EVALUATING THE INVERSE LAPLACE TRANSFORMS FROM THE POLE-ZERO CONFIGURATION BY DIGITAL COMPUTER

By

F. CSÁKI and T. KOVÁCS

Department of Automation, Technical University Budapest

(Received November 17, 1969)

Introduction

In the analysis and design of linear control systems the inverse LAPLACE transformation is often required because in a final account the behavior of the systems within the time domain is of interest.

Theoretically the return into the time domain does not meet any difficulty and even ready formulae and tables can be found for the case of rational fractional functions.

But even then the inverse LAPLACE transformation is often tedious and wearysome. Neither does the following graphical method essentially reduce the computation work. Therefore it was practical to insert a relevant program in our program library. This program was developed only for the case of single poles and the a priori knowledge of the pole-zero configuration was supposed. The latter condition set up an additional claim for a root-finding procedure. The computer centers in general have many effective programs for this purpose.

Many other programs are also known [e.g.: 6] which apply directly the polynomial in the numerator rather than the zero configuration. The advantage of the technique presented in this paper relies on its uniformity, i.e. on the fact that both the numerator and denominator of the coefficients C_i are obtained by the same algorithm. The proposed method is very easy to apply when the pole-zero configuration is known beforehand. Our ALGOL—60 program is similar to the FORTRAN program in [3].

To support the above statements suppose for the sake of simplicity, that the control system consists of an overall element in the forward path with transfer function $G(s)$ and a feedback element with transfer function $H(s)$. Denoting the numerators by $N_g(s)$ and $N_h(s)$, and the denominators by $D_g(s)$ and $D_h(s)$, respectively, the controlled variable of the closed-loop system can be expressed as

$$C(s) = \frac{G(s)}{1+G(s)\,H(s)}\,R(s) = \frac{N_g(s)\,D_h(s)}{D_g(s)\,D_h(s)+N_g(s)\,N_h(s)}\,R(s)$$

where $R(s)$ is the LAPLACE transform of the reference input. As in control engineering the polynomials $N_g(s)$, $N_h(s)$, $D_g(s)$, $D_h(s)$ are mostly given in factorial form, and $R(s)$ is $1/s$ in our case, it follows that the numerator of $C(s)$ is given also in factorized form, thus, the zeros are directly obtained. On the contrary, the poles are to be determined since the denominator $s(D_g(s)D_h(s) + N_g(s)N_h(s))$ is not at all in factored form because of the addition encountered within the expression investigated. The factorization of the denominator or the root finding of the denominator polynomial cannot be avoided, as the poles are absolutely necessary to perform the inverse LAPLACE transformation. Similar conclusions can be drawn for multiloop control systems with a single forward path as well.

For the sake of generalization, in the following the transform of the variable to be inverted and the variable desired in the time domain will be denoted by $X(s)$, and by $x(t)$, respectively.

The graphical method and the algorithm of computation

As it is known, if $X(s)$ is a rational fractional function with distinct poles, then its inverse LAPLACE transform yields

$$x(t) = \sum_{i=1}^{k} \lim_{s \to s_i} [(s-s_i)\, X(s)\, e^{st}] = \sum_{i=1}^{k} C_i\, e^{s_i t}.$$

Supposing the pole-zero configuration to be known then the coefficients C_i of the time function can readily be got.

Instead of generalizing, let us demonstrate by an example the simplified graphical method [1, 2]:

Let

$$X(s) = \frac{s-z_1}{s(s-p_1)\,[(s+\sigma_0)^2+\omega_d^2]}\,.$$

Its inverse LAPLACE transform is

$$\mathfrak{L}^{-1}[X(s)] = x(t) = C_0 + B_1\, e^{p_1 t} + C_d\, e^{(-\sigma_0+j\omega_d)t} + \check{C}_d\, e^{(-\sigma_0-j\omega_d)t}$$

where

$$C_0 = sX(s)|_{s=0} = \frac{z_1}{p_1(\sigma_0^2+\omega_d^2)}$$

$$C_1 = (s-p_1)\, X(s)|_{s=p_1} = \frac{p-z_1}{p_1[(p_1+\sigma_0)^2+\omega_d^2]}$$

$$C_d = (s+\sigma_0-j\omega_d)\, X(s)|_{s=-\sigma_0+j\omega_d} =$$

$$= \frac{-\sigma_0+j\omega_d-z_1}{(-\sigma_0+j\omega_d)\,(-\sigma_0+j\omega_d-p_1)\,2j\omega_d}\,.$$

While

$\check{C}_d$ is the conjugate of C_d.

In each coefficient C_i of the time function every single expression is seen to be a quotient of generally complex phasor products. The numerators of these

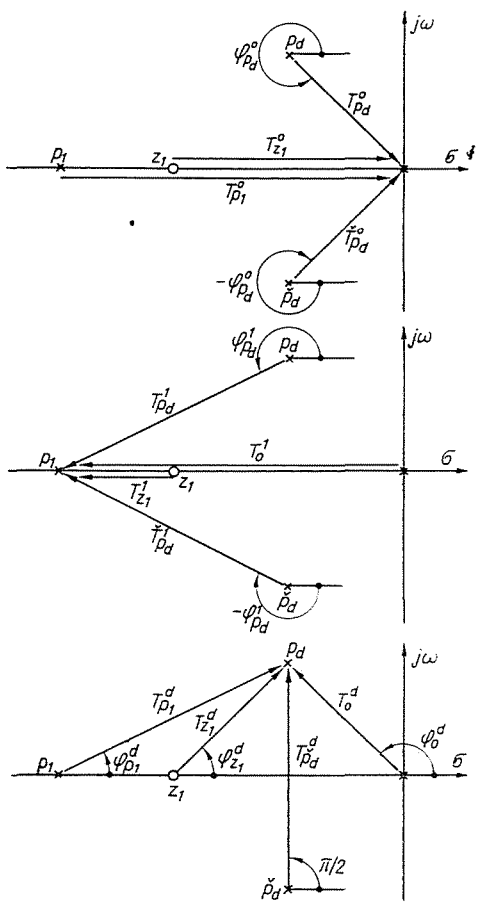

Fig. 1. Graphical method to evaluate the transient response

coefficients arised from phasors plotted from the pole in question to all zeros, while the denominators are phasors constructed to all other poles. (The same considerations lead to the algorithm of complicated cases.) Thus (see Fig. 1)

$$C_0 = \frac{T^0_{z1}}{T^0_{p_1} T_{pd} < \varphi^0_{pd} T^0_{pd} < -\varphi^0_p}$$

$$C_1 = \frac{T^1_{z_1}}{T^1_0 T_{pd} < \varphi^1_{pd} T^1_{pd} < -\varphi^1_{pd}}$$

$$C_d = \frac{T^d_{z_1} < \varphi^d_{z_1}}{T^d_0 < \varphi^d_0 T^d_{p_1} < \varphi^d_{p_1} T^d_{pd} < \varphi^d_{pd}}$$

where

$$T^d_{pd} < \varphi^d_p = 2j\omega_d.$$

If the complex terms expressed in exponential form are reduced, we have

$$x(t) = C_0 + C_1 e^{p_1 t} + 2\,|C_d|\, e^{-\sigma_0 l} \cos(\omega_d t + \varphi_d)$$

where

$$|C_d| = \frac{T^d_{z_1}}{T^d_0 T^d_{p_1} 2\omega_d}$$

and

$$\varphi_d = \varphi^d_{z_1} - \varphi^d_0 - \varphi^d_{p_1} - \pi/2\,.$$

(Remark: T denotes magnitudes, not time constants.)

The flow-chart

Since the Appendix contains the complete program, the simplified flow-chart shown in Fig. 2 will not be discussed here in detail.

The program

The program was designed for a computer RAZDAN—3 at the Computation Center of the Technical University, Budapest, Hungary. The used algorithmic language was the representation of the ALGOL—60 relative to this digital computer.

To explain and comment the published program it must be mentioned, that the statement *output (a : i : k)* has the meaning here: print the variable a on the lineprinter for i integer and for k fractional digit.

As it can also be seen from the flow-chart, for soving a problem, the number of the data groups, poles, and zeros must be given. The real and the complex part of the poles and zeros are also needed.

The *text* statements are organized so that the computed coefficients are substituted immediately in the right terms. Thus the calculated results of the time functions are arranged at the successive values of j. For example, the outputs

$j = 1 \ldots .0.15$
$j = 2 \ldots .—0.13 \times \exp(—1.0 \times t)$
$j = 3 \ldots \quad 0.11 \times \exp(—3.0 \times t) \times \cos(2.0 \times t — 4.45)$

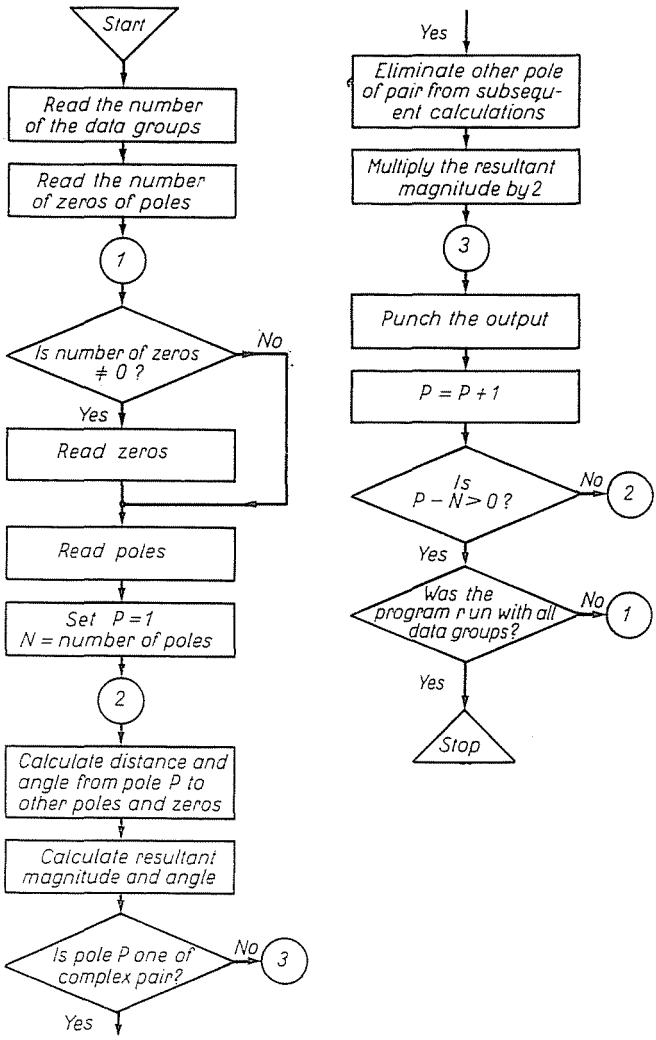

Fig. 2. The flow-chart

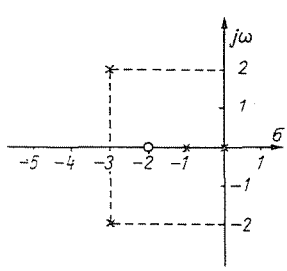

Fig. 3. The pole-zero configuration of the example

mean the time function:

$$x(t) = 0.15 - 0.13\, e{-}\mathrm{t} + 0.11\, e{-}3\mathrm{t}\cos(2t{-}4.45)$$

For an illustrative numerical example the program was runned with parameters $\sigma_0 = 3$, $\omega_d^2 = 4$, $z_1 = -2$ and $p_1 = -1$ (Fig. 3). With these data the outputs were:

$j = 1 \ldots\ldots 0.154$
$j = 2 \ldots\ldots -0.125 \times \exp(-1.000 \times t)$
$j = 3 \ldots\ldots 0.110 \times \exp(-3.000 \times t) \times \cos(2.000 \times t - 4.446)$

Appendix

The complete program for inverse LAPLACE transformation is as follows:

```
begin
real nu, dnu, alfa, frea, fim, beta, teta, frad;
integer adat, data, ir, iz, i, j, mult, jj, jip, look, n, nf, ind;
array pol, zero [1: 20, 1 : 20];
procedure pola (frea, fim, frad, beta, ind);
value frea, fim;
real frea, fim, frad, beta;
integer ind;
begin real afr, afi, eps, omeg;
integer isi;
switch s: = p1, p2, p3, p4;
ind: = 0;
afr: = abs (frea);
afi: = abs (fim);
eps: = 10 ↑ (-18);
if (afr < eps) and (afi < eps) then
begin beta: = 0; ind: = 1; goto poll end;
if (afr < eps) and (fim > eps) then
begin beta: = 1.5707963; goto poll end;
if (afr < eps) and (fim < (- eps) then
begin beta: = 1.5707963; goto poll end;
isi: = sign (frea) - sign (fim)/2 + 2.5;
goto s[isi];
p1 : p2: omeg: = 3.1415927; goto pol2;
p3 : p4: omeg: = 0.0;
pol2: beta: = artg (fim/frea) + omeg:
poll: frad: = sort (frea↑2 + fim ↑ 2);
end pola;
procedure vet (frad, beta, frea, fim);
value frad, beta;
real frad, beta, frea, fim;
begin
frea: = frad × cos (beta);
fim: = frad × sin (beta);
end vet;
integer procedure fact(n);
value n;
integer n;
begin
```

```
integer i, ik;
ik: = 1;
for i: = 1 step 1 until n do
ik: = ik×i;
fact: =
ik end fact;
begin
input (adat);
for data: = 1 step 1 until adat do
begin
input (iz,ip);
if iz = 0 then goto pr;
for i: = 1 step 1 until iz do
for j: = 1 step 1 until 2 do
input (zero[i, j]);
pr: for i: = 1 step 1 until ip do
for j: = 1 step 1 until 2 do
input (pol [i, j]);
for j: = 1 step 1 until ip do
begin
if pol (j, 2] < 0 then goto pr1;
mult: = alfa: = teta: = 0;
nu: = dnu: = 1;
for jj: = 1 step 1 until ip do
begin
jip: = jj;
if (j−jj) = 0 then goto pr2;
frea: = pol[j, 1] − pol [jj, 1];
fim: = pol[j, 2] − pol [jj, 2];
if frea neq 0 or fim neq 0 then goto pr 3;
mult: = mult + 1; goto pr 2;
pr3: pola (frea, fim, frad, beta, ind);
if ind = 1 then begin text indeterminate; line end;
dnu: = frad×dnu;
teta: = teta + beta;
pr2: end;
if (j−jip) = 0 then goto pr4;
if mult le 0 then goto pr4;
text multiplicity;
lines 2;
pr4: if iz = 0 then goto pr 5;
for jj: = 1 step 1 until iz do
begin
frea: = pol [j, 1] − zero [jj, 1];
fim: = pol [j, 2] − zero [jj, 2];
pola (frea, fim, frad, beta, ind);
if ind = 1 then begin text indeterminate; line end;
nu: = nu × frad;
alfa: = alfa + beta;
end; goto pr6;
pr5: nu: = 1; alfa: = 0;
pr6: if pol [j, 2] > 0 then goto pr7 else if
pol [j, 2] < 0 then goto pr1;
frad: = nu/dnu;
beta: = alfa − teta;
vet (frad, beta, frea, fim);
if pol [j, 1] = 0 then goto pr14;
if mult le 0 then goto pr8; look: = 1; goto pr 9;
pr 8: text j = ; output [j : 2); spaces 3;
output (frea: 5 : 3); text × exp(; output)
pol [j, 1]: 5 : 3); text×t); lines 2; goto pr1;
pr14: if mult le 0 then goto pr 10; look: = 0;
goto pr 9;
```

```
prl0: text j = ; output (j : 2); spaces 3;
output (frea: 5 : 3); lines 2; goto prl;
pr7: frad: = 2 × nu/dnu; beta: = alfa − teta;
if mult le 0 then goto prll; look: = − 1;
goto pr 9;
prll: text j = ; output (j : 2); spaces 3;
output (frad : 5 : 3); text × exp (; output (pol [j, 1] : 5 : 3);
text × t) × cos(; output(pol[j, 2] : 5 : 3); text × t +; output (beta : 3 : 3);
text); lines 2; goto pr 1;
pr9: nf: = fact (mult);
if look < 0 then goto prl2 else if look = 0
then goto prl3;
text j = ; output (j : 2); spaces 3;
text t× ×; output (mult : 2); text/; output (nf: 5);
text × exp(; output (pol[j, 1]; 5 : 3); text × t);
lines 2; goto prl;
prl3: text j = ; output (j : 2); spaces 3; text t×;
output (mult: 2); text/; output (nf: 5); lines 2;
goto prl;
prl2: text j = ; output (j : 2); spaces 3; text t×;
output (mult: 2); text/; output (nf: 5);
text × cos(; output (pol [j, 2] : 5 : 3); text × t×;
output (beta : 3 : 3); text); lines 2;
prl: end
end
end;
```

Summary

For the determination of the time behaviour of linear control systems with constant lumped parameters, inverse LAPLACE transformation is often required.

To make easier the tedious computing work for complicated cases, a complete ALGOL program is inserted for the case of known pole-zero configuration and distinct poles.

References

1. GRABBE, E. M.—RAMO, S.—WOOLDRIDGE, D. E. (editors): Handbook of Automation, Computation and Control. John Wiley et Sons, Inc. New York 1958.
2. CSÁKI, F.: Szabályozások dinamikája (Dynamics of control systems). Akadémiai Kiadó, Budapest 1966.
3. CHEN, CH.-F.—HAAS, I. J.: Elements of Control Systems Analysis. Prentice Hall, 1968.
4. MCCRACKEN, D.: A Guide to ALGOL Programming. Wiley, 1962.
5. GYÖRGYI, A.—KOVÁCS, M.—MÁRKUS, T.: RAZDAN—3 programozása, O. Ü. F. Budapest 1968.
6. ROSKA, T.: PZIL-program, Tanulmány, dokumentáció, KGM—ISZSZI számára 1968.

Prof. Dr. Frigyes CSÁKI
Tivadar KOVÁCS
} Budapest XI., Egry József u. 18. Hungary