On the generation of the pseudo-remainder in polynomial division

Y. L. Varol*

Department of Mathematics, Ben Gurion University of the Negev, PO Box 2053, Beersheva 84 120, Israel

All the known methods for finding the GCD-greatest common divisor—of two polynomials are based on some variation of Euclid's algorithm, which uses repeated division of the successive divisors by the remainders. Two new algorithms for generating the remainder resulting from the pseudo-division of polynomials over a commutative ring are suggested. They are compared with respect to memory and CPU requirements. It is found that the new algorithms are more efficient, in particular, when the difference between the degrees of the polynomials is large, and/or the commutative ring is not that of the integers.

(Received July 1975)

1. Introduction

Symbolic manipulation of polynomials is a field which has received considerable attention from computer scientists, and resulted in the development of various software packages such as ALPAC (Brown, Hyde anu Tague, 1963), PM (Collins, 1966), and SAC (Collins, 1971). Finding the greatest common divisor of two polynomials, is of particular interest in this field (Brown, 1971), (Brown and Traub, 1971), and (Collins, 1967). This in turn is based on some variation of Euclid's algorithm, which uses repeated division of the successive divisors by the remainders. The efficiency of the whole process is very much dependent on the generation of the remainder, which is the subject of this paper.

Given two polynomials $A = \sum_{i=0}^{m} a_i x^{m-i}$, and $B = \sum_{i=0}^{u} b_i x^{n-i}$ over any commutative ring, we wish to obtain the pseudo-remainder R resulting from the division of $(b_0)^{m-n+1}$ A by B. Multiplying A by $(b_0)^{m-n+1}$ enables one to carry out the division within the commutative ring. Knuth (1969, p. 369) gives the following classical algorithm to find the remainder:

$S1$: Do step $S2$ for $k = m - n, m - n - 1, \ldots, 0$; then the algorithm terminates with $a_{m-n+1}, a_{m-n+2}, \ldots, a_m$ as the coefficients of the remainder.

$S2$: For $j = n + k - 1, n + k - 2, \ldots, 0$ set
$a_{m-j} \leftarrow b_0 a_{m-j} - a_{m-n-k}\, b_{n+k-j}$ for $j \geq k$, and set
$a_{m-j} \leftarrow b_0 a_{m-j}$ for $j < k$.

Note that the order of the coefficients has been reversed to simplify the notation further on. This algorithm is quite efficient timewise, and has minimal memory requirements when one is working over the commutative ring of the integers. However, serious degradation takes place in both respects for the case of multivariate polynomials, or any other commutative ring. In what follows, two alternative algorithms are suggested and then, all three are compared with respect to storage and time requirements.

2. Generation of the remainder

For any fixed i, $0 \leq i \leq m - n$, let C_i be the set of all sequences $\{\lambda_k\}_{k=1}^{c}$, of positive integers such that,

$$\begin{aligned} &1 \leq c \leq m + 1 - n - i \\ &1 \leq \lambda_k \leq n, \text{ for every } k \\ &\sum_{k=1}^{c-1} \lambda_k \leq m - n - i \\ &\sum_{k=1}^{c} \lambda_k > m - n - i\,. \end{aligned} \qquad (1)$$

An element $\{\lambda_k\}_{k=1}^{c}$ of C_i corresponds to the remainder term resulting from the following sequence of operations in the division process: The term $a_i b_0^{m-n+1} x^{m-i}$ is divided by $b_0 x^n$; the quotient, multiplied by $b_{\lambda_1} x^{n-\lambda_1}$ results in a term $-a_i b_0^{m-n} b_{\lambda_1} x^{m-i-\lambda_1}$ which is divided by the leading term in B; the quotient of this operation, multiplied by $b\lambda_2 x^{n-\lambda_2}$ results in $a_i b_0^{m-n-1} b_{\lambda_1} b_{\lambda_2} x^{m-i-\lambda_1-\lambda_2}$ which is again divisable by $b_0 x^n$; . . .; this process is stopped the first time $\sum_{k=1}^{c} \lambda_k > m - n - i$, with a remainder term of $(-1)^c a_i b_0^{m+1-n-c} b_{\lambda_1} b_{\lambda_2} \ldots b_{\lambda c} x^{m-i-\sum_{k=1}^{c} \lambda_k}$.

The inequalities in Equation 1 should now be apparent, since λ_k's are the subscripts of the coefficients in B, and if $\sum_{k=1}^{c-1} \lambda_k > m - n - i$, then the last division by $b_0 x^n$ could not have been performed.

Clearly, there is a one-to-one correspondence between elements of C_i and the terms of the remainder resulting from the division of $a_i b_0^{m-n+1} x^{m-i}$ by the polynomial B. This remainder can be written as:

$$R_i = a_i \left(\sum_{c_i} (-1)^c b_0^{m+1-n-c} b_{\lambda_1} b_{\lambda_2} \ldots b_{\lambda_c} x^{m-i-\sum_{k=1}^{c} \lambda_k} \right). \qquad (2)$$

Accordingly, the pseudo-remainder in the division of $b_0^{m+1-n} A$ by B is:

$$R = b_0^{m+1-n} \left(\sum_{i=m+1-n}^{m} a_i x^{m-i} \right) + \sum_{i=0}^{m-n} R_i\,. \qquad (3)$$

To obtain all the coefficients of x^{n-q}, in R, we shall consider a combinatorial method for deriving the set of all the sequences $\{\lambda_k\}$ corresponding to them. This set, denoted by Q_{n-q}, will be formulated as a union of additive decompositions of integers which are easy to generate, and well known in number theory. The new algorithms will use this formulation of Q_{n-q} to compute the cumulative coefficients of the remainder.

We first observe that, combining the definition of Q_{n-q} with the exponent in Equation 2, we get

$$m - i - \sum_{k=1}^{c} \lambda_k = n - q\,.$$

This implies that every sequence in Q_{n-q} must have the last element, λ_c, greater than or equal to q. Otherwise, $\sum_{k=1}^{c-1} \lambda_k > m - n - i$, which contradicts Equation 1. Sequences in Q_{n-q}

*Now at: University of the Witwatersrand, Johannesburg.

must therefore satisfy the following conditions:

$$\begin{aligned}
&1 \le \lambda_k \le n, \text{ for every } k \ , \\
&1 \le c \le m + 1 - n \ , \\
&q \le \lambda_c \le n \ , \qquad\qquad\qquad\qquad (4) \\
&0 \le i \le m - n \ , \\
&m - 2n - i + q \le \sum_{k=1}^{c-1} \lambda_k \le m - n - i \ .
\end{aligned}$$

Using straightforward induction, one can now prove that,

$$Q_{n-q} = U_{i=0}^{m-n} \{U_{j=0}^{n-q} \{P_{m-n-i-j} \times \{q+j\}\}\} \ , \quad (5)$$

where Pl is the set of all additive decompositions of the positive integer l from integers less or equal to n, $q + j$ stands for λ_c, and $\times$ is the usual cross product of sets. Note that the union is defined for $m - n - i - j \ge 0$, with P_0 being the identity with respect to the cross product, i.e. $P_0 \times S = S$ for any set S. One can also show by induction that,

$$P_l = U_{\substack{\mu=1 \\ l-\mu\geqslant 0}}^{n} \{P_{l-\mu} \times \{\mu\}\} \ . \quad (6)$$

Using equation (6) and interchanging finite unions of sets,

$$\begin{aligned}
U_{j=0}^{n-q}\{P_{m-n-i-j}\times\{q+j\}\} &= U_{j=0}^{n-q}\{\{U_{\mu=1}^{n}\{P_{m-n-i-j-\mu}\times\{\mu\}\}\} \\
&\quad \times\{q+j\}\} \\
&= U_{\mu=1}^{n}\{\{U_{j=0}^{n-q}\{P_{m-n-i-j-\mu} \\
&\quad \times\{q+j\}\}\}\times\{\mu\}\} \ . \quad (7)
\end{aligned}$$

A simple comparison between Q_{n-q} and $Q_{n-(q+\mu)}$ leads to our final observation. From equation (5), we see that for any i, $P_{m-n-j-i} \times \{q+j\}$ which appears in Q_{n-q}, is also part of $Q_{n-(q+\mu)}$ for $i - \mu$, and $0 \le \mu \le \min(i, n - q - 1)$. Therefore, to generate all the Q_{n-q}'s, one could initially derive all sets of the form $P_l \times \{k\}$ for $1 \le l \le m + 1 - n$, $1 \le k \le n$, and then consider proper subsets of them according to equation (5) (Algorithm U). A second approach could be to generate each Q_{n-q} separately using equation (5) in conjunction with equation (6) and (7) (Algorithm T). This approach would be more efficient if one is interested in only some of the coefficients in the remainder and not all of them.

3. Algorithms

The two algorithms presented below generate directly the product $\prod (b_{\lambda_k})$ rather than the sequence of subscripts $\{\lambda_k\}$. Accordingly, the letter Q is now used to stand for the sum of such products.

Algorithm T

This computes the coefficient r_{n-q} of x^{n-q} in the remainder. Repeating steps $T1$ to $T3$ for $q = 1, 2, \ldots, n$ would produce the whole remainder.

```
T1: Set $Q_1 \leftarrow b_0^{m+1-n}$ and
    $Q_j \leftarrow -(Q_1/b_0)b_{q+j-2}$, for $2 \le j \le n - q + 2$, and
    $Q_j \leftarrow 0$                    , for $n-q+3 \le j \le m-n+2$.
T2: For $l = 3, 4, \ldots, m - n + 2$ set $k \leftarrow \min(n, l - 2)$ and
    $Q_l \leftarrow Q_l - \sum_{j=1}^{k} (Q_{l-j}/b_0)b_j$ .
T3: Set $r_{n-q} \leftarrow Q_1 a_{m-n+q} + \sum_{l=2}^{m-n+2} Q_l a_{m-n+2-l}$ .
```

Algorithm U

The sum of products $\prod (b_\lambda)$ which correspond to sequences in $P_l \times \{k\}$ for various l and k are initially computed, and then combined to produce the coefficients.

```
U1: Set $P_1 \leftarrow b_0^{m-n}$
U2: For $i = 2, 3, \ldots, m + 1 - n$, set $k \leftarrow \min(i - 1, n)$ and
    $P_i \leftarrow -\sum_{j=1}^{k} (P_{i-j}/b_0)b_j$
U3: For $i = 1, 2, \ldots, m + 1 - n$, and $j = 1, 2, \ldots, n$ set
    $K_{i,j} \leftarrow P_i b_j$, and
    $l \leftarrow \min(n + 1 - j, i)$
    $Q_{i,j} \leftarrow \sum_{\mu=1}^{l} K_{i+1-\mu, j+\mu-1}$
U4: For $q = 1, 2, \ldots, n$ set
    $r_{n-q} \leftarrow P_1 b_0 a_{m-n+q} - \sum_{i=1}^{m+1-n} Q_{i,q} a_{m-n+1-i}$ .
```

4. Discussion

Both algorithms T and U are given in their form implied by the formulas derived earlier. The sign $(-1)^c$ of equation (2) is obtained by introducing a minus sign with each multiplication by b_λ. Note that the leading coefficient, b_0 of B, is initially raised to some power, and later stages all involve removing these powers by successive divisions. This may look untidy, and the loops can indeed be organised in reverse order to eliminate what seems to be a duplication. This can only be accomplished at the expense of loosing the recursive nature in the computations of Q_l and P_l. and leads to inefficiency. In algorithm U step $U3$ introduces two matrices which increase storage requirements, and the hidden multiplications involved in accessing their elements contribute to CPU time. However, the operations in step $U3$ could be implemented using only one matrix. In fact, no matrix notation is necessary. Steps $U3$ and $U4$ could be replaced by:

For $q = 1, 2, \ldots, n$ set

$$r_{n-q} \leftarrow P_0 a_{m-n+q} - \sum_{i=1}^{m+1-n} \left(\sum_{\mu=1}^{\min(n+1-q,i)} (P_{i+1-\mu}) b_{q+\mu-1} \right) a_{m-n+1-i} \ . \quad (8)$$

These different versions of algorithm U would imply a tradeoff between memory and CPU requirements.

Exact memory requirements for the three algorithms depend on their implementation. For polynomials over the integers, a quick inspection of algorithms S, T, and U reveals that their memory requirements are bounded below by $m + n$, $2(m + 1)$, and $2(m + 1)$ respectively. In algorithm U, this lower bound is $2(m + n(m + 1 - n))$ if step $U3$ is implemented as specified. In the multivariate case where the coefficients in A and B are themselves polynomials, the bounds above represent the number of pointers to be used. In this, as well as the case of a general commutative ring, where the division process is carried through symbolically, the actual storage requirements are the above bounds added to a much larger constant. This constant depends on the amount of reduction or simplification that could be performed among terms of the form $a_i b_0^{m+1-n-k} b_{\lambda_1} \ldots b_{\lambda_k}$, and is independent of the algorithm chosen. Therefore, no significant superiority can be claimed in favour of one of the algorithms, as far as memory requirements are concerned.

In estimating CPU time requirements one must take into account the organisational overhead involved in algorithms T and U. However, if one confines oneself to counting the operations * and /, it can be shown that exactly $(m - n + 1)(m + 3n)/2$ such operations are performed in algorithm S. A similar operation count for any implementation of algorithms

Table 1

	$m = 50$							
n	15	12	9	6	5	4	3	2
S	29	29	27	25	26	24	26	23
T	244	178	108	53	35	22	13	4
U	173	133	90	54	39	29	20	9
U^*	115	86	61	34	25	17	12	6

T or U would result in an expression involving the dominant term $n^2(m - n)$. Clearly, algorithms T and U are more time efficient for m sufficiently larger than n. They also enable computation of only part of the remainder.

The above conclusions have been substantiated by the results of computer programs implementing the three algorithms. The programs were run on a CDC CYBER-73 computer. **Table 1** shows results pertaining to the division of a polynomial of degree 50 by polynomials of varying degree n. In the table U stands for the algorithm as specified, and U^* stands for a version based on equation (8). Except for m and n, all entries are in milliseconds.

References

Brown, W. S. (1971). On Euclid's algorithm and the computation of polynomial greatest common divisors, *JACM*, Vol. 18, pp. 478-504.

Brown, W. S., and Traub, J. F. (1971). On Euclid's algorithm and the theory of subresultants, *JACM*, Vol. 18, Oct 1971 pp. 505-514.

Brown, W. S., Hyde, J. P., and Tague, B. A. (1963). The ALPAK system for non-numerical algebra on a digital computer, Pt. I: *Bell System Tech. J.*, Vol. 42, No. 5, Sept. 1963, pp. 2081-2119; Pt. II: *Ibid.*, Vol. 43, No. 2, March 1964, pp. 785-804; Pt. III: *ibid.*, Vol. 43, No. 4, July 1964, pp. 1547-1562.

Collins, G. E. (1966). PM, a system for polynomial manipulation, *CACM*, Vol. 9, No. 8, pp. 578-589.

Collins, G. E. (1967). Subresultants and reduced polynomial remainder sequences, *JACM*, No. 14, pp. 128-142.

Collins, G. E. (1971). The SAC-1 system: an introduction and survey, *Proc. 2nd Symposium on Symbolic and Algebraic Manipulation*, ACM, New York, pp. 144-152.

Knuth, D. E. (1969). *The Art of Computer Programming*, Vol. 2: Semi-numerical Algorithms, Addison-Wesley, Reading, Mass.

Book reviews

Applications of Algol 68, Conference proceedings edited by V. J. Rayward-Smith, 1976; 264 pages. (*University of East Anglia*, £6·50)

'Which of languages *W*, *X*, *Y*, . . ., is best suited to my problem?' is the first question that a multi-lingual programmer should ask when starting a new project. This book contains evidence on how well or ill suited ALGOL 68 (or more often ALGOL 68R) has already shown itself to be in a dozen or more projects in fields as diverse as automatic text editing, compiler writing, polynomial manipulation, a mailing list system, interactive graphics, and initial programming courses. Other papers—there are 26 in all—deal with more general experience such as use of subsets, introduction of high level macros, and the results of some benchmark tests.

Most of the myths are shown up for what they are—wishful thinking generated as counter-propaganda by competitors in the rat race. Thus at the Pierre et Marie Curie University in Paris, use of ALGOL 68 in an initial programming course for non-specialists led to sounder problem analysis and shorter debugging times than in previous years; the only 'undesirable' result was that as the good became better the gap between good and bad students widened. 'Sceptic comment' (= sceptical ?, or septic ?) from colleagues showed itself to be based on fears that shallow understanding of the nature of computing would be shown up. Of particular interest is the benchmark report; untuned programs in ALGOL 60, FORTRAN and ALGOL 68 ran at much the same speed; each language had its own style of 'tuning for speed' and ALGOL 68 could (admittedly at some trouble) be more finely tuned than the others. It then reached a speed hardly distinguishable from Pascal, a language in which facilities have been denied to the user in order to force the runtime pace. But where there is criticism, 68R is often preferred, an unexpected conclusion which seems to have a moral for compiler writers, to wit, that if you need facilities to debug your compiler which run counter to your high level philosophy (e.g. procedure bodies in machine code), these should be left in as 'additional, possibly machine dependent' facilities, because sooner or later some user will also need them, and if you withdraw them from the compiler before releasing it, he will reject your implementation.

B. Higman (Lancaster)

Annual Review in Automatic Programming, Vol. 7, 1974. (*Pergamon Press*, £2·25)

This volume is one of the International Tracts in Computer Science and Technology and their Application. The General Editors were N. Metropolis, E. Piore and S. Ulam. The administrative editors were Mark I. Halpern and William C. McGee. The contributing editors were Louis Bolliet, Andrei P. Ershov, and J. P. Laski (however, none of these appear to have contributed). The contents are: A Tutorial on Data-Base Organisation, R. W. Engles; General Concepts of the Simula 67 Programming Language, J. D. Ichbiah and S. P. Morse; Incremental Compilation and Conversational Interpretation, M. Berthaud and M. Griffiths; Dynamic Syntax: A Concept for the Definition of the Syntax of Programming Languages, K. V. Hanford and C. B. Jones; An Introduction to ALGOL 68, H. Bekic; A General Purpose Conversational System for Graphical Programming, O. Lecarme; Automatic Theorem Proving Based on Resolution, A. Pirotte; A Survey of Extensible Programming Languages, N. Solntseff and A. Yezerski.

This is a collection of tutorial and survey articles, with a sprinkling of novel ideas; they are of a kind and length which are not readily published elsewhere; and their publication in an occasional review is to be welcomed. The article on Automatic Theorem Proving by Resolution is particularly clear, though the method is not now considered as promising as it once was.

C. A. R. Hoare (Belfast)