B2),J(1,0)))));
U1(C2[  3 ],PRC((a+c0) × (a+c0) × D,P(T(mu2),P(T(COE2),J(1,0)))));
U1(C2[  4 ],PRC((a+c0) × (a+c0) × D,P(T(k1),P(T(CST1),J(1,0)))));
U1(C2[  5 ],PRC((a+c0) × (a+c0) × D,P(T(mu1),P(T(CST1),J(1,0)))));
U1(C2[  6 ],PRC((a+c0) × (a+c0) × D,P(T(m1),P(T(CST2),J(1,0)))));
U1(C2[  7 ],PRC((a+c0) × (a+c0) × D,P(T(mu1),P(T(CST2),J(1,0)))));
U1(C2[  8 ],PRC(OMEGA↑(-1) × (a+c0) × (a+c0) × D,P(T(k1 3 ),P(T(A1),
 P(T(E1),J(+    0     ,+.100[10]+1 ))))));
U1(C2[  9 ],PRC(OMEGA↑(-1) × (a+c0) × (a+c0) × D,P(T(k1 3 ),P(T(B1),
 P(T(E1),J(+.100[10]+1 ,-    0     ))))));
U1(C2[ 10 ],PRC((a+c0) × D,P(T(mu1),P(T(E1 2 ),J(+.200[10]+1 ,+    0     )))));
U1(C2[ 11 ],PRC(OMEGA↑(-1) × GAMMA×c0×(a+c0),P(T(k1 4 ),P(T(A1 2 ),
 J(+    0     ,+.300[10]+1 )))));
U1(C2[ 12 ],PRC(OMEGA↑(-1) × GAMMA×c0×(a+c0),P(T(k1 4 ),P(T(A1),
 P(T(B1),J(+.400[10]+1 ,-    0     ))))));
U1(C2[ 13 ],PRC(OMEGA↑(-1) × GAMMA×c0×(a+c0),P(T(k1 2 ),P(T(m1 2 ),
 P(T(A1),P(T(B1),J(+.200[10]+1 ,-    0     )))))));
U1(C2[ 14 ],PRC(OMEGA↑(-1) × GAMMA×c0×(a+c0),P(T(k1 2 ),P(T(m1 2 ),
 P(T(B1 2 ),J(-    0     ,-.200[10]+1 ))))));
U1(CA2[  2 ],PRC(GAMMA×c0×(a+c0),P(T(k1 2 ),J(+.400[10]+1 ,+    0     ))));
```

```
comment Continuation of Calculation program;
U1(CB2[  2 ],PRC(GAMMA×c0×(a+c0),P(T(k1),P(T(m2),
 J(-   0     ,-.200₁₀+1 )));
U1(C2[ 15 ],PRC(GAMMA×c0×(a+c0),P(T(k1),P(T(m1),P(T(PSI),
 J(-   0     ,-.200₁₀+1 )));
U1(C2[ 16 ],PRC(GAMMA×c0×(a+c0),P(T(k1 2 ),P(T(PSI),
 J(-   0     ,-.200₁₀+1 )));
U1(C2[ 17 ],PRC(OMEGA↑(-1) × GAMMA×c0×(a+c0),P(T(k1 4 ),
 P(T(B1 2 ),J(-   0     ,-.100₁₀+1 )));
U1(C2[ 18 ],PRC(GAMMA×(2×c0+a),P(T(k1 2 ),P(T(A1),P(T(E1),
 J(+.100₁₀+1 ,+   0     )));
U1(C2[ 19 ],PRC(GAMMA×(2×c0+a),P(T(k1),P(T(m1),P(T(B1),P(T(E1),
 J(-   0     ,-.100₁₀+1 )));
U1(C2[ 20 ],PRC(OMEGA × OMEGA↑(-1) × GAMMA×a,P(T(k1),P(T(mu1),
 P(T(A1),P(T(E1),J(-.100₁₀+1 ,+   0     )));
U1(C2[ 21 ],PRC(OMEGA × OMEGA↑(-1) × GAMMA×a,P(T(k1),P(T(mu1),
 P(T(B1),P(T(E1),J(+   0     ,+.100₁₀+1 )));
U1(CA2[  3 ],PRC(OMEGA × GAMMA×a,P(T(CE2A2),
 J(+   0     ,+.200₁₀+1 ));
U1(CB2[  3 ],PRC(OMEGA × GAMMA×a,P(T(CE2B2),
 J(+   0     ,+.200₁₀+1 ));
U1(C2[ 22 ],PRC(OMEGA × GAMMA×a,P(T(COE2),J(+   0     ,+.200₁₀+1 ));
U1(C2[ 23 ],PRC(OMEGA × GAMMA×a,P(T(CST1),J(+   0     ,+.200₁₀+1 ));
U1(C2[ 24 ],PRC(OMEGA × GAMMA×a,P(T(CST2),J(+   0     ,+.200₁₀+1 ));
U1(C2[ 25 ],PRC(GAMMA×a,P(T(k1 2 ),P(T(A1),P(T(E1),
 J(+.100₁₀+1 ,+   0     )));
U1(C2[ 26 ],PRC(GAMMA×a,P(T(k1),P(T(m1),P(T(B1),P(T(E1),
 J(-   0     ,-.100₁₀+1 )));
U1(C2[ 27 ],PRC(GAMMA×a,P(T(k1),P(T(mu1),P(T(A1),P(T(E1),
 J(-.100₁₀+1 ,+   0     )));
U1(C2[ 28 ],PRC(GAMMA×a,P(T(k1),P(T(mu1),P(T(B1),P(T(E1),
 J(+   0     ,+.100₁₀+1 )));
U(coeff21,Sum(  3 ,CA2));
U(coeff22,Sum(  3 ,CB2));
U(coeff23,PRC(-1,Sum( 28 ,C2)));
FLOPC(⊰coeff21 =⊱,coeff21); FLOPC(⊰coeff22 =⊱,coeff22);
FLOPC(⊰coeff23 =⊱,coeff23);
comment From the label CALCULATION until so far this program
is, besides slight differences concerning the lay-out, constructed
by the Second order program. Finally A2 and B2 are calculated;
U(PSI,S(P(T(coeff11),T(coeff22)),PRC(-1,P(T(coeff12),T(coeff21)))));
U(A2,Q(S(P(T(coeff13),T(coeff22)),PRC(-1,P(T(coeff12),T(coeff23)))),
 T(PSI))); FLOPC(⊰A2 =⊱,A2);
U(B2,Q(S(P(T(coeff11),T(coeff23)),PRC(-1,P(T(coeff13),T(coeff21)))),
 T(PSI))); FLOPC(⊰B2 =⊱,B2); RUNOUT
end
```

References

1 R.P. van de Riet — Algebraic operations in ALGOL 60 (series expansions)
Report T.W. 97, Mathematical Centre, Amsterdam.

2 M. van den Tempel, R.P. van de Riet: Damping of waves by surface-active materials
Journal of Chemical Physics, april 1965.