The ALPAK System for Nonnumerical Algebra on a Digital Computer — II: Rational Functions of Several Variables and Truncated Power Series with Rational-Function Coefficients

By W. S. BROWN, J. P. HYDE and B. A. TAGUE

(Manuscript received November 13, 1963)

This is the second in a series of papers describing the ALPAK system for nonnumerical algebra on a digital computer. The first paper, Ref. 1, is concerned with polynomials in several variables and truncated power series with polynomial coefficients. This paper is concerned with rational functions of several variables and truncated power series with rational-function coefficients. A third paper, Ref. 3, will discuss systems of linear equations with rational-function coefficients.

The ALPAK system has been programmed within the BE-SYS-4 monitor system on the IBM 7090 computer, but the language and concepts are machine-independent. Several practical applications are described in Ref. 1.

This paper is divided into five sections. The first deals with basic concepts, the second defines canonical forms, and the third describes ALPAK's greatest common divisor algorithm. These three sections do not presuppose any knowledge of computers or computer programming. Section IV describes the use and the implementation of the algebraic operations relating to rational functions of several variables and truncated power series with rational-function coefficients. The reader of this section is assumed to be familiar with the basic concepts of computer programming and with Ref. 1. Finally, Section V discusses very briefly some of our plans and hopes for the future.

TABLE OF CONTENTS

I. SUMMARY OF THE AVAILABLE RATIONAL-FUNCTION OPERATIONS

1.1 *Introduction*

The ALPAK system is a programming system for performing routine manipulations of algebraic expressions on a digital computer. The system operates on rational functions of several variables and on truncated power series in several variables with rational functions of several other variables as coefficients. It is capable of performing the operations of addition, subtraction, multiplication, division, substitution and differentiation. In the present version of the program the coefficients of the rational functions are integers, but the change to coefficients from any other integral domain can be made without major program reorganization. ALPAK is also capable of solving systems of equations linear in certain variables with coefficients which are rational functions of other variables (see Ref. 3). The ALPAK system as described in this paper has been programmed for the IBM 7090 computer.

1.2 *Input-Output*

Rational functions can be entered into the machine from punched cards, and the output can be printed and/or punched. The polynomial

$$P(x,y,z) = 8xy^2z + 2xy^2z^2 - 10x^3yz^3$$

can be entered into the machine by punching the following array of coefficients and exponents one term per card:

8	1,2,1
2	1,2,2
−10	3,1,3
0	

The zero coefficient is an end-of-polynomial signal. The subroutine which reads these cards must have access to a *polynomial format statement* (previously read from cards) containing the names of the variables and the number of bits to be allocated for the exponents of each. One full word (35 bits plus sign) is allocated for the coefficient of each term, thus permitting coefficients up to $2^{35} - 1$ in magnitude. Rational functions which are not polynomials are entered by punching the numerator and denominator polynomials successively and calling the rational-function reading subroutine.

The punched and printed output consists of arrays of coefficients and exponents similar to those that are accepted as input. Input and output are accomplished by simple commands such as

RFNRDF	FMT	read format statement *FMT*
RFNRDD	R,FMT	read rational function *R*
RFNPRT	R	print rational function *R*.

1.3 *An Example of the ALPAK Language*

The simplicity of ALPAK programming is illustrated by the following example. Suppose rational functions A, B, C, and D and a format statement *FMT* have been punched on cards, and we wish to compute and print the rational function

$$F = (A*B/C) + D,$$

where the asterisk denotes multiplication.
The required program is

RFNBEG	10000	begin (reserve 10000 words of storage for data and working space)
RFNRDF	FMT	read polynomial format statement *FMT* from cards
RFNRDD	A,FMT	read polynomial A from cards
RFNRDD	B,FMT	read polynomial B from cards
RFNRDD	C,FMT	read polynomial C from cards
RFNRDD	D,FMT	read polynomial D from cards
RFNMPY	F,A,B	replace F by $A*B$
RFNDIV	F,F,C	replace F by F/C (C must not be zero)
RFNADD	F,F,D	replace F by $F + D$
RFNPRT	F	print F
TRA	ENDJOB	go to ENDJOB.

II. CANONICAL FORMS

2.1 *Introduction*

All rational functions stored by the program are kept in a unique canonical form which is the subject of this section. The read routines place the input functions in canonical form and all operations leave their results in canonical form. The uniqueness of the canonical form ensures that two equal rational functions with the same format are precisely identical in storage and that, in particular, zero is uniquely represented.

2.2 *Polynomial Canonical Form*

A polynomial is always represented as an ordered list of its nonzero terms. It is convenient to order the terms according to the magnitude of the first exponent, and to order those terms having the same first exponent according to the magnitude of the second, etc. The order of the variables is the order in which they appear in the format statement.

2.3 *Rational-Function Canonical Form*

A rational function is represented as an ordered pair of polynomials, namely its numerator and denominator respectively. These must be in polynomial canonical form, and they must be relatively prime. In addition, the sign of the numerator must be chosen so that the first term of the denominator is positive.

III. THE GREATEST COMMON DIVISOR ALGORITHM

3.1 *Introduction*

Since rational functions in canonical form must have numerator and denominator relatively prime, the ALPAK program must be capable of finding the *greatest common divisor* (G.C.D.) of polynomials in several variables. This is the essential ingredient in the extension of ALPAK from polynomials to rational functions. Since each rational-function operation must leave its result in canonical form, the G.C.D. operation will be performed very frequently in most programs involving rational functions.

Let $a_1, a_2, \cdots, a_n$ be a set of nonzero polynomials. A G.C.D. of $a_1, a_2, \cdots, a_n$ is defined to be a polynomial g such that

(i) g divides each of $a_1, a_2, \cdots, a_n$; and

(*ii*) any polynomial g' that divides each of $a_1, a_2, \cdots, a_n$ also divides g. We denote a G.C.D. of $a_1, a_2, \cdots, a_n$ by $(a_1, a_2, \cdots, a_n)$. Since every polynomial has a decomposition into primes that is unique up to sign, this definition implies that a G.C.D. is unique up to sign. In the special case of integers, the positive value is often referred to as *the* G.C.D.

The next three subsections discuss the Euclidean algorithm for integers, ALPAK's generalization of it for polynomials, and some special strategies which make the latter more effective. The final subsection attempts to present a balanced picture of the present capabilities of the ALPAK algorithm.

Algebraic background relevant to the following discussion can be found in Chapter 1 of Ref. 2, or almost any other algebra text that treats polynomial rings.

3.2 *The Euclidean Algorithm*

The G.C.D. of a set of n nonzero integers can be obtained by a series of pairwise computations, because

$$(a_1, \cdots, a_n) = (((\cdots((a_1,a_2),a_3), \cdots),a_{n-1}),a_n). \tag{1}$$

The *Euclidean algorithm* obtains the G.C.D. of two nonzero integers a and b. Without loss of generality we can assume that both are positive and that $a \geqq b$. By the division algorithm we can write

$$a = qb + c \tag{2}$$

with

$$0 \leqq c < b. \tag{3}$$

If $c = 0$, then b divides a, so $(a,b) = b$. Otherwise the common divisors of a and b are the same as those of b and c, so $(a,b) = (b,c)$. Since $b + c < a + b$, the process terminates in a finite number of steps.

3.3 *The ALPAK G.C.D. Algorithm*

We shall consider a polynomial in v variables as a polynomial in one variable, to be called x, with coefficients from the integral domain of polynomials in the remaining $v - 1$ variables. We shall represent these $v - 1$ variables by the vector y. If $p(x,y)$ is such a polynomial, then $\partial_x(p)$ denotes the degree in x of p.

Now let a and b be a pair of nonzero polynomials. We shall present an inductive definition of the ALPAK algorithm, to be called POLGCD, for obtaining their G.C.D. Let v' be the number of variables in the pair.

If $v' = 0$, then a and b are both integers and the Euclidean algorithm is used. Assume POLGCD works for $v' < v$. We shall define it for $v' = v$. To begin, we write

$$\begin{aligned} a(x,y) &= a_r(y)x^r + a_{r-1}(y)x^{r-1} + \cdots + a_0(y) \\ b(x,y) &= b_s(y)x^s + b_{s-1}(y)x^{s-1} + \cdots + b_0(y). \end{aligned} \tag{4}$$

Our first task is to rewrite this as

$$\begin{aligned} a(x,y) &= x^\alpha f(y)a'(x,y) \\ b(x,y) &= x^\beta g(y)b'(x,y) \end{aligned} \tag{5}$$

where a' and b' are *primitive in* x; that is, neither is divisible by x or by any polynomial independent of x except ± 1. Clearly, α and β are the largest integers such that x^α divides $a(x,y)$, and x^β divides $b(x,y)$; while $f(y)$ is the G.C.D. of the nonzero $a_i(y)$, and $g(y)$ is the G.C.D. of the nonzero $b_j(y)$. Since the a_i and the b_j depend on fewer than v variables, our induction hypothesis implies that POLGCD will obtain f and g. Next we observe that

$$(a,b) = (x^\alpha,x^\beta)(f,g)(a',b'). \tag{6}$$

(The proof of this depends on the fact that a' and b' are primitive.) It is obvious that

$$(x^\alpha,x^\beta) = x^\gamma \tag{7}$$

where γ is the smaller of α and β. Since f and g depend on fewer than v variables, our induction hypothesis implies that we can use POLGCD to obtain (f,g).

We shall now define a subalgorithm, to be called PRMGCD, for obtaining the G.C.D. of the primitive polynomials $a'(x,y)$ and $b'(x,y)$. To begin, we write

$$\begin{aligned} a'(x,y) &= a_m'(y)x^m + a_{m-1}'(y)x^{m-1} + \cdots + a_0'(y) \\ b'(x,y) &= b_n'(y)x^n + b_{n-1}'(y)x^{n-1} + \cdots + b_0'(y), \end{aligned} \tag{8}$$

where a_m',a_0',b_n',b_0' are all nonzero. Without loss of generality we can assume that $\partial_x(a') \geqq \partial_x(b')$. If $\partial_x(b') = 0$, then $b' = b_0' = \pm 1$,* so $(a',b') = 1$. Otherwise, we use POLGCD to compute

$$h = (a_m',b_n'). \tag{9}$$

Then we form

$$c(x,y) = \left[\frac{b_n'(y)}{h(y)}\right] a'(x,y) - \left[\frac{a_m'(y)}{h(y)}\right] b'(x,y)x^{m-n}, \tag{10}$$

* Here we have used the definition of primitivity which is given following (5).

in which the bracketed fractions are polynomials. If $c = 0$, then $a' = \pm b'$;* so $(a',b') = b'$. Otherwise, let $c'(x,y)$ be the primitive part of $c(x,y)$, defined as in (6). Then the common divisors of a' and b' are the same as those of b' and c',* so $(a',b') = (b',c')$. (Note the similarity of this algorithm to the Euclidean algorithm, which was defined in the preceding subsection.) By construction

$$\partial_x(c') \leqq \partial_x(c) < \partial_x(a'). \tag{11}$$

Hence

$$\partial_x(b') + \partial_x(c') < \partial_x(a') + \partial_x(b'), \tag{12}$$

and so the process terminates in a finite number of steps.

3.4 *Special Strategies*

From a practical point of view the POLGCD algorithm leaves much to be desired. As the degree in x is reduced, the coefficients grow. This phenomenon is vividly illustrated by the following example. It is desired to find the G.C.D. of the primitive polynomials

$$\begin{cases} -39x^4 + 125x^3 - 15x^2 - 135x - 44 \\ -12x^4 - 89x^3 + 192x^2 - 6x - 85. \end{cases} \tag{13}$$

The successive pairs of primitive polynomials produced by the PRMGCD algorithm are:

$$\begin{cases} -12x^4 - 89x^3 + 192x^2 - 6x - 85 \\ -1657x^3 + 2556x^2 + 462x - 929 \end{cases}$$

$$\begin{cases} 178145x^3 - 312600x^2 - 1206x + 140845 \\ -1657x^3 + 2556x^2 + 462x - 929 \end{cases}$$

$$\begin{cases} -1657x^3 + 2556x^2 + 462x - 929 \\ 5219965x^2 - 6692054x - 5656955 \end{cases} \tag{14}$$

$$\begin{cases} 2253497062x^3 - 6961950605x - 4849347485 \\ 5219965x^2 - 6692054x - 5656955 \end{cases}$$

.
.
.

To proceed farther would require double-precision coefficients, which are not now available in ALPAK. For polynomials in many variables this problem is even more acute. If POLGCD were programmed to handle coefficients and exponents of arbitrary size, the time it would

* Here we have used the definition of primitivity which is given following (5).

require to find the G.C.D. of two *general* polynomials in n variables, each having degrees $d_1, \cdots, d_n$ respectively and each having coefficients of modest size, can be shown to be proportional to

$$\begin{matrix} & & & & (c^{d_1}) \\ & & & \cdot & \\ & & \cdot & & \\ & (c^{d_{n-1}}) & & & \\ (c^{d_n}) & & & & \end{matrix} \tag{15}$$

where $c \approx 2.6$ is the square of the Fibonacci ratio, $\frac{1}{2}(1 + \sqrt{5})$.

It is apparent that any real solution to the problem of coefficient growth would require a fundamentally different algorithm. However, many of the G.C.D. problems which arise in practice exhibit special properties which can be exploited.

The most important of these special properties is variable independence. If one of the inputs to POLGCD is independent of one or more of the variables, then the other can immediately be broken into subpolynomials, and we obtain a set of subproblems each involving only those variables which both of the original inputs depend on. For example, it is clear by inspection that the G.C.D. of the pair

$$\begin{aligned} &2z(2x^4 - 17x^3 + 65x^2 - 144x + 72) \\ &\quad - 3y^2(2x^3 - 17x^2 + 66x - 72) \\ &6x^4 - 41x^3 + 104x^2 - 116x + 48 \end{aligned}$$

is equal to the G.C.D. of the triple

$$\begin{aligned} &2x^4 - 17x^3 + 65x^2 - 144x + 72 \\ &2x^3 - 17x^2 + 66x - 72 \\ &6x^4 - 41x^3 + 104x^2 - 116x + 48, \end{aligned}$$

and the POLGCD algorithm will discover this provided that x is the last variable in the format statement. Unfortunately POLGCD does not now optimize the order of the variables. If the preceding example is attempted using the variable ordering (x,y,z), disaster ensues. The inputs are viewed as

$$\begin{aligned} &4zx^4 - (6y^2 + 34z)x^3 + (51y^2 + 130z)x^2 \\ &\quad - (198y^2 + 288z)x + (216y^2 + 144z) \\ &6x^4 - 41x^3 + 104x^2 - 116x + 48, \end{aligned}$$

and it is easily seen that both are primitive in x. The next two pairs of primitive polynomials produced by PRMGCD are

$$\begin{cases} 6x^4 - 41x^3 + 104x^2 - 116x + 48 \\ (18y^2 + 20z)x^3 - (153y^2 + 182z)x^2 \\ \qquad + (59y^2 + 452z)x - (648y^2 + 336z) \end{cases}$$

$$\begin{cases} (45y^2 + 68z)x^3 - (423y^2 + 158z)x^2 \\ \qquad + (450y^2 - 76z)x + (216y^2 + 240z) \\ (18y^2 + 20z)x^3 - (153y^2 + 182z)x^2 \\ \qquad + (59y^2 + 452z)x - (648y^2 + 336z). \end{cases}$$

$$\vdots$$

The variable independence has now been lost, and the subsequent pairs will have progressively higher degrees in y and z, and progressively larger coefficients.

Both POLGCD and PRMGCD test their inputs to see whether either divides the other. Since a given G.C.D. problem may involve many recursive calls to POLGCD and PRMGCD, this strategy pays frequent dividends. POLGCD and PRMGCD also make full use of the fact that the G.C.D. of the set of terms of a polynomial, and similarly the G.C.D. of a monomial and a polynomial, can be computed simply and directly.

Finally, we remark that the PRMGCD process is terminated as soon as the degree in x of either input is zero or one. A primitive polynomial of degree zero is obviously equal to ± 1, while a primitive polynomial of degree one is irreducible. At the last variable level the PRMGCD process is terminated as soon as the degree in x of either input is three or less. A quadratic polynomial can be factored, if it is reducible, with the aid of the quadratic formula. A reducible cubic must have at least one rational root. A simple change of variable produces a related cubic which must have at least one integral root, and it is easy to test for this numerically.

3.5 *Concluding Remarks*

As we have already stated, the G.C.D. operation is the essential ingredient in the extension of ALPAK from polynomials to rational functions. The weakness of the ALPAK G.C.D. algorithm is apparent from (15). Its strength lies in the fact that most G.C.D. computations which arise in problems of practical interest have a degree of immunity from that formula because of their special structure.

As an example we wish to mention the problem of a single-server

queue with feedback. The computation of the first two moments of the total time is outlined in Section II of Ref. 1 and in the Appendix of Ref. 5. We recently obtained the third moment in a six-minute run on the 7090. This involved solving a triangular linear system of nine equations in nine unknowns. The equations, expressed as polynomials in the nine unknowns and five additional parameters, have over 900 terms. The result is a rational function of the five parameters with a numerator of 200 terms and a denominator of 39 terms. The coefficients of largest magnitude are 1896 in the numerator and 1460 in the denominator. The degrees are 1, 1, 3, 7, and 9 for the numerator and 0, 0, 0, 7, and 9 for the denominator.

IV. INFORMATION FOR THE ALPAK PROGRAMMER

4.1 *Introduction*

This section is an extension of Section III of Ref. 1. The use and implementation of the ALPAK polynomial operations are described there, while the use and implementation of the rational-function operations are described here. The loading instructions are unchanged except that an additional binary deck, called ALPAK3, must be included for a run.

A nonpolynomial rational function is stored as an ordered pair of polynomials, namely its numerator and denominator, as illustrated in Fig. 1. It consists of a pointer, a rational-function heading, and two polynomials stored in the usual way (see Fig. 2 of Ref. 1). The rational-function heading contains pointers to the polynomials, which must have a common format.

Integers and polynomials are always recognized as special cases of rational functions. If a rational-function operation is used where a polynomial operation might have been used, the only penalty will be a fraction of a millisecond of additional overhead.

A rational function can be constructed from its numerator and denominator polynomials by using RFNDIV (divide) or RFNFRM (form). RFNDIV duplicates the two polynomials and constructs the rational function from the copies. RFNFRM constructs the rational function from the given polynomials and clears their pointers. RFNFRM has an optional argument which can be used to indicate that the numerator and denominator are known to be relatively prime.

4.2 *Input-Output Operations*

	RFNRDF	F	read format	(a)
F	RFNCVF	(X,15,Y,21,Z,36)	convert format	(b)

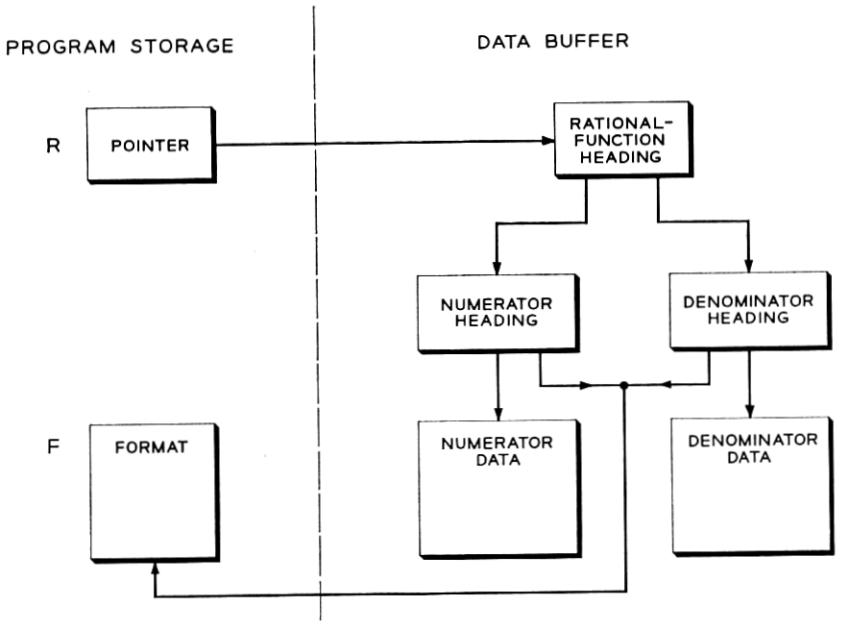

Fig. 1 — A rational function R with format F.

RFNRDD	R,F	read data	(c)*
RFNCVD	R,F,HN,HD	convert data	(d)
RFNCLR	R	clear	(e)
RFNSTZ	R	store zero	(f)
RFNSTI	R	store identity	(g)
RFNSTC	R,A,B	store constant	(h)
RFNSTV	R,X,F	store variable	(i)
RFNPRT	R,CC,(NAME)	print	(j)
RFNPCH	R,(NAME)	punch	(k)*
RFNPRP	R,CC,(NAME)	print and punch	(l)*
RFNRDP	R,F,CC,(NAME)	read and print	(m)
RFNCVP	R,F,HN,HD,CC,(NAME)	convert and print	(n)

A = numerator of constant

B = denominator of constant; if omitted, the denominator is understood to be one

* RFNRDD reads *two* polynomials from cards, interpreting the first as the numerator and the second as the denominator of a rational function. If a polynomial is to be read by RFNRDD, a unit denominator with a complete set of zero exponents must be provided. A rational function with a constant numerator (denominator) punched by RFNPCH or RFNPRP cannot be read by RFNRDD unless a complete set of zero exponents is added to the numerator (denominator).

CC = control character for printer
F = format (symbolic address of/for format statement)
HN = Hollerith data for numerator (symbolic address of data)
HD = Hollerith data for denominator (symbolic address of data); if omitted, the denominator is understood to be one
NAME = alternative name for rational function (not exceeding 21 characters)
R = rational function (symbolic address of pointer)
X = variable (specified in the manner indicated by the last previous VARTYP declaration. See Ref. 1, Section 3.5).

(a) RFNRDF F

Same as POLRDF. See Ref. 1, Section 3.2.

(b) F RFNCVF (X,15,Y,21,Z,36)

Same as POLCVF. See Ref. 1, Section 3.2.

(c) RFNRDD R,F

Read the rational function R from cards according to format F. R is the address of a pointer for the rational function and F is the address of a format statement. R must consist of a polynomial numerator and polynomial denominator punched in cards in that order as specified by Ref. 1, Section 3.2.

(d) RFNCVD R,F,HN,HD

Same as RFNRDD except that the numerator and denominator polynomials are to be found in core in blocks of no more than 12 BCI words each starting at HN and HD, respectively.

(e) RFNCLR R

Clear the rational function R. This clears both numerator and denominator polynomials as well as the R heading if R is not itself a polynomial. If the R pointer contains zero or points to an idle heading, then RFNCLR is a no-op.

(f) RFNSTZ R

Same as POLSTZ. See Ref. 1, Section 3.2.

(g) RFNSTI R

Same as POLSTI. See Ref. 1, Section 3.2.

(h) RFNSTC R,CN,CD

Same as POLSTC if CD is omitted. See Ref. 1, Section 3.2. If CD is present, then CN and CD are the addresses of constants which become the numerator and denominator, respectively, of the rational constant R = CN/CD.

(i) RFNSTV R,X,F

Same as POLSTV. See Ref. 1, Section 3.2.

(j) RFNPRT R,CC,(NAME)

Print the rational function R using CC for the control character of the first line of print and NAME (not more than 21 characters) for the name. If NAME is not provided, "R" will be used for the name; and, if CC is not provided, a minus (for triple spacing) is used for the control character. A rational function not a polynomial is printed by printing the name on the first line, followed by two polynomial prints with the names "NUMERATOR" and "DENOMINATOR" respectively

(k) RFNPCH R,(NAME)

Punch the rational function R on cards using NAME (no more than 21 characters) for the name. If NAME is not provided "R" will be used for the name. A rational function not a polynomial will be punched as two polynomials, numerator and denominator in that order.

(l) RFNPRP R,CC,(NAME)

Same as RFNPRT followed by RFNPCH.

(m) RFNRDP R,F,CC,(NAME)

Same as RFNRDD followed by RFNPRT.

(n) RFNCVP R,F,HN,HD,CC,(NAME)

Same as RFNCVD followed by RFNPRT.

4.3 *Arithmetic Operations*

RFNADD	R,P,Q	$R = P + Q$	add	(a)
RFNSUB	R,P,Q	$R = P - Q$	subtract	(b)
RFNMPY	R,P,Q	$R = P*Q$	multiply	(c)
RFNDIV	R,P,Q	$R = P/Q$	divide	(d)*
RFNSST	G,F(LISTR) (LISTV)	$G = F$(LISTV = LISTR)	substitute	(e)

* RFNDIV can form the quotient of any two rational functions provided the divisor is not zero. In contrast, POLDIV has a "no divide" return which is used whenever the quotient is not a polynomial.

RFNDIF	Q,P,X	$Q = \partial P/\partial X$	differentiate	(f)
RFNZET	R	Skip *iff* $R = 0$	zero test	(g)
RFNNZT	R	Skip *iff* $R \neq 0$	nonzero test	(h)
RFNEQT	P,Q	Skip *iff* $P = Q$	equality test	(i)
RFNDUP	Q,P	$Q = P$	duplicate	(j)
RFNCHS	R	$R = -R$	change sign	(k)

F,G,P,Q,R = rational functions (symbolic addresses of pointers)
X = variable (specified in the manner indicated by the last previous VARTYP declaration) †
LISTR = list of rational functions.
LISTV = list of variables (specified in the manner indicated by the last previous VARTYP declaration) †

4.3.1 *Notation*

In the following descriptions, if R denotes a rational function, we denote its numerator by RN and its denominator by RD. In particular, if R is a polynomial, RD is the constant polynomial 1.

4.3.2 *Descriptions*

(a) RFNADD R,P,Q

The inputs P and Q are rational functions in canonical form. First POLGCD is used to obtain

$$G = (PD,QD).$$

Then the polynomials

$$AN = PN*(QD/G) + QN*(PD/G)$$

and

$$AD = (PD/G)*(QD/G)*G$$

are computed. Note that the parenthesized fractions are polynomials. Next,

$$H = (AN,G)$$

is obtained and

$$R = \frac{(AN/H)}{(AD/H)}$$

† See Ref. 1, Section 3.5.

is formed. *RN* and *RD* are now relatively prime (the proof of this is an exercise for the reader), so POLGCD need not be used when placing *R* in canonical form.

(b) RFNSUB R,P,Q

RFNCHS [see (k) below] and RFNADD are applied to compute

$$R = P + (-Q).$$

(c) RFNMPY R,P,Q

First the functions

$$A = PN/QD$$

and

$$B = QN/PD$$

are formed and placed in canonical form. Then

$$R = AN*BN/AD*BD$$

is formed. *RN* and *RD* are already relatively prime, so POLGCD need not be used when placing R in canonical form.

(d) RFNDIV R,P,Q

RFNDIV is identical to RFNMPY except that *Q* must be not zero and the roles of *QN* and *QD* are interchanged. If *Q* is zero, the diagnostic remark "ZERO DENOMINATOR" is printed and the job is terminated.

(e) RFNSST G,F(LISTR) (LISTV)

RFNSST is exactly the rational function equivalent of POLSST; in particular, the format constraints on *F* and *G* are identical. If *F* is a polynomial, the rational functions of LISTR are substituted for the variables in LISTV term-by-term to accumulate the final result. If *F* is a rational function, this procedure is applied to the numerator and denominator polynomials of *F* in succession and the resulting rational functions are divided (using RFNDIV) to obtain *G*.

(f) RFNDIF Q,P,X

First POLDIF is used to compute

$$PN' = \partial(PN)/\partial X$$

$$PD' = \partial(PD)/\partial X,$$

and POLGCD is used to obtain

$$G = (PD,PD').$$

Next the polynomials

$$AN = (PD/G)*PN' - PN*(PD'/G)$$

and

$$AD = PD*(PD/G) = (PD/G)^2*G$$

are computed. Note that the parenthesized fractions are polynomials. Finally

$$H = (AN,G)$$

is obtained, and

$$Q = \frac{(AN/H)}{(AD/H)}$$

is formed. Since QN and QD are relatively prime, POLGCD need not be used when placing Q in canonical form.

(g) RFNZET R

Same as POLZET. See Ref. 1, Section 3.3

(h) RFNNZT R

Same as POLNZT. See Ref. 1, Section 3.3.

(i) RFNEQT P,Q

If P and Q are both polynomials, POLEQT is applied. If only one of them is a polynomial, they are not equal. If neither is a polynomial, POLEQT is applied to both numerators and both denominators. If the rational functions are unequal, the next instruction is executed, if they are equal, then the next instruction is skipped.

(j) RFNDUP Q,P

Q is replaced by a copy of P.

(k) RFNCHS R

If R is a polynomial, POLCHS is applied to R. If R is not a polynomial, POLCHS is applied to its numerator polynomial.

4.4 *Truncated Power Series Operations*

ALPAK contains two macros for dealing with truncated power series with rational function coefficients. These are RFNTRC (truncate) and

RFNMPT (multiply and truncate). Addition can be handled with RFNTRC and RFNADD. Each truncated power series must be stored as a rational function in a format whose first k variables are the power series variables. The denominator, if any, must be independent of these variables. The command

RFNTRC P,ORD,K

where K contains the number of power series variables k and ORD contains an integer n, causes P to be truncated to order n. That is, all terms of order greater than n in the first k variables are deleted. The command

RFNMPT R,ORDR,P,ORDP,Q,ORDQ,K

is represented by the equation

$$R = P*Q$$

where P and Q are truncated power series. K is the address of the number of power series variables, ORDP and ORDQ are the addresses of the orders of P and Q respectively, and ORDR is an address for the order of R, which is to be comupted. If $P(Q)$ contains any terms of order greater than ORDP(ORDQ), they will be deleted.

4.5 *Miscellaneous Operations*

(Caution: read descriptions carefully.)

POLGCD	G,A,B	greatest common divisor of polynomials	(a)
INTGCD		greatest common divisor of integers in AC and MQ	(b)
PWVSTO	XK,K,W,FA	store a power of the Wth variable	(c)
VARNUM	W,X,FA	variable number	(d)
RFNFRM	R,N,D	form	(e)
EXPAND	N,D,R,IORP	expand	(f)
SUBLCK	PJ,P,J,W	sub-block	(g)
PWVFAC	K,P,W	factor off a power of the Wth variable	(h)
DEGREE	K,P,W	degree.	(i)

(a) POLGCD G,A,B

Replace G by a greatest common divisor of the polynomials A and B.

(b) INTGCD

Replace the integer in the AC by the greatest common divisor of it and the integer in the MQ.

(c) PWVSTO XK,K,W,FA

Replace XK by the Kth power of the Wth variable in the format whose address is at FA. K is the address of the power, and W is the address of the variable number.

(d) VARNUM W,X,FA

Replace the contents of W by the variable number of the variable X in the format whose address is at FA. X is the address of the variable name in BCI.

(e) RFNFRM R,N,D

Same as RFNDIV except: (*i*) N and D must be polynomials; (*ii*) N and D become the property of R, and their pointers in the calling program are replaced by zeros; (*iii*) R must be distinct from N and D; and (*iv*), if N and D are known to be relatively prime, a fourth argument NOGCD can be added to the calling sequence in order to save the time which would otherwise be spent in finding their greatest common divisor.

(f) EXPAND N,D,R,IORP

This is the inverse of RFNFRM. N and D must initially contain zeros. They are filled in with pointers to the numerator and denominator of R respectively. The R heading is marked as idle. If R is an *integer or a polynomial*, N is filled in with a pointer to R, the R pointer is replaced by zero, and control is transferred to IORP. If IORP is omitted, control is transferred to the next instruction.

(g) SUBLCK PJ,P,J,W

P must be a polynomial independent of the first $W - 1$ variables, if any. Then, by the definition of the polynomial canonical form, the terms of P are ordered according to the powers of the Wth variable. SUBLCK replaces PJ by the polynomial consisting of that sub-block of P, if any, whose terms all involve the Jth power ($0 < J <$ degree of P) of the Wth variable. If P contains no terms involving the Jth power of the Wth variable, SUBLCK replaces PJ by the zero polynomial.

(h) PWVFAC K,P,W

P must be a polynomial independent of the first $W - 1$ variables, if

any. PWVFAC replaces the contents of K by the smallest exponent of the Wth variable in P, and divides P by that power of the Wth variable.

(i) DEGREE K,P,W

P must be a polynomial independent of the first $W - 1$ variables, if any. DEGREE replaces the contents of K by the degree of P in the Wth variable.

V. OUTLOOK

A new version of ALPAK (to be called ALPAKB) is now being developed. Its foundation is a programming system (see Ref. 4) called STGPAK (*storage package*), which provides (*i*) dynamic storage allocation, (*ii*) automatic recursion, and (*iii*) "delayed-decision diagnostics."

The storage allocation orders make it possible to obtain contiguous blocks of storage of arbitrary length as needed (provided that sufficient space is available) and to return idle space to the system. A block may contain sub-blocks and/or pointers to other blocks. This will permit the introduction of higher level data structures including formal products of polynomials, thereby helping to alleviate the greatest-common-divisor problem.

The use of a public push-down list for subroutine storage makes recursive programming fully automatic. That is, a subroutine can call itself without taking special measures to preserve its arguments and intermediate results. The diagnostic facilities permit the decision regarding what to do about an overflow (shortage of space or time) or error detected in a given subroutine, to be delayed until control has been returned to some higher level subroutine or to the main program.

The authors hope that STGPAK together with a macro compiler now being developed by Miss D. C. Leagus and W. S. Brown will simplify and expedite the programming of ALPAKB subroutines. The compiler should also be useful in the writing of main programs.

Apart from these matters, which are not directly related to algebra, our plans for ALPAKB include multiple precision integer arithmetic, an improved strategy for finding greatest common divisors, and a complete set of operations for truncated power series.

REFERENCES

1. Brown, W. S., The ALPAK System for Nonnumerical Algebra on a Digital Computer — I: Polynomials in Several Variables and Truncated Power Series with Polynomial Coefficients, B.S.T.J., **42**, September, 1963, p. 2081.

2. Zariski, O., and Samuel, P., *Commutative Algebra*, Vol. I, D. Van Nostrand, Princeton, 1958. (See especially Chapter 1.)
3. Hyde, J. P., The ALPAK System for Nonnumerical Algebra on a Digital Computer — III: Systems of Linear Equations and a Class of Side Relations, to be published.
4. Brown, W. S., and Leagus, D. C., STGPAK: A Programming System for Dynamic Storage Allocation, Automatic Recursion, and "Delayed-Decision Diagnostics," to be published.
5. Takacs, L., A Single-Server Queue with Feedback, B.S.T.J., **42**, March, 1963, p. 505. (See especially the Appendix by W. S. Brown.)