INVERSE LAPLACE TRANSFORMATION FOR ARBITRARY POLE-ZERO CONFIGURATION USING DIGITAL COMPUTER

By

M. VAJTA, Jr. and T. KOVÁCS

Department of Automation, Technical University, Budapest

(Received July 10, 1973)

Presented by Prof. F. CSÁKI

Introduction

In control techniques the Laplace transformation is the most frequent method for determining transient responses and investigating the dynamical characteristics of elements or systems. By means of Laplace transformation, the differential equations or the set of differential equations describing linear invariant systems can be converted into algebraic equations or set of equations, respectively, drawn about the behaviour of systems. In spite of the fact that returning from the operator domain into the time domain — that is, the inverse Laplace transformation — is theoretically well established and solved, its performance may encounter difficulties as to the calculus procedure. That is only why we considered it advisable to develop a computing algorithm to perform inverse Laplace transformation, that is, to determine time function. In case of distinct poles a very simple and rapid algorithm or rather a complete ALGOL program is described in [5]. Another computing program described in [21] can be used in case of at most threefold multiplicity. This method can be expanded in theory for multiplicities higher than three, but it demands a lot of computing. DUBNER and ABATE [5], ZAKIAN [23, 24, 25], STEHFEST [19, 20] and PIESSENS [16, 17] carried out inverse Laplace transformation on different theoretical bases and their results had been compared by ATKINSON and LANG [1].

1. *Determination of residues*

In most cases it is the rational fractional functions whose inverse Laplace transform must be determined in control engineering practice. Then the fractional function must be factored and the partial fraction coefficients must be determined. Contrary to mathematics here these coefficients will be called. A brief historical review will be given below of the applications of this method, without claim to completeness.

POTTLE [18] has developed the digital computer oriented version of the pioneering iterative method due to MOSKOWITZ—RACKER [12]. KUO—KAISER [11] proposed a simpler and more suitable method but some of its inconveniencies have to be mentioned, e. g. transforming the multiple poles into the origo, power series expansion, error accumulation and running-time consumption. BRUGIA [2] suggested a non-iterative method for calculating the residues independently of each other. The main disadvantage of this method is that even the numerator must be factored. CHEN and HAAS [3] and CHEN and SHIEH [4] worked out a generalized matrix method. The method described below is based upon C. W. VALENTINE's paper [22].

Let the Laplace transformation of the output signal in the system have the following form:

$$C_k(s) = \frac{N(s)}{D(s)} = \frac{\sum_{i=0}^{m} a_i \cdot s^i}{\sum_{j=0}^{n} b_j \cdot s^j} = \frac{N(s)}{\prod_{j=1}^{P} (s-s_j)^{\lambda_j}} = \frac{N(s)}{(s-s_i)^{\lambda_i} \cdot \prod_{\substack{j=1 \\ j\neq i}}^{P-1} (s-s_j)^{\lambda_j}} = \tag{1}$$

$$= \frac{N(s)}{(s-s_i)^{\lambda_i} \cdot D_i(s)} = \sum_{j=1}^{P} \sum_{k=1}^{\lambda_j} \frac{C_{jk}}{(s-s_j)^{\lambda_j-k+1}} .$$

Our purpose is to get the residues C_{jk} by a recursive way. The coefficients C_{j1} can be defined as follows:

$$C_{j1} = \frac{N(s)}{D_j(s_j)} \qquad j = 1, 2 \ldots n \tag{2}$$

where s_j denotes the j-th pole, and:

$$D_i(s) = \prod_{\substack{j=1 \\ j\neq i}}^{P-1} (s - s_j)^{\lambda_j} . \tag{3}$$

Our statement is that the residues C_{j2} can be determined by means of the formula:

$$C_{j2} = \frac{N'(s_j) - C_j \cdot D_j'(s_j)}{D_j(s_j)} \tag{4}$$

where superscript denotes the derivates of s.

Proof

Expressing the term containing C_{j2} from Eq. (1):

$$\frac{C_{j2}}{(s-s_j)^{\lambda_j-1}} = \frac{N(s)}{D_j(s)\cdot(s-s_j)^{\lambda_j}} - \sum_{\substack{i=1 \\ (i\neq j1}}^{P} \sum_{\substack{k=1 \\ k\neq 2)}}^{\lambda_i} \frac{C_{ik}}{(s-s_i)^{\lambda_i-k+1}} \tag{5}$$

and rearranging

$$C_{j2} = \lim_{s \to s_j} \left[\frac{N(s)}{(s - s_j) \cdot D_j(s)} - \frac{C_{j1}}{(s - s_j)} \right] = \lim_{s \to s_j} \left[\frac{N(s) - C_{j1} \cdot D_j(s)}{(s - s_j) \cdot D_j(s)} \right] . \quad (6)$$

Using the L'Hospital rule C_{j2} is given by

$$C_{j2} = \frac{N'(s_j) - C_{j1} \cdot D_j'(s_j)}{D_j(s_j)} = \frac{d}{ds} \left(\frac{N(s)}{D_j(s)} \right)_{s = s_i} . \quad (7)$$

Thereby statement (4) has been verified.

Generalizing the above train of thought for C_{ji}, the following formula is obtained:

$$C_{jk} = \frac{N^{(k-1)}(s_j) - \sum_{i=2}^{k-1} \frac{(k-1)!}{(k-i)!} \cdot C_{ji} \cdot D_j^{(k-i)}(s_j)}{(k-1)!\, D_j(s_j)} . \quad (8)$$

Although this formula is a recursive term suitable for determining the residual coefficient, it is advisable to rearrange our formula in view of the programming work. The computation of term (8) is difficult because of the need to determine the derivates of the numerator and their substitutive values.

The Taylor's series of the numerator and the denominator about the i-th pole is given by:

$$\begin{aligned} N(s) &= p_0 + p_1(s - s_i) + p_2(s - s_i)^2 + \ldots + p_{\lambda_i} \cdot (s - s_i)^{\lambda_i} + \ldots \\ D(s) &= q_0 + q_1(s - s_i) + q_2(s - s_i)^2 + \ldots + q_{\lambda_i} \cdot (s - s_i)^{\lambda_i} + \ldots \end{aligned} \quad (9)$$

They lead to the formula fundamental to the program in the Appendix:

$$C_{jk} = \frac{p_{k-1} - \sum_{i=2}^{k-1} C_{ji} \cdot q_{k-i}}{q_0} . \quad (10)$$

The inverse Laplace transform of $C_k(s)$ can be written with the help of the residues as follows:

$$C_k(t) = \sum_{j=1}^{P} \sum_{k=1}^{\lambda_j} \frac{C_{jk}}{(\lambda_j - k)!} \cdot t^{\lambda_j - k} \cdot e^{s_j t} \quad (11)$$

writing the root s_i in the form $s_i = \sigma_i + jw_i$, the final expression of the time function is:

$$C_k(t) = \sum_{j=1}^{P} \sum_{k=1}^{\lambda_j} \frac{t^{\lambda_j - i}}{(\lambda_j - k)!} \cdot e^{\sigma_j t} \cdot [\operatorname{Re} C_{jk} \cos(w_j t) - \operatorname{Im} C_{jk} \sin(w_j t)] . \quad (12)$$

Although this expression is less known, it is rather practical as it implies real arithmetic.

2. *Description of the program*

In accordance with the above mentioned we have written a digital program permitting to determine the inverse Laplace transform of any expression $C_k(s)$ given by a rational fractional function. The inverse Laplace transformation or more exactly the determination of the points of function $c_k(t)$ in possession of coefficients C_{jk} is carried out by the INVL procedure. This constitutes the core of the program. The flow-chart shown in Fig. 1 explains the

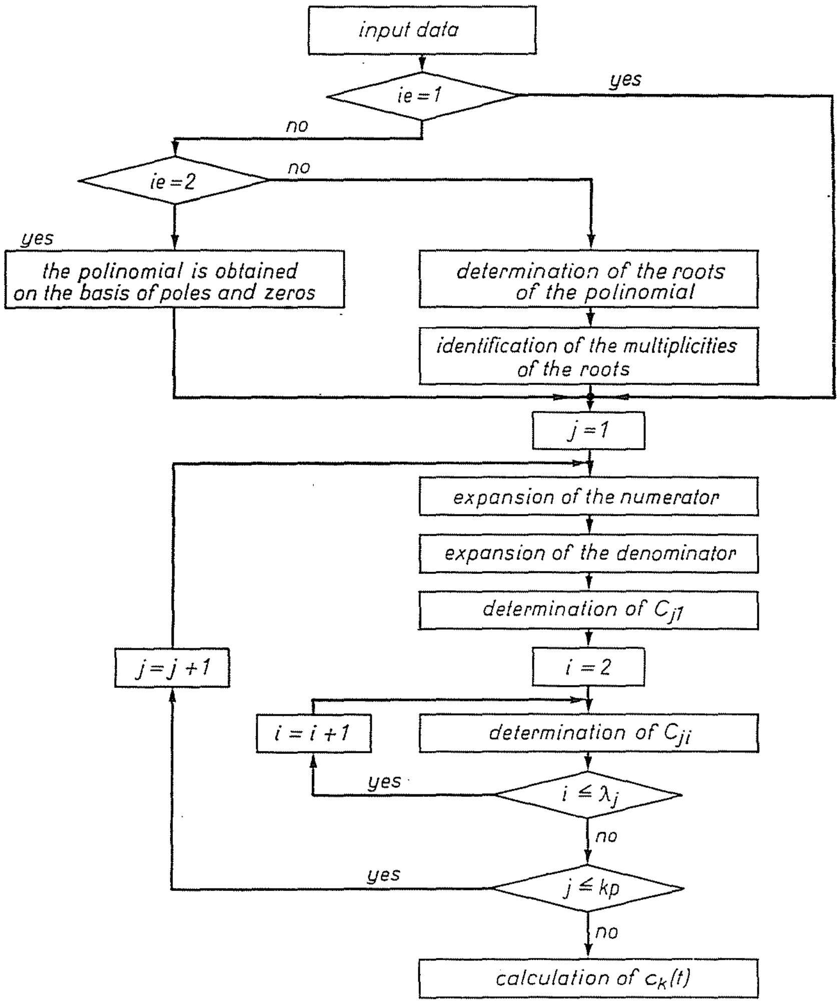

input data
ie = 1
yes
no
ie = 2
no
yes
the polinomial is obtained on the basis of poles and zeros
determination of the roots of the polinomial
identification of the multiplicities of the roots
j = 1
expansion of the numerator
expansion of the denominator
determination of C_{j1}
j = j + 1
i = 2
i = i + 1
determination of C_{ji}
yes
$i \leq \lambda_j$
no
yes
$j \leq kp$
no
calculation of $c_k(t)$

Fig. 1. The flow-chart of the program for inverse Laplace transformation

operational principle of the procedure. The formal parameters of the procedure are as follows:

ie	= specifies how to give the input parameters of the program. It equals one, if the poles and the zeros of the function $C_k(s)$ as well as the polynomials of the numerator and denominator are known. Its value is two, if only the poles and zeros are known, otherwise only the polynomials are known (i),
nn	= order of numerator (i),
nd	= order of denominator (i),
a	= coefficients of numerator, polynomial $[0 : nn]$, $a[0]$ is the coefficient of the highest order term of the numerator,
b	= coefficients of denominator polynomial $[0 : nd]$, $b[0]$ is the coefficient of the highest order term of the denominator,
iz	= number of zeros in the numerator (i),
ip	= number of distinct poles of $C_k(s)$, (i),
pz	= array of poles and zeros of $C_k(s)$, $[1 : iz+ip, 1 : 2]$,

$$pz = \begin{array}{c} \begin{array}{cc} re & im \end{array} \\ \left[\begin{array}{cc} \cdots & \cdots \\ \cdots & \cdots \\ \vdots & \vdots \\ \cdots & \cdots \\ \cdots & \cdots \end{array}\right] \end{array} \begin{array}{l} \left.\vphantom{\begin{array}{c}a\\a\end{array}}\right\} iz \\ \left.\vphantom{\begin{array}{c}a\\a\\a\end{array}}\right\} ip \end{array}$$

mmu	= maximal multiplicity of denominator (i),
mu	= array of multiplicities of poles and zeros $[1 : iz+ip]$, multiplicities of zeros are always one,
ti	= initial time value (r),
dt	= increment of time (r),
te	= final time (r),
fo	= initial value of $c_k(t)$, (r),
fv	= final value of $c_k(t)$, (r),
tp	= points of $c_k(t)$, (i),
fi	= array of time values, $[1 : tp]$,
ft	= array of values of $c_k(t)$, $[1 : tp]$,
cor	= array of real parts of residues, $[1 : ip, 1 : mmu]$,
coi	= array of imaginary parts of residues, $[1 : ip, 1 : mmu]$.

The procedure can operate with three different kinds of data presentation depending on the value of parameter "*ie*". If the pole-zero configuration and polynomials of the numerator and denominator are known, the residues can directly be calculated.

If only the pole-zero configuration is known, first the ZEPO procedure yields the polynomial coefficients.

The formal parameters of the ZEPO procedure are as follows:

ze = is the array containing the roots of the polynomial to be determined. The real parts of the roots are in the first, and the imaginary ones in the second column. The multiple roots must be written in a separate row, $[1:n, 1:2]$,
n = number of roots (i),
po = array of the coefficients of the polynomial, $[0:m]$,
$po[0]$ is coefficient of the highest order term of the polynomial,
m = power of the polynomial, (i).

The ZEPO procedure involves the MULT procedure, which determines the product of two polynomials.

If only the polynomials are known, the roots of the polynomials are determined by ROOT procedure, and by their investigation the multiplicity of the roots.

The ROOT procedure has the following formal parameters:

a = array of the polynomial coefficients, $[0:n]$,
$a[0]$ is coefficient of the highest order term of the polynomial,
n = order of the polynomial, (i),
x = array of the real parts of the roots of the polynomial, $[0:n]$,
y = array of the imaginary part of the roots of the polynomial, $[0:n]$,
eps = relative error solution, (r),

It follows naturally from the three kinds of data presentation that not every formal parameter of the INVL procedure will be assigned arbitrary value. Thus the pattern and the sequence of the data tape have the following form:

m = order of numerator, (i),
n = order of denominator, (i),
kz = number of distinct zeros of numerator (i),
kp = number of distinct roots of denominator (i),
$mult$ = maximal multiplicity of the denominator, (i),
ien = integer variable controlling the data presentation, (i),
ti = initial time value, (r)
dt = increment of time, (r),
te = final time, (r),
a = array of the polynomial coefficients of numerator $[0:m]$,
b = array of the polynomial coefficients of denominator $[0:n]$,
} if $ien = 2$ then these are not needed

$mul\ [i]$ multiplicity
$poz\ [i, 1]$ real part of poles and zeros
$poz\ [i, 2]$ imaginary part of poles and zeros
} $i = 1, 2, \ldots, kz + kp$ } if $ien = 2$ and $ien = 1$ then these are not needed

3. *Example*

To support this method and the adaptability of the program let us compute the inverse Laplace transform of the function given by BRUGIA [2]:

$$C_k(s) = \frac{s \cdot (s+3)^4}{(s+1)^6 \cdot (s+2) \cdot (s+1+j)^3 \cdot (s+1-j)^3} \,. \qquad (13)$$

This is a rather extreme function nevertheless very suitable to demonstrate the versatility of the program. If beyond the pole-zero configuration also the polynomials of the numerator and denominator are known, the values on the data tape are as follows:

5, 13, 5, 4, 6, 1, 0.1, 0.05, 20,
1, 12, 54, 108, 81, 0, (coefficients of the numerator)

$$\left.\begin{array}{l} 1,\ 14,\ 93,\ 388,\ 1133,\ 2442,\ 3991,\ 5000,\ 4794,\ 3468, \\ 1836,\ 672,\ 152,\ 16, \end{array}\right\} \text{(coefficients of the denominator)}$$

$$\left.\begin{array}{rrr} 1, & 0, & 0, \\ 1, & -3, & 0, \\ 1, & -3, & 0, \\ 1, & -3, & 0, \\ 1, & -3, & 0, \end{array}\right\} \text{all zeros}$$

$$\left.\begin{array}{rrr} 6, & -1, & 0, \\ 1, & -2, & 0, \\ 3, & -1, & -1, \\ 3, & -1, & -1, \end{array}\right\} \text{distinct poles}$$

The resulting time function $c(t)$ is seen in Fig. 2. In Table 1 the residues are given one by one. With their help the time function can be produced even analytically from Eq. (12).

If the term (13) is given in polynomial form the ROOT procedure yields the following roots:

$$\begin{array}{lcll}
p_1 & = & -1.00174 - j0.01172 & \simeq -1 \\
p_2 & = & -1.00174 + j0.01172 & \simeq -1 \\
p_3 & = & -0.99684 - j0.00466 & \simeq -1 \\
p_4 & = & -0.99684 + j0.00466 & \simeq -1 \\
p_5 & = & -0.98941 & \simeq -1 \\
p_6 & = & -1.01342 & \simeq -1 \\
p_7 & = & -2.00000 & \simeq -2 \\
p_8 & = & -0.99886 - j0.99968 & \simeq -1 - j \\
p_9 & = & -0.99886 + j0.99968 & \simeq -1 + j \\
p_{10} & = & -1.00085 - j0.99917 & \simeq -1 - j \\
p_{11} & = & -1.00085 + j0.99917 & \simeq -1 + j \\
p_{12} & = & -1.00029 - j1.00115 & \simeq -1 - j \\
p_{13} & = & -1.00029 + j1.00115 & \simeq -1 + j
\end{array}$$

The procedure obviously produces the multiple roots of the polynomial of 13th order at a relatively great accuracy. Knowing the roots the program states their multiplicities, yielding in turn the residues.

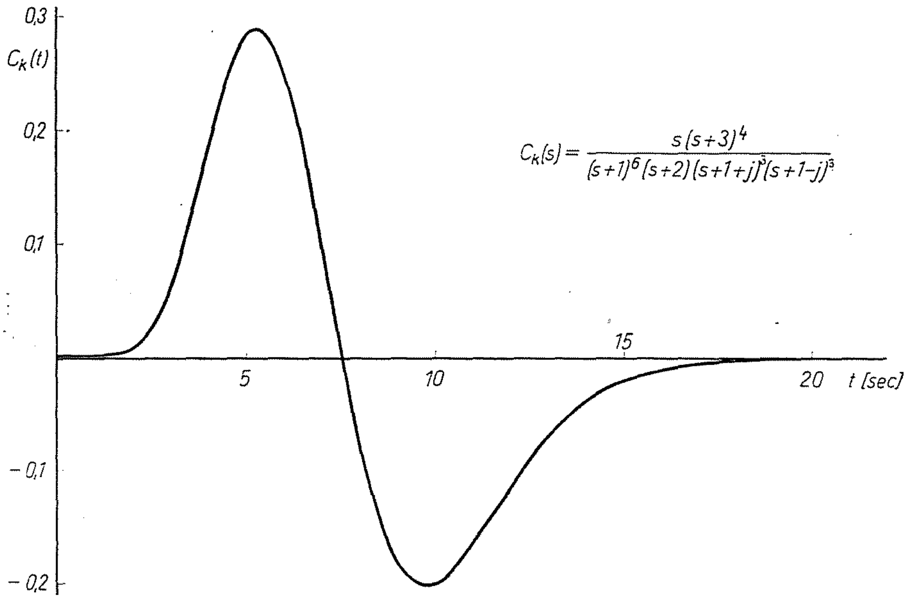


Fig. 2

Table 1

Poles	Multiplicity	Re C_{ij}					
		1	2	3	4	5	6
-1	6	-16	0	56	8	-121	-22
-2	1	0.25	0	0	0	0	0
$-1-j$	3	-0.875	-20.625	11.125	0	0	0
$-1+j$	3	-0.875	-20.625	11.125	0	0	0

Poles	Multiplicity	Im C_{ij}					
		1	2	3	4	5	6
-1	6	0	0	0	0	0	0
-2	1	0	0	0	0	0	0
$-1-j$	3	-3	4.0625	81	0	0	0
$-1+j$	3	3	$-$ 4.0625	-81	0	0	0

It is worth mentioning in connection with the calculating of the residues that both the numerator and the denominator have been expanded into series by a version of the well-known HORNER's method [15], extended to complex form.

The time function shown in Fig. 2 can be considered as the impulse response of a control loop described by $C(s)$ (13). It may be instructive to show also the step response of the same control loop in Fig. 3, again determined by means of the inverse Laplace transformation. The residual coefficients are given in Table 2.

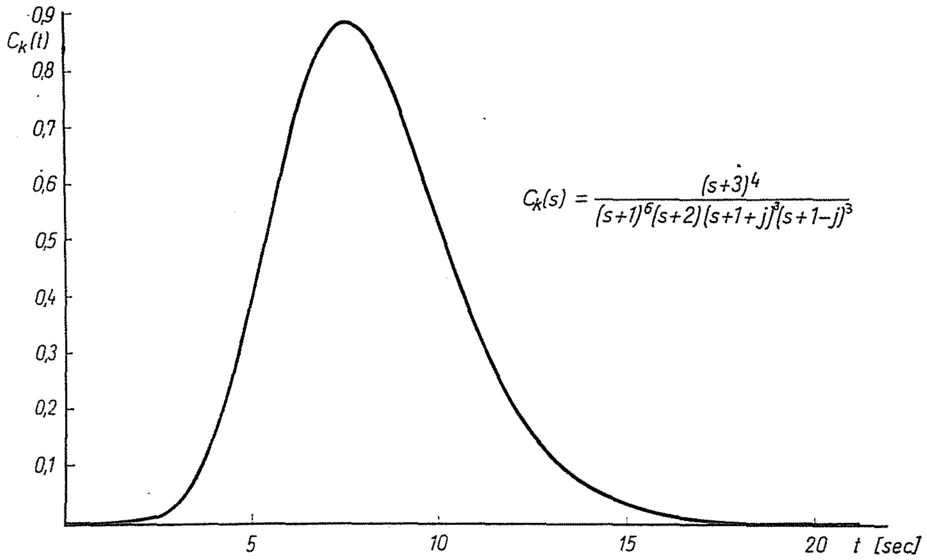


Fig. 3

Table 2

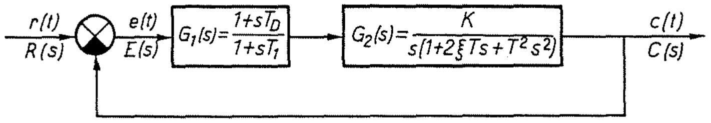
Poles	Multiplicity	Re C_{ij} 1	2	3	4	5	6
-1	6	16	16	-40	-48	73	95
-2	1	0.125	0	0	0	0	0
$-1-j$	3	1.9375	9.7812	-47.562	0	0	0
$-1+j$	3	1.9375	9.7812	-47.562	0	0	0

Poles	Multiplicity	Im C_{ij} 1	2	3	4	5	6
-1	6	0	0	0	0	0	0
-2	1	0	0	0	0	0	0
$-1-j$	3	1.0625	-12.781	-46.219	0	0	0
$-1+j$	3	-1.0625	12.781	46.219	0	0	0

4. *Systems analysis with inverse Laplace transformation*

Our Laplace transform inversion program is very suitable for analysing control loops. This will be demonstrated on example:

Let us consider the control loop in Fig. 4. The transfer functions of



r(t)
R(s)
e(t)
E(s)
$G_1(s)=\frac{1+sT_D}{1+sT_1}$
$G_2(s)=\frac{K}{s(1+2\xi Ts+T^2s^2)}$
c(t)
C(s)

Fig. 4. Block representation of the control loop

the two elements are:

$$G_1(s) = \frac{1+sT_D}{1+sT_1} = 2\,\frac{s+1}{s+2} \tag{14}$$

$$G_2(s) = \frac{K}{s(1+2\xi Ts+T^2s^2)} = \frac{0.5}{s(s+0.5-j3)\,(s+0.5+j3)} \tag{15}$$

where $T_D = 1$ [sec], $T_1 = 0.5$ [sec], $T = 0.329$ [sec], $\xi = 0.16425$, $K = 4.625$.

The transfer function of the open loop is

$$G(s) = G_1(s)\cdot G_2(s) = \frac{s+1}{s(s+2)\,(s+0.5+j3)\,(s+0.5-j3)}\,. \tag{16}$$

If the input signal to the open loop system is $\delta(t)$ or $1(t)$, the Laplace transform of the output signal is

$$C_{k1}(s) = G(s) \qquad \text{if} \qquad r(t) = \delta(t) \tag{17}$$

and

$$C_{k2}(s) = \frac{1}{s}\,G(s) \qquad \text{if} \qquad r(t) = 1(t) \tag{18}$$

respectively.

With the help of the program described here we have determined their Laplace transforms. In Fig. 5 the function $c_{k1}(t)$ (that is, the impulse response of the open loop) and in Fig. 6 the function $c_{k2}(t)$ (that is, the step response of the open loop) are shown.

As the open loop is integrating type, the function $c_{k2}(t)$ will monotonously increase in time after decay of the transient part. In Tables 3 and 4 the residual coefficients are given, and with their help the time functions can be written even in analytical form, e. g.:

$$\begin{aligned} c_{k1}(t) = {} & 0.054054 + 0.04444\cdot e^{-2t} + \\ & e^{-0.5t}\cdot[0.049249\cdot\cos(3t) + 0.0066066\cdot\sin(3t)] + \\ & e^{-0.5t}\cdot[0.049249\cdot\cos(-3t) + 0.0066066\cdot\sin(-3t)]\,. \end{aligned} \tag{19}$$

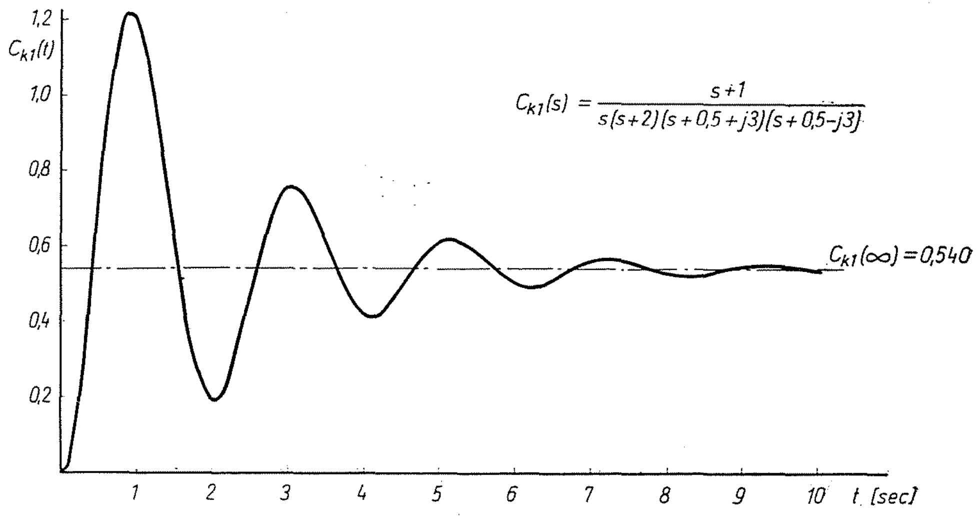


Fig. 5. Impulse response of the open loop system

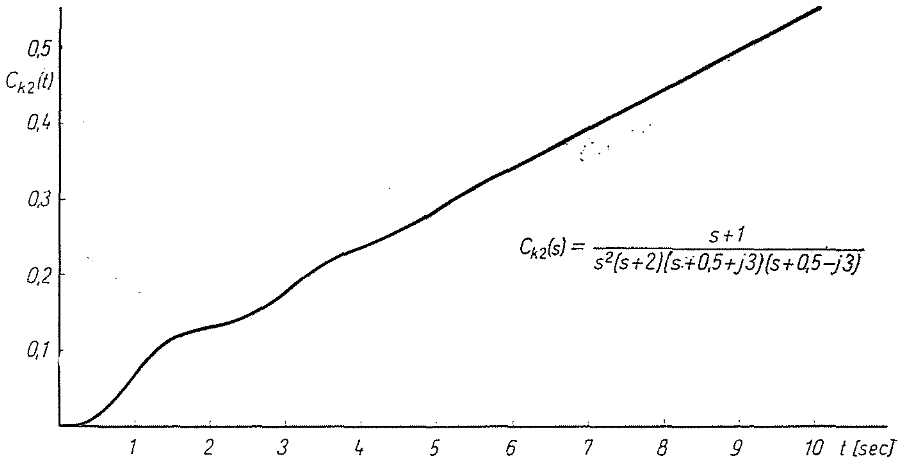


Fig. 6. Step response of the open loop system

Table 3

Poles	Multiplicity	Re C_{ij}	Im C_{ij}
0	1	0.054054	0
-2	1	0.044444	0
$-0.5 + j\,3$	1	-0.049249	-0.0066066
$-0.5 - j\,3$	1	-0.049249	0.0066066

The resultant transfer function of the closed loop system is:

$$W(s) = \frac{G(s)}{1+G(s)} = \frac{s+1}{s^4+3s^3+11.25s^2+19.5s+1}\,. \tag{20}$$

The poles of the system can be determined by the ROOT procedure:

$$\begin{aligned} p_1 &= -0.052873\,, \\ p_2 &= -2.0449\,, \\ p_3 &= -0.45113 - j3.0076\,, \\ p_4 &= -0.45113 + j3.0076\,. \end{aligned} \tag{21}$$

It is seen that the feedback changed the integrating type into proportional type and also the poles have got other values. Excitation by $\delta(t)$ and by $1(t)$ of the resulting system yields impulse and step function, respectively. These results are plotted in Figs 7 and 8, the residual coefficients are given in Tables 5 and 6. On the basis of these functions characterizing the system the analysis can be carried out and even system performances can be determined.

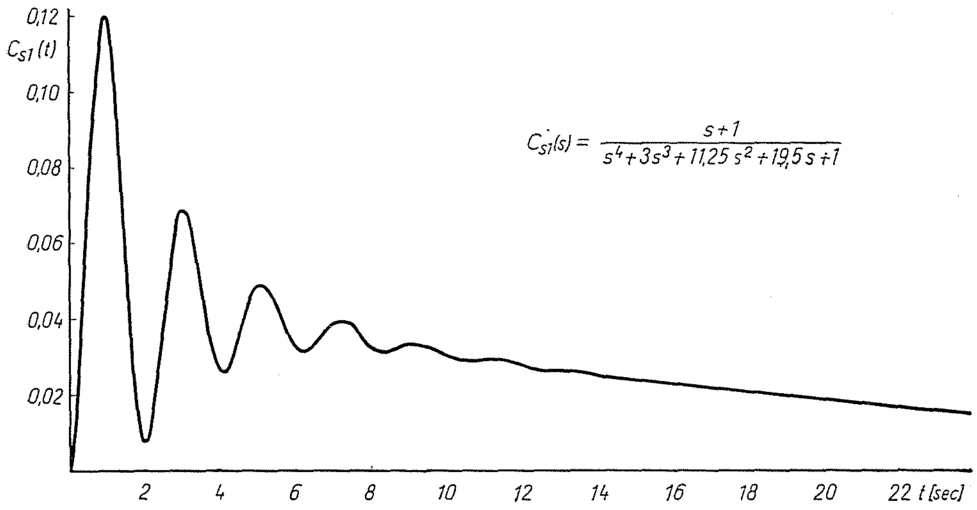


Fig. 7. Impulse response of the closed loop system

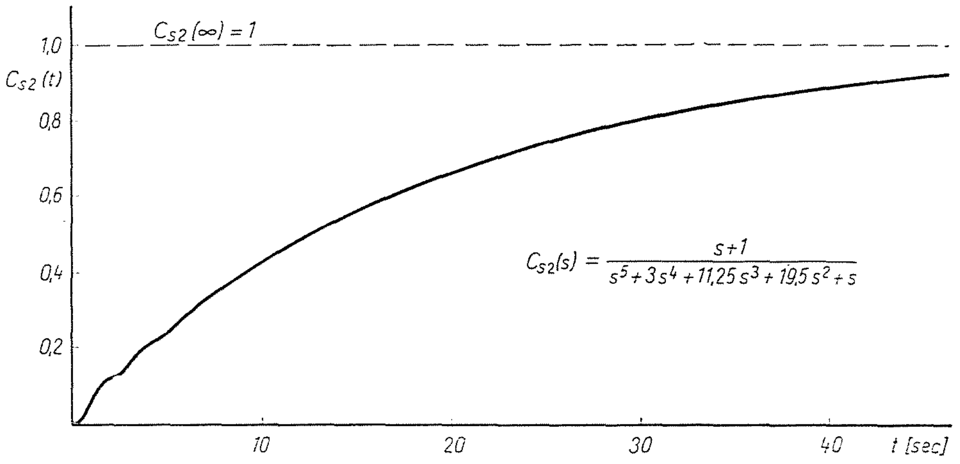


Fig. 8. Step response of the closed loop system

Table 4

Poles	Multi-plicity	Re C_{ij}		Im C_{ij}	
		1	2	1	2
0	2	0.054054	0.021183	0	0
-2	1	-0.022222	0	0	0
$-0.5 + j\,3$	1	0.00051944	0	0.016330	0
$-0.5 - j\,3$	1	0.00051944	0	-0.016330	0

Table 5

Poles	Multi-plicity	Re C_{ij}	Im C_{ij}
-0.0529	1	0.051657	0
-2.0449	1	0.045274	0
$-0.4511 + j\,3.0076$	1	-0.048466	-0.0085753
$-0.4511 - j\,3.0076$	1	-0.048466	0.0085753

Table 6

Poles	Multi-plicity	Re C_{ij}	Im C_{ij}
0	1	1.00000	0
-0.0529	1	-0.97600	0
-2.0449	1	-0.02214	0
$-0.4511 + j\,3.0076$	1	-0.00042542	0.016178
$-0.4511 - j\,3.0076$	1	-0.00042542	-0.016178

Summary

Recursive relationships of factoring the rational fractional functions by means of the TAYLOR expansion of the numerator and the denominator of the transfer function are briefly described. These helped to establish a complete ALGOL program for determining the inverse Laplace transform of the rational fractional functions with arbitrary multiplicity. In addition to the program two examples are shown to demonstrate its way of operation.

APPENDIX

```
begin
integer i, j, m, n, kp, kz, mult, ien, it, po, type, k1, k2, k3;
real ti, dt, te, fo, fv, del, a1, a2, t;
input(m, n, kz, kp, mult, ien, ti, dt, te, del);
comment This program evaluates the inverse Laplace transform from the transfer function
        G(s) given in term of rational fractional function with multiple poles.;
it=entier ((te−ti)/dt)+1;
begin
integer array mul[1:kp+kz];
real array a[0:m], b[0:n], poz[1:kp+kz, 1:2];
real array fi, ft[1:it], cor, coi[1:kp,1:mult];

procedure KOM1(a1, a2, b1, b2, c1, c2, in);
value a1, a2, b1, b2, in; integer in;
begin integer i; real x1,x2; switch s=s1, s2, s3, s4;
  goto s[in];
  s1: c1:=a1+b1; c2:=a1+b2; goto vege;
  s2: c1:=a1−b1; c2:=a2−b2;  goto vege;
  s3: c1:=a1×b1−a2×b2; c2:=a2×b1+a1×b2; goto vege;
  s4: x1:=b1×b1+b2×b2;
        if x1<₁₀−30 then begin lines 5;
        text The complex number is divided by zero.; lines 5
        goto vege end if;
        c1:=(a1×b1+a2×b2)/x1; c2:=(a2×b1−a1×b2)/x1;
  vege:
end of procedure KOM1;

procedure MULT(pol1, i, n1, pol2, j, n2, res, k, nr);
value n1, n2; integer n1, n2, nr, i, j, k; real pol1, pol2, res;
begin  integer ir;
  nr:=n1+n2;  ir=0;
  for k:=0 step 1 until nr do res:=0;
  for j:=0 step 1 until n2 do
     begin k:=ir;
       for i:=0 step 1 until n1 do
          begin
             res:=res+pol1×pol2;   k:=k+1
          end i;
     ir:=ir+1
     end
end of procedure MULT;

procedure ZEPO(ze, n, po, m);
value n;  integer n, m;  array ze, po;
begin  integer k, p, j, i, n1, r1; array ab[0:2], r[0:2×n];
  n1:=0; po[0]:=1; m:=0;
  for i:=1 step 1 until n do
     begin
        if abs(ze[i,2])≤₁₀−20 then
          begin
             ab[0]:=−ze[i,1]; ab[1]:=1; m:=m+1;
          MULT(ab[j], j, 1, po[k], k, nl, r[p], p, rl)
        end else
        begin
          ab[0]:=ze[i,1]×ze[i,1]+ze[i,2]×ze[i,2];
          ab[1]:=−2×ze[i,1]; ab[2]:=1; m:=m+2;
          MULT(ab[j], j, 2, po[k], k, nl, r[p], p, rl)
        end if;
      nl:=rl;
      for j:=0 step 1 until rl do
        po[i]:=r[i]
    end i;
  for i:=0 step 1 until m do
    po[i]:=r[m−i]
end of procedure ZEPO;

procedure HOR1(n, a, k, r, x1, x2);
value n, k, x1, x2; integer n, k; real x1, x2; array a, r;
begin  integer i, j, p;  real r1, c1, c2;
  r1:=a[0];
  for i:=0 step 1 until k do
    begin
      r[i, 1]:=r1; r[i, 2]:=0
    end i;
  for j:=1 step 1 until n do
    begin
      c1:=r[0,1]×x1−r[0,2]×x2;
      c2:=r[0,2]×x1+r[0,1]×x2;
      r[0,1]:=c1; r[0,2]:=c2;
      p:= if n−j<k then k else n−j;
      for i:=1 step 1 until p do
        begin
            c1:=r[i, 1]×x1−r[i, 2]×x2;
            c2:=r[i, 2]×x1+r[i, 1]×x2;
            r[i, 1]:=c1; r[i, 2]:=c2;
            r[i, 1]:=r[i, 1]+r[i−1,1];
            r[i, 2]:=r[i, 2]+r[i−1,2]
        end i
    end j
end of procedure HOR1;

procedure ROOT(a, n, x, y, eps);
value n, eps; integer n; real eps; array a, x, y;
begin  integer i, j, p, nl;
real u, v, w, k, m, f, fm, fc, xm, ym, xr, yr, xc, yc, dx, dy, err,
  p:=n;  err:=eps ↑ 2;
  for i:=0 step 1 until p do
    x[i]:=a[i];
  REP: nl:=p−1; xc:=yc:=0; fc:=x[p] ↑ 2;
  dx:=abs(x[p]/x[0]) ↑ (1/p); dy:=0;
  ITER: fm:=fc×2+1;
  for i:=1, 2, 3, 4 do
    begin
      u:=−dy; dy:=dx; dx:=u; xr:=xc+dx;
      yr:=yc+dy; k:=2×xr; m:=xr ↑ 2+yr ↑ 2;  u:=v:=0;
      for j:=0 step 1 until nl do
          begin
            w:=x[j]+k×u−m×v; v:=u; u:=w
          end j;
      f:=(x[p]+u×xr−m×v) ↑ 2+(u×yr) ↑ 2;
      if f≤fm then
          begin
            ym:=yr; xm:=xr; fm:=f
          end if
      end i;
    if fm≤fc then
      begin
        dx:=1.5×dx:  dy:=1.5×dy;
        xc:=xm;  yc:=ym;  fc:=fm
      end else
      begin
        u:=0.4×dx−0.3×dy;  dy:=0.4×dy+0.3×dx;  dx:=u
      end if;
    if (dx ↑ 2+dy ↑ 2)<(xc ↑ 2+yc ↑ 2)×err  ∧  fc≠0 then goto ITER;
    u:=v:=0; k:=2×xc; m:=xc ↑ 2;
    for j:=0 step 1 until n1 do
      begin
        w:=x[j]+k×u−m×v; v:=u; u:=w
      end j;
    if (x[p]+u×xc−m×v) ↑ 2≤fc then
      begin
        for j:=1 step 1 until n1 do
          x[j]:=x[j−1]×xc+x[j];
        x[p]:=xc; y[p]:=0; p:=p−1
      end else
      begin
        k:=2×xc; m:=xc ↑ 2+yc ↑ 2; x[1]:=x[1]+k×x[0];
        for j:=2 step 1 until p−2 do
          x[j]:=x[j]+k×x[j−1]−m×x[j−2];
        x[p−1]:=x[p]:=xc;
        y[p]:=yc; y[p−1]:=−yc; p:=p−2
      end if;
    if p>0 then goto REP;
    x[0]:=y[0]:=0
  end of procedure ROOT;

  procedure INVL(ie, nn, nd, a, b, iz, ip, pz, mmu, mu, ti, dt, te, fo, fv, tp, fi, ft, cor, coi, delt);
  value ie, nn, nd, ti, dt, te, delt; integer ie, nn, nd,i z, ip, mmu, tp;
  real ti, dt, te, fo, fv, delt; integer array mu; array a, b, pz, fi, ft, cor, coi;
  begin
    integer i, j, k, n1, n2, li, nq, mp, hp, hp1, hq, hq1, iz1, zp, r1, r2, np, kc:
    real eps, s1, s2, x1, x2, y1, y2, t, fa, s;
    integer array mu1[1:ip];
    real array pr, pi[0:nd], zr, zi[0:nn], zn[1:nn, 1:2], zd[1:nd, 1:2]; iz1=iz+1; zp=iz+ip;
    comment It follows determination of the initial and final value of time function c(t);
    if nn=nd then fo=−1 else
      begin
        if nn−1=nd then fo:=a[nn]/b[nd] else fo:=0
      end;
    n1:=n2:=0;
    for i:=0 step 1 until nn do
      if abs(a[nn−i])<₁₀−20 then n1:=n1+1;
    for j:=0 step 1 until nd do
      if abs(b[nd−j])<₁₀−20 then n2:=n2+1;
    if n1≤n2 then fv:=0 else
      begin
        if n1+1=n2 then fv:=a[nn−n1]/b[nd−n2] else fv:=−1
      end;
    We have finished the determination the initial and final time value of time function c(t);
    if ie=1 then goto CIM1 else
      if ie=2 then goto CIM2;
    eps:=₁₀−10;
    comment Only the polinomials are known;
    if nn=0 then goto CIM3;;
    if nn=1 then
      begin
        pz[1, 1]:=−a[1]/a[0]; pz[1, 2]:=0; mu[1]:=1; iz:=1; goto CIM3;
    end;
  comment Roots of numerator are calculated;
  ROOT(a, nn, zr, zi, eps);
  for i:=1 step 1 until iz do
    begin
      pz[i, 1]:=zr[i]; pz[i, 2]:=zi[i]
    end;
  CIM3: if nd=1 then:
    begin
      pz[izl, 1]:=—b[1]/b[0]; pz[izl, 2]:=0×; mu[izl]:=1; ip:=1; goto CIM1
    end;
  comment Roots of denominator are calculated;
  ROOT(b, nd, pr, pi, eps);
  for i:=1 step 1 until zp do mu[i]:=1;
  for j:=1 step 1 until ip do mul[i]:=1;
  for i:=1 step 1 until nd do
    if abs(pi[i])<delt then pi[i]:=0;
  ip:=0;
  comment It follows the determination of multiplicities of zeros of denominator;
  for i:=1 step 1 until nd do
    begin
      if mul[i]=0 then goto CI11;
      ip:=ip+1; zp:=iz+ip; kc:=0; rl:=izl+ip;
      mu[zp]:=1; pz[zp,1]:=pr[i]; pz[zp, 2]:=pi[i];
      if i=nd then goto CI11;
      for j:=i+1 step 1 until nd do
          begin
            if mul[j]=0 then goto CI14;
             if (abs(pr[i]—pr[j])<delt) ∧ (abs(pi2i]—pi[j])<delt) then
              begin
                mul[j]:=0; mu[zp]:=mu[zp]+1;
                pz[zp, 1]:=pz[zp, 1]+pr[j]; pz[zp, 2]:=pz[zp, 2]+pi[j]; goto CI14
              end;
            if (abs(pr[i]—pr[j])<delt) ∧ (abs(pi[i]+pi[j])<delt) then
              begin
                mul[j]:=0; kc:=1; mu[rl]:=mu[zp];
                pz[rl, 1]:=pz[zp, 1]; pz[rl, 2]:=p—pz[zp, 2]
              end;
            CI14:
          end;
      if kc=1 then ip:=ip+1;
      CI11:
    end;
comment The average of poles with multiplicities are calculated;
  for i:=1 step 1 until ip do
    begin
      rl:=i+iz; pz[rl,1]:=pz[rl,1]/mu[rl];
      pz[rl,2]:=pz[rl,2]/mu[rl]
    end;
  mmu:=mu[izl]; k:=izl;
  for i:=2 step 1 until ip do
    if mu[i+iz]≤mu[k] then
        begin
          k:=iz+1; mmu:=mu[i+iz]
        end;
  zp:=iz+ip; goto CIM1;
  CIM2:
  comment Only the zeros and poles are known;
  for i:=1 step 1 until iz do
    begin
      zn[i,1]:=pz[i,1]; zn[i,2]:=pz[i,2]
    end;
  comment The coefficiens of polynomials are calculated;
```

```
ZEPO(zn, iz, a, r1);
n1:=0;
for i:=iz1 step 1 until zp do
  begin
    for j:=1 step 1 until mu[i] do
      begin
        if pz[i,2]<0 then goto CIM;
        n1:=n1+1; zd[n1, 1]:=pz[i, 1]; zd[n1, 2]:=pz[i, 2];
        CIM:
      end
  end;
comment The coefficient of denominator is calculated from the roots;
ZEPO(zd, n1, b, r2);
CIM1:
begin real a1, a2, b1, b2, c1, c2, nev;
    integer array fact[0:mmu];
    real array p[0:mmu—1, 1:2], q[0:2×mmu—1, 1:2];
    for i:=1 step 1 until ip do
      for j:=1 step 1 until mmu do
        begin
          cor[i, j]:=coi[i, j]:=0
        end;
    for j:=1 step 1 until ip do
      begin
        x1:=pz[j+iz, 1]; x2:=pz[j+iz, 2];
        hp:=mu[j+iz]; hp1:=hp—1;
        if hp1<nn then
          begin np:=nn;
            for i:=nn+1 step 1 until hp1 do
              begin
                p[i, 1]:=0; p[i, 2]:=0
              end
          end else np:=hp1;
        HOR1(nn, a, np, p, x1, x2);
        hq:=2×hp; hq1:=hq—1;
        if hq1>nd then
          begin np:=nd;
            for i:=nd+1 step 1 until hq1 do
              begin
                q[i, 1]:=0; q[i, 2]:=0
              end
          end else np:=hq1;
        HOR1(nd, b, np, q, x1, x2);
        KOM1(p[0, 1], p[0, 2], q[hp, 1], q[hp, 2], cor[j, 1], coi[j, 1], 4);
        for i:=2 step 1 until hp do
          begin
            y1:=y2:=0;
            for li:=2 step 1 until i do
              begin
                KOM1(q[li+hp1, 1], q[li+hp1, 2], cor[j, i—li+1], coi[j, i—li+1], s1, s2, 3);
                KOM1(s1, s2, y1, y2, y1, y2, 1)
              end;
            KOM1(p[i—1, 1], p[i—1, 2], —y1, —y2, y1, y2, 1);
            KOM1(1, y2, q[hp, 1], q[hp, 2], cor[j, i], coi[j, i], 4)
          end
      end;
    fact[0]:=1;
    for i:=1 step 1 until mmu do
      fact[i]:=i×fact[i—1];
    i:=0;
    for t:=ti step dt until te do
      begin
      fa=0; i:=i+1; fi[i]:=t;
    for j:=1 step 1 until ip do
      begin
        hp:=mu[j+iz]; s1:=pz[j+iz,1]×t; s2:=pz[j+iz,2]×t;
        y1:=cos(s2); y2:=sin(s2); x1:=exp(s1); x1:=0;
        for k:=1 step 1 until hp do
          x2:=x2+t ↑ (hp−k)×(cor[j, k]×y1−coi[j, k]×y2)/fact[hp−k];
        x2:=x1×x2; fa:=fa+x2
      end j;
    ft[i]:=fa; tp:=i
    end t;
  end;
end of procedurei nverse Laplace;
comment After the declaration of the procedures follows the main program;
if ien=2 then goto POLU else
  begin
   input (array a); input (array b)
  end;
if ien≠1 then goto FOLY;
POLU: po:=kp+kz;
for i:=1 step 1 until po do
  begin
   input (mul[i]); input (poz[i, 1]); input (poz[i, 2])
  end;
FOLY:
INVL(ien, m, n, a, b, kz, kp, poz, mult, mul, ti, dt, te, fo, fv, it, fi, ft, cor, coi, del);
text degree of numerator                          m     =;
output (m:3); line;
text number of distinct zeros                     kz    =;
output (kz:3); line;
text degree of denominator                        n     =;
output (n:3); line;
text number of distinct poles                     kp    =;
output (kp:3); line;
text maximum multiplicity of denominator          mul   =;
output (mult:3); line;
text initial time value                           ti    =;
output (ti:3:5); line;
text increment of time                            dt    =;
output (dt:3:5); line;
text final time                                   te    =;
output (te:3:5); line;
text initial value of c(t)                        c(0)  =;
if fo<0 then text infinite else output (fo/5);
text final value of c(t)                          c(∞) =;
if fv<0 then text infinite else output (fv/5); lines 3;
spaces 23; text coefficient of polynomials;
lines 3; spaces 18; text numerator; spaces 15;
text denominator; lines 2;
for i=0 step 1 until n do
  begin
    text s ↑ ; output (i:2);
    if i≤m then
      begin
       spaces 7; output (a[m−i]:5:4); spaces 12;
       output (b[n−i]:5:4); line
      end else
      begin
       spaces 32; output (b[n−i]:5:4); line
      end
    end i;
lines 4; spaces 10; text zeros; spaces 30; text poles; lines 3;
text        re          im            re          im    mult;
lines 2;
```

```
if kz≤kp then
  begin
    k1:=kz; k2:=1; k3:=kp
  end else
  begin
    k1:=kp; k2:=2; k3:=kz
  end;
for i:=1 step 1 until k1 do
  begin
    output (poz[i,1]:3:4); spaces 3; output (poz[i,2]:3:4); spaces 5;
    output (poz[i+kz,1]:3:4); spaces 3;
    output (poz[i+kz,2]:3:4); spaces 3;
    output (mul[i+kz]:2); line
  end i;
for i:=k1+1 step 1 until k3 do
  begin
    if k2=1 then
      begin
        spaces 30; output (poz[i+k1,1]:3:4); spaces 3;
        output (poz[i+k1,2]:3:4); spaces 3; o
        output (mul[i+k1]:1); line
      end else if k2=2 then
      begin
        output (poz[i,1]:3:4); spaces 3; output (poz[i,2]:3:4); line
      end
    end i;
lines 6;
text real parts of the residues:; lines 3;
output (array cor/5); lines 5;
text imaginary parts of residues:; lines 3;
output (array coi/5); lines 5;
text points of the time finction:; lines 3;
i:=0;
for t=ti step dt until te do
  begin
    i:=i+1; output (i:3); spaces 5;
    output (t:3:3); spaces 7;
    output (ft[i]/6); line
  end t;
text The program was run on computer type RAZDAN-3 of University Computing Centre.;
    lines 5;
end end of program
```

References

1. Atkinson, M. P.—Lang, S. R.: A comparison of some inverse Laplace transform techniques for use in circuit design. Computer Journal, **15**, 138—139 (19).
2. Brugia, O.: A Noniterativ Method for Partial-Fraction Expansion of a Rational Function with High Order Poles. SIAM Rev. **7**, No. 3 (1965).
3. Chen, Ch. F.—Haas, I. J.: Elements of Control System Analysis. Prentice Hall (1968).
4. Chen, Ch. F.—Shieh, L. S.: Generalization and Computerization of Heaviside Expansion for Performing the Inverse Laplace Transform of High Order Systems. UEEF Region III CR (1967).
5. Dubner, H.—Abate, J.: Numerical Inversion of Laplace Transform. JACM, **15**, 115—123 (1968).
6. Csáki, F.: Szabályozások dinamikája. Dynamics of Control Systems. Akadémiai Kiadó, Budapest, (1966).
7. Csáki, F.—Kovács, T.: Evaluating the Inverse Laplace Transforms from the Pole-Zero Configuration by Digital Computer. Periodica Polytechnica (Electrical Engineering), **14**, No. 2. 173—180 (1970).
8. Fodor, Gy.: A Laplace-transzformáció műszaki alkalmazása. Technical Applications of Laplace Transformation. Műszaki Kiadó, Budapest (1967).

9. Kovács, S.: Explicit képletek racionális törtfüggvények részlettörtekre bontásához, többszörös gyökök esetén. Explicit Expressions for Partial Fractioning of Rationals with Multiple Poles. Híradástechnika, **20**. No. 7 (1969).
10. Kovács, T.: Lineáris, koncentrált paraméterű, invariáns rendszerek vizsgálata digitális számítógéppel. Műszaki Egyetemi doktori értekezés, Budapest (1971). Analysis of Linear, time invariant systems with concentrated parameters by Digital Computer. Doctoral Dissertation at Technical University of Budapest.
11. Kuo, F.—Kaiser, J. F.: System Analysis by Digital Computer. John Wiley, New York (1966).
12. Moskowitz, S.—Racker, J.: Pulse Techniques. Englewood Cliffs, New Jersey, Prentice Hall (1951).
13. Nagy, F. L. N.—Uraz, A.: Partial-Fraction Expansion of Transfer Function having Multiple Poles. Proc. IEE. **119**, No. 9, 1415—1416 (1972).
14. Nagy, F. L. N.—Tikriti, M. N. Al. Computer Aided System Design using Symbolic Array Technique. IFAC Symposium (1970), Kyoto.
15. Pankiewicz, W.: Calculation of a Polynomial and Its Derivate Values by Horner Scheme. Comm. ACM. **11**, No. 9 (1968).
16. Piessens, R.: Partial-Fraction Expansion and Inversion of Rational Laplace Transforms. Electronic Letters, **5**, No. 5 (1969).
17. Piessens, R.: Reports on the Numerical Inversion of the Laplace Transform. TW1—TW3. Afdeling Toegepaste Wiskunde katolike Universiteit Leuven, Heverlee, Belgium (1969).
18. Pottle, C.: On the Partial Fraction Expansion of a Rational Function with Multiple Poles by Digital Computer. IEEE. Trans. onCT. CT-**11** (1964).
19. Stehfest, H.: Numerical Inversion of Laplace Transforms. Comm. ACM. **13**, No. 1 (1970).
20. Stehfest, H.: Comment and Correction on Algorithm 368. Comm. ACM. **13**, No. 10 (1970).
21. Szász, D.: Algoritmusok mindenkinek. Algorithms for everybody. Számológép, **2**, No. 4 (1972).
22. Valentine, C. W.: A Method for Partial-Fraction Decomposition. SIAM Rev. No. 9 (1967).
23. Zakian, V.: Numerical Inversion of Laplace Transform. Electronic Letters, **5**, No. 6, 120—121 (1969).
24. Zakian, V.: Rational Approximation to Transform Function Matrix of Distributed System. Electronic Letters, **6**, No. 15, 474—476 (1970).
25. Zakian, V.: Optimisation of Numerical Inversion of Laplace Transform. Electronic Letters, **6**, No. 21, 677—679 (1970).

Dr. Miklós Vajta Jr.
Dr. Tivadar Kovács } H-1521. Budapest