THE SAC-1 POLYNOMIAL REAL ZERO SYSTEM*

by

G. E. Collins and L. E. Heindel

Technical Report #93†

August, 1970

Computer Sciences Department
The University of Wisconsin
1210 W. Dayton Street
Madison, Wisconsin 53706

†This report is also being published as the University of Wisconsin Computing Center Technical Report No. 18.

TABLE OF CONTENTS

THE SAC-1 POLYNOMIAL REAL ZERO SYSTEM*

by

G. E. Collins and L. E. Heindel

1. INTRODUCTION

The SAC-1 Polynomial Real Zero System is the latest addition to the SAC-1 system. SAC-1 (system for Symbolic and Algebraic Calculations--version 1) is a computer-independent system written, for the most part, in ASA FORTRAN (see [ASA64] and [ASA65])for manipulating infinite-precision integers and rational numbers, multivariate polynomials over the integers, rational functions over the integers, elements of GF(p) for a single-precision prime p, and multivariate polynomials over GF(p). The previously completed SAC-1 subsystems are the List Processing System [COG67], the Integer Arithmetic System [COG68c], the Polynomial System [COG68a], the Rational Function System [COG68b], The Modular Arithmetic System [COG69b], and the Partial Fraction Decomposition and Rational Function Integration System [COG70a].

The Polynomial Real Zero System requres the implementation of all the previous subsystems except the Partial Fraction Decomposition and Rational Function Integration System. In the Partial Fraction Decomposition and Rational Function Integration System the common block TR4 was introduced, which is also used in this subsystem. Hence we present the definition of this common block: COMMON/TR4/PRIME,PEXP. PRIME = $(p_1, \ldots, p_r)$ is the location of a non-null list of distinct single-precision

*The research described in this report was supported by the National Science Foundation grant GJ-239, by the University of Wisconsin Computing Center, and by the Wisconsin Alumni Research Foundation through the Graduate School Research Committee.

odd primes and PEXP is a FORTRAN integer equal to $\lfloor \log_2(\min\{p_1,\ldots,p_r\})\rfloor$. Thus PEXP is equal to the exponent of the largest power of 2 less than the smallest prime on the list PRIME. The list of primes, PRIME, may be constructed using the subprogram GENPR of the Modular Arithmetic System [COG69b] and PEXP can be computed using the subprogram ELOPF2. These techniques are employed in the test program in Section 4.

It should be pointed out that both ELPOF2 and PPOWER are subprograms of the Partial Fraction Decomposition and Rational Function Integration System and care should be taken to include these subprograms onlyonce if both subsystems are implemented. The algorithm description and FORTRAN code listing of PPOWER were inadvertently left out of the report "The SAC-1 Partial Fraction Decomposition and Rational Function Integration System" [COG70a] and can be found in this report.

The SAC-1 Polynomial Real Zero System uses the infinite-precision arithmetic, modular arithmetic, and the symbolic manipulation capabilities of the SAC-1 system to solve the following problems for all univariate polynomials with infinite-precision integer coefficients: 1) given a polynomial P with possible multiple zeros to compute a polynomial Q with the same zeros as P but only occurring as simple zeros, 2) determine the total number of real zeros a polynomial with all simple zeros has, 3) determine the number of real zeros a polynomial with all simple zeros has in an interval, 4) isolate the real zeros of a polynomial with all simple zeros, and 5) refine the real zeros of a polynomial with all simple zeros until the error in them is less than some arbitrary positive rational number.

By isolation of the real zeros of a polynomial we mean finding real intervals $I_1, I_2, \ldots, I_k$ such that each interval I_i, $1 \le i \le k$, contains exactly one real zero of the polynomial, and every real zero of the polynomial is contained in some interval I_i. By refinement of a real zero of a polynomial we mean, given an interval containing exactly one real zero of the polynomial, to replace the interval by successively smaller subintervals which still contain the zero until the last one is shorter in length than some arbitrary positive rational error bound.

The algorithms of the SAC-1 Polynomial Real Zero System can be divided into three sets. The first set uses infinite-precision integer and rational arithmetic to isolate and refine real zeros; the second set uses modular arithmetic and the Chinese Remainder Theorem to isolate and refine real zeros; and the third set consists of miscellaneous algorithms shared by the first two sets. Before going on to discuss how the algorithms solve the five previously mentioned problems, we present the following definitions and theorems which provide the basis of the solutions.

Definition 1: Define for real a,b, the interval $I_{a,b} = (a,b] = \{x: a < x \le b\}$, if $a < b$, and $I_{a,b} = \{a\}$, if $a = b$.

Definition 2: Define the SAC-1 representation of the interval $I_{a,b}$, for rational a,b, to be the list (a,b).

Definition 3: Define the length of the interval $I_{a,b}$ to be b-a.

Definition 4: Say that $I_{a,b} \prec I_{c,d}$ in case $b < c$.

Definition 5: Say that $I_{a,b} \preceq I_{c,d}$ in case $b \le c$.

Definition 6: Say that the polynomial P is square-free in case there is no polynomial Q of positive degree such that $Q^2 | P$.

Definition 7: Say that Q is a square-free divisor of P if Q is square-free and $Q|P$.

Definition 8: Say that Q is a greatest square-free divisor of P if Q is a square-free divisor of P, and $\bar{Q}|Q$ whenever $\bar{Q}$ is any other square-free divisor of P.

Notice that if P is a primitive polynomial with integer coefficients and Q is a greatest square-free divisor of P then $\pm Q$ are the only greatest square-free divisors of P. If $\bar{P} = a \cdot P$ where P is a primitive polynomial whose greatest square-free divisors are $\pm Q$, then the greatest square-free divisors of $\bar{P}$ are $\pm a \cdot Q$.

Definition 9: Say that the sign of a real number x is -1 if $x < 0$, is 0 if $x = 0$, and is $+1$ if $x > 0$.

Definition 10: Let $X = (x_1, x_2, \ldots, x_r)$ be any sequence of real numbers. Then the number of variations in sign of X is defined to be the number of pairs (i,j) such that $1 \le i < j \le r$, $x_i \cdot x_j = -1$ and $x_k = 0$ for $i < k < j$.

Definition 11: Let $F_1(x)$ and $F_2(x)$ be two continuous real-valued functions of a single real variable x. A Sturm sequence for F_1 and F_2 is a sequence $F_1, F_2, \ldots, F_r$ of continuous real-valued functions of a single real variable satisfying the following three properties:

1. If $F_1(y) = 0$, then there exists an $\varepsilon > 0$ such that $F_2(x) \neq 0$ for $y - \varepsilon < x < y + \varepsilon$ and $x \neq y$, $F_1(x) \cdot F_2(x) < 0$ for $y - \varepsilon < x < y$, and $F_1(x) \cdot F_2(x) > 0$ for $y < x < y + \varepsilon$.
2. If $1 < i < r$ and $F_i(x) = 0$, then $F_{i-1}(x) \cdot F_{i+1}(x) = 0$.
3. $F_r(x) \neq 0$ for all real x.

Definition 12: Let P_1 and P_2 be two univariate polynomials with integer coefficients. Let n_i = degree (P_i), $n_1 \geq n_2 > 0$, and let P_i be defined by the following three equations:

$$\text{ldcf}(P_{i-1})^{n_{i-2}-n_{i-1}+1} \cdot P_{i-2} = Q_i \cdot P_{i-1} + R_i,$$

$$P_i = R_i/\text{cont}(R_i),$$

if $\text{ldcf}(P_{i-1})^{n_{i-2}-n_{i-1}+1} < 0$, and:

$$P_i = - R_i/\text{cont}(R_i),$$

otherwise, for $3 \leq i \leq r$, where R_{r+1} is the first $R_i = 0$. Then we shall call the sequence $P_1, P_2, \ldots, P_r$ a negative primitive polynomial remainder sequence for P_1 and P_2.

Definition 13: Let $P = (P_1, P_2, \ldots, P_r)$ be a p.r.s. over the integers. Then we shall say that $\bar{P} = (\emptyset_p^*(P_1), \emptyset_p^*(P_2), \ldots, \ldots, \emptyset_p^*(P_r))$ is the image of P in GF(p), where $\emptyset_p$ is the unique homomorphism of the integers I onto GF(p) such that $\emptyset_p(i) = i$ for $0 \leq i < p$ and $\emptyset_p^*$ is the homomorphism from I[x] onto GF(p)[x] induced by $\emptyset_p$. If P is a Sturm sequence, then we shall say that $\bar{P}$ is a Sturm sequence over GF(p).

Definition 14: Let P_1 and P_2 be two univariate polynomials over the integers, degree $(P_1) \geq$ degree $(P_2) \geq 0$, and let $P_1, P_2, \ldots, P_r, P_{r+1}$ be a polynomial remainder sequence satisfying the following equation for $3 \leq i \leq r + 1$:

$$a_i \cdot P_{i-2} = Q_i \cdot P_{i-1} - b_i \cdot P_i,$$

where $a_i \cdot b_i > 0$ and P_{r+1} is the first $P_i = 0$. Then we shall say that $P_1, P_2, \ldots, P_r$ is a negative polynomial remainder sequence.

Definition 15: Let A_1 and A_2 be univariate polynomials over some integral domain I. Let $A_1, A_2, \ldots, A_r, A_{r+1}$ be a p.r.s. over the quotient field of I satisfying the following formula:

$$A_i = A_{i+1} \cdot Q_i - A_{i+2},$$

where A_{r+1} is the first $A_i = 0$. Let $n_i = \text{degree}(A_i)$, $\delta_i = n_i - n_{i+1}$, and $a_i = \text{ldcf}(A_i)$, let $B_1 = A_1$, $B_2 = A_2$, and:

$$B_k = (\pi_{i=2}^{k-2} a_i^{\delta_{i-1} - \delta_i}) \cdot a_{k-1}^{\delta_{k-2}+1} \cdot A_k,$$

for $3 \le k \le r$. Then we shall say that $B_1, B_2, \ldots, B_r$ is a quasi-negative subresultant p.r.s. for A_1 and A_2.

Definition 16: Let A be a non-zero univariate polynomial with real coefficients. Say that A is positive in case its leading coefficient is positive.

Theorem 1: Let P be a univariate polynomial over the integers, degree $(P) > 0$. Then $P/\gcd(P,P')$ is a greatest square-free divisor of P.

Theorem 2: Let $P(x) = \sum_{i=0}^{m} a_i x^i$ be a univariate polynomial with complex coefficients, degree $(P) = m > 0$, and let

$$B = \max\{2\left|\frac{a_{m-k}}{a_m}\right|^{1/k} \mid 1 \le k \le m\} \neq 0.$$

Then if x_o is any complex number such that $P(x_o) = 0$, $|x_o| < B$.

Theorem 3: (Generalized Sturm's Theorem): Let $F_1(x)$ be a continuous real-valued function of a single real variable, $F_1, F_2, \ldots, F_r$ a Sturm sequence for F_1 and F_2, and let $w(x)$ be the function whose value is the number of variations in sign of the sequence $F_1(x), F_2(x), \ldots, F_r(x)$. Then for $a < b$ the number of distinct real zeros of $F_1(x)$ in the interval $(a,b]$ is $w(a) - w(b)$.

Theorem 4: Let P_1 be a primitive, positive square-free univariate polynomial of positive degree with integer coefficients, let P_2 be the primitive part of the derivative pf P_1, and let $P_1, P_2, \ldots, P_r$ be a negative primitive polynomial remainder sequence. Then $P_1, P_2, \ldots, P_r$ is a Sturm sequence for P_1 and P_2.

Theorem 5: Let P_1 be a primitive, positive square-free univariate polynomial of positive degree with integer coefficients, let P_2 be the primitive part of the derivative of P_1, and let $P_1, P_2, \ldots, P_r$ be a negative polynomial remainder sequence. Then $P_1, P_2, \ldots, P_r$ is a Sturm sequence for P_1 and P_2.

The above definitions and theorems are taken from [HEL70], where the algorithms in this report are developed in detail.

Let us now look at how the algorithms in this report solve the five previously mentioned problems.

First we describe how, given a polynomial P with possible multiple zeros, to compute a polynomial with the same zeros as P but only occurring as simple zeros. By Theorem 1, if P is a univariate polynomial over the integers, degree $(P) > 0$, then $P/\gcd(P,P')$ is a greatest square-free divisor of P. This greatest square-free divisor has the same zeros as P, but only occurring as simple zeros.

The second and third problems are handled by the technique of applying Theorem 3 (Generalized Sturm's Theorem) which enables one to determine the number of real zeros of a polynomial having only simple zeros in a left-open, right-closed interval. To determine how many zeros a polynomial with simple zeros has, an integer B^* is computed which is equal to or greater than the B described in Theorem 2. Then all the real zeros of the polynomial lie in the interval $(-B^*, B^*]$.

We now look at how to isolate the real zeros of a polynomial P with only simple zeros. We again compute an interval $I = (-B^*, B^*]$ as described in the preceding paragraph. Then the Generalized Sturm's Theorem is applied to the interval I to determine the number of zeros in the interval. If the interval I contains one or no real zero, then the zeros of P have been isolated. If I contains two or more real zeros, I is bisected and then the Generalized Sturm's Theorem is used to determine the number of zeros in each subinterval. Any subinterval containing no real zero is discarded, any subinterval containing more than one real zero is bisected and the process repeated. By this process all the real zeros end up isolated into disjoint intervals.

Before looking at the last problem, we discuss briefly how the integer and modular arithmetic sets of algorithms apply the Generalized Sturm's Theorem. The integer arithmetic set of algorithms computes a negative primitive polynomial remainder sequence over the integers for the square-free polynomial P and its derivative, P', which by Theorem 4 is a Sturm sequence. Infinite-precision integer arithmetic is then used to determine the signs of the Sturm sequence at the appropriate points in order to apply the Generalized Sturm's Theorem.

The modular arithmetic set of algorithms computes quasi-negative subresultant p.r.s.'s over $GF(p_i)$ starting with $\emptyset^*_{p_i}(P)$ and $\emptyset^*_{p_i}(P')$ for a number of primes p_i and then decides which of these sequences are images of the quasi-negative subresultant polynomial remainder sequence over the integers starting with P and P'. Sign correction factors are then computed which allow these quasi-negative subresultant p.r.s.'s to be converted into negative p.r.s.'s. These negative p.r.s.'s are images of a Sturm sequence for P and P' by Theorem 5 and the homomorphisms from the integers onto $GF(p_i)$. Modular arithmetic and the Chinese Remainder Theorem are then used to determine the signs of the Sturm sequence over the integers at the appropriate points to apply the Generalized Sturm's Theorem.

We now look at the last problem, which is how to refine the real zeros of a polynomial P with all simple zeros until their errors are less than some arbitrary positive rational number. Suppose $I = (a,b]$ is an isolating interval. Then either b is a zero of P or P changes sign between a and b, since P has only simple zeros. Hence P is first evaluated at b. If $P(b) = 0$, then the zero in the interval I_i has been found exactly. Otherwise P is evaluated at $\frac{a+b}{2}$. Again if $P(\frac{a+b}{2}) = 0$, the root has been found exactly. If $P(\frac{a+b}{2}) \neq 0$ and $P(\frac{a+b}{2}) \cdot P(b) < 0$, then the zero in the interval I is contained in the subinterval $(\frac{a+b}{2}, b]$; otherwise it is contained in the subinterval $(a, \frac{a+b}{2}]$. The length of whichever subinterval contains the zero is checked to see if it is equal to or less than the arbitrary positive rational error bound. If it is, then the interval has been sufficiently refined; otherwise this subinterval is bisected and the process is repeated.

The last algorithm to be discussed is IPRINT, which outputs the endpoints of intervals having powers of 2 as denominators. If an endpoint is zero or has a denominator of 1, it is output as an integer. Otherwise, let a/b be an endpoint, $b = 2^k$ for some $k > 0$. Then we can compute a q and an r such that $a = b \cdot q + r$, $0 < |r| < b$ ($r \neq 0$ since $b > 1$ and $\gcd(a,b) = 1$). Now q is the integer part of the decimal equivalent of a/b.

To determine the fractional part, we observe that $\frac{|r|}{b} = \frac{5^k \cdot |r|}{5^k \cdot 2^k} = \frac{5^k \cdot |r|}{10^k}$. The last rational number has a denominator which is a power of 10. Hence the fractional part can be computed by computing $5^k \cdot |r|$ and then prefixing enough zeros and a decimal point that there are k decimal places. The decimal equivalent of a/b then is the concatenation of its sign and the integral and fractional parts.

Notice that if IPRINT is used, for instance, to output the results of isolating the roots of a polynomial, the number of decimal places in the output may be much greater than the number of decimal positions to which the result is accurate, and this must be taken into account when interpreting the output.

In Section 2, descriptions of the algorithms are presented along with the theoretical computing times of the various algorithms. The theoretical computing times of the algorithms are taken from [HEL70] and are maximum computing time bounds. Whether these bounds can actually be reached is an open question. Section 3 contains empirical computing time results of running the various algorithms. Section 4 contains a test program, and Section 5 contains the FORTRAN subprogram listings.

2. ALGORITHM DESCRIPTIONS

2.1 Integer Arithmetic Algorithms

Algorithm LI = ANALR(Q,Y)

Input: Q, a square-free, positive univariate polynomial of positive degree with integer coefficients; and Y, a rational number equal to or greater than zero.

Output: If Q has no real zeros, LI is set to zero. If Q has any real zeros then LI is a list $(I_1,\ldots,I_r)$, $I_1 \preceq I_2 \preceq \cdots \preceq I_r$, where each I_i, $1 \le \le r$ contains exactly one real zero of Q and every real zero of Q is contained in some I_i. If $Y \neq 0$, the length of each of the I_i is equal to or less than Y.

(1) P1 ← Q; EPS ← Y; R ← RBOUND(P1); L ← RDIF(O,R); INTER ← PFL (L,PFL(R,O)); IN ← ISOLT(P1,INTER); erase INTER; if IN $\neq$ 0 and EPS $\neq$ 0, go to (2); LI ← IN; return.

(2) LI ← 0.

(3) ADV(I,IN); LI ←PFL(REFINE(I,EPS), LI); if IN $\neq$ 0, go to (3); LI ← INV(LI); return.

Computing Time: $O(m^{10}(\ln d)^3 + m^3(\ln \frac{d}{y})(m^2(\ln d) + (\ln \frac{1}{y}))^2)$, if $0 < y \le 1$, and $O(m^{10}(\ln d)^3)$, if $1 < Y$ or $y = 0$, where m = degree (Q), and d = norm(Q).

Algorithm N = IRTS(X,Y)

Input: X a square-free, positive polynomial of positive degree with integer coefficients; and Y, an interval.

Output: N, the number of real zeros of X in the interval Y.

(1) P1 ← X; INTER ← Y; DUM ←0; RRTS(P1,INTER,1,DUM,0,DUM1); erase DUM; return.

Computing Time: $O(m^3(\ln c)^2 + m^4(\ln c)(\ln md) + m^4(\ln md)^2)$, if the Sturm sequence computed is normal and $O(m^3(\ln c)^2 + m^4(\ln c)(\ln md) + m^5(\ln md)^2)$ if it is not, where $Y = (\frac{a}{b}, \frac{e}{f}]$, $c = \max\{|a|, |e|, b, f\}$, $m = \text{degree}(x)$, and $d = \text{norm}(x)$.

Algorithm INS = ISOLT(X,Y)

Input: X, a square-free, positive univariate polynomial of positive degree with integer coefficients; and Y, an interval.

Output: $INS = (I_1, \ldots, I_r)$, $I_1 \preceq I_2 \preceq \cdots \preceq I_r$, where each I_i, $1 \le i \le r$, contains exactly one real zero of X and each real zero of X in the interval Y is contained in one of the I_i.

(1) P1 ← X; INTER ← Y; DUM ← 0; RRTS(P1,INTER,1,DUM,DUM1,INS); erase DUM; return.

Computing Time: Let $m = \text{degree}(x) > 0$, $d = \text{norm}(x)$, let $Y = (\frac{a1}{b1}, \frac{a2}{b2}]$ be an interval of length h containing r real zeros, $e = \max\{|a_1|, |a_2|, b_1, b_2\}$, and if $r \ge 1$, let $\lambda = \min\{|\alpha - \beta| : X(\alpha) = 0, X(\beta) = 0, \alpha \ne \beta \text{ and } \alpha, \beta \in Y\}$. Then if $r \ge 2$, the maximum computing time is:

$$O(rm^3(\ln \tfrac{h}{\lambda})(\ln \tfrac{eh}{\lambda})^2 + rm^4(\ln \tfrac{h}{\lambda})(\ln \tfrac{eh}{\lambda})(\ln md) + m^4(\ln md)^2),$$

if the Sturm sequence constructed is normal, and:

$$O(rm^3(\ln \tfrac{h}{\lambda})(\ln \tfrac{eh}{\lambda})^2 + rm^4(\ln \tfrac{h}{\lambda})(\ln \tfrac{eh}{\lambda})(\ln md) + m^5(\ln md)^2),$$

if the Sturm sequence constructed is non-normal.

If $r \le 1$, the maximum computing time is:

$$O(m^3(\ln e)^2 + m^4(\ln e)(\ln md) + m^4(\ln md)^2),$$

if the Sturm sequence constructed is normal, and:

$$O(m^3(\ln e)^2 + m^4(\ln e)(\ln md) + m^5(\ln md)^2),$$

if the Sturm sequence constructed is non-normal.

Algorithm SEQ = ISTURM(X,Y)

Input: X and Y are two univariate polynomials with integer coefficients, degree (X) > 0, degree (Y) ≥ 0, degree (X) ≥ degree (Y).

Output: SEQ = $(X,Y,P_3,\ldots,P_r)$ where $X,Y,P_3,\ldots,P_r$ is a negative primitive polynomial remainder sequence for X and Y.

(1) P1 ← X; P2 ← Y; SEQ ← PFL(BORROW(P2), PFL(BORROW(P1),0)); DP1 ← PDEG(P1).

(2) DP2 ← PDEG(P2); DUM ← PSREM(P1,P2); if DUM ≠ 0, go to (3); SEQ ← INV(SEQ); return.

(3) D ← DP1-DP2; SP2 ← PSIGN(P2); P1 ← P2; DP1 ← DP2; DUM1 ← PCONT(DUM); P2 ← PSQ(DUM,DUM1); erase DUM,DUM1; if SP2 = -1 and SP2**D = 1, go to (4); DUM ← P2; P2 ← PNEG(P2); erase DUM.

(4) SEQ ← PFL(P2,SEQ); go to (2).

Computing Time: $O)m^4(\ln d)^2)$, if the negative primitive p.r.s. computed is normal and $O(m^5(\ln d)^2)$ if it is not, where m = degree (X), and d bounds the norms of X and Y.

Algorithm V = IVAR(X,Y)

Input: X, a Sturm sequence; and Y, a rational number.

Output: V, the number of variations in sign of the Sturm sequence X at the rational point Y .

(1) SEQ ← X; PT ← Y; V ← 0; LS ← 0.

(2) ADV(P,SEQ); S ← REVAL(P,PT); if LS = 0, go to (3); if S = 0 or S = LS, go to (4); V ← V + 1.

(3) LS $\leftarrow$ S.

(4) If SEQ $\neq$ 0, go to (2); return.

Computing Time: $O(m^2(\ln md))$ if $Y = 0$ and $O(m^3(\ln |a| \cdot b)^2 + m^4(\ln |a| \cdot b)(\ln md))$ if $Y = \frac{a}{b}$, $a \neq 0$ and $b > 0$, where the Sturm sequence is the primitive Sturm sequence generated from P and P', m = degree (P) > 0, d = norm (P).

Algorithm N = NRTS(X)

Input: X, a square-free, positive univariate polynomial of positive degree with integer coefficients.

Output: N, the number of real zeros of the polynomial X.

(1) P1 $\leftarrow$ X; R $\leftarrow$ RBOUND(P1); L $\leftarrow$ RDIF(0,R); INTER $\leftarrow$ PFL(L,PFL(R,0)); N $\leftarrow$ IRTS(P1,INTER); erase INTER; return.

Computing Time: $O(m^4(\ln md)^2)$, if the Sturm sequence computed is normal and $O(m^5(\ln md)^2)$ if it is not, where m = degree(X) and d = norm(X).

Algorithm I = REFINE(X,Y,Z)

Input: X, a square-free univariate polynomial over the integers; Y, an interval containing exactly one real zero of X; and Z, a positive rational number.

Output: I, a subinterval of the interval Y containing a real zero of the polynomial X. The length of I is equal to or less than Z .

(1) P $\leftarrow$ X; IN $\leftarrow$ Y; EPS $\leftarrow$ Z; HALF $\leftarrow$ PFL(PFA(1,0), PFL(PFA(2,0),0)); ADV(L,IN); ADV(R,IN); L $\leftarrow$ BORROW(L); R $\leftarrow$ BORROW(R); SR $\leftarrow$ REVAL(P,R), if SR = 0, go to (8).

(2) DIF $\leftarrow$ RDIF(R,L); DUM $\leftarrow$ RCOMP(DIF,EPS); erase DIF; if DUM = 1, go to (3); I $\leftarrow$ PFL(L,PFL(R,0)); go to (7).

(3) SUM ← RSUM(L,R); MID ← RPROD(SUM,HALF); erase SUM; SMID ← REVAL (P,MID); if SMID = 0, go to (5); if SMID = SR, go to (4); erase L; L ← MID; go to 2.

(4) Erase R; R ←MID; go to (2).

(5) Erase R.

(6) I ←PFL(MID,PFL(BORROW(MID),0)); erase L.

(7) Erase HALF; return.

(8) MID ← R; go to (6).

Computing Time: $O(m^2(\ln \frac{h}{z})(\ln \frac{eh}{z})(\ln \frac{ehd}{z}))$, for $Z < h$ and $O(m^2(\ln e)(\ln ed))$ for $Z \geq h$, where $Y = (\frac{a_1}{b_1}, \frac{a_2}{b_2}]$, $e = \max\{|a_1|, |a_2|, b_1, b_2\}$, h is the length of Y, m = degree (X), and d = norm (X).

Algorithm S = REVAL(Q,P)

Input: Q, a univariate polynomial with integer coefficients; and P, a rational number

Output: S = sign (Q(P)).

(1) S ← 0; if Q = 0, return; QI ← TAIL(Q); if P ≠ 0, go to (4).

(2) ADV(CI,QI); ADV(EI,QI); if QI ≠ 0, go to (2).

(3) If EI = 0, S ←ISIGNL(CI); return.

(4) A ←FIRST(P); B ← FIRST(TAIL(P)); ADV(S,QI); S ←BORROW(S); ADV(I,QI); BI ← PFA(1,0); go to (7).

(5) T ←IPROD(S,A); erase S; S ← T; T ←IPROD(BI,B); erase BI; BI ← T; if QI = 0, go to (7); if FIRST(TAIL(QI) < I, go to (7).

(6) ADV(CI,QI); QI ← TAIL(QI); T ← IPROD(CI,BI); U ← ISUM(S,T); erase S,T; S ← U.

(7) I ← I - 1; if I ≥ 0, go to (5).

(8) Erase BI; T ← ISIGNL(S); erase S; S ← T; return.

Computing Time: $O(m + (\ln d))$, if $P = 0$ and $O(m^2(\ln |a| \cdot b)^2 + m^2(\ln d)(\ln |a| \cdot b))$ if $P = \frac{a}{b}$, $b > 0$, $m = \text{degree}(Q)$, and $d = \text{norm}(Q)$.

Algorithm RRTS(Q,XX,F,SEQ,ROOTS,ISOLAT)

Input: Q, a primitive, positive, square-free, univariate polynomial of positive degree with integer coefficients; XX, an interval; F, a FORTRAN integer equal to 0 or 1; and SEQ, either 0 or a primitive Sturm sequence for Q and Q'.

Output: ROOTS, the number of real zeros that Q has in the interval XX; SEQ, a primitive Sturm sequence for Q and Q'; and if F = 1, ISOLAT = $(I_1, I_2, \ldots I_r)$ is a list of subintervals of XX such that each I_i, $1 \le i \le r$, contains exactly one zero of Q in the interval XX, $I_1 \lesssim I_2 \lesssim \ldots \lesssim I_r$, and every real zero of Q in the interval XX is contained in one I_i.

(1) P1 ← Q; INTER ← XX; FLAG ← F; if SEQ ≠ 0, go to (2); DUM ← PDERIV (P1,FIRST(P1)); P2 ← PPP(DUM); erase DUM; SEQ ← ISTURM(P1,P2); erase P2.

(2) ADV(L,INTER); ADV(R,INTER); LV ← IVAR(SEQ,L); RV ← IVAR(SEQ,R); ROOTS ← LV - RV; ISOLAT ← 0; if FLAG = 0 or ROOTS = 0, return; HALF ← PFL(PFA(1,0), PFL(PFA(2,0),0)); LIST ← 0; L ← BORROW(L); R ← BORROW(R).

(3) RTS ← LV − RV; if RTS = 0, go to (7); if RTS ≠ 1, go to (4);

ISOLAT ← PFL(PFL(L,PFL(R,0)),ISOLAT); go to (8).

(4) SUM ← RSUM(L,R); MID ← RPROD(SUM,HALF); erase SUM; MIDV ← IVAR (SEQ,MID); if LV − MIDV = 0, go to (5); LIST ← PFA(LV,PFA(MIDV, PFA(L,PFA(BORROW(MID),LIST)))); go to (6).

(5) Erase L.

(6) L ← MID: LV ← MIDV; go to (3).

(7) Erase L,R.

(8) If LIST = 0, go to (9); DECAP(LV,LIST); DECAP(RV,LIST); DECAP(L,LIST); DECAP(R,LIST); go to (3).

(9) Erase HALF; return.

Computing Time: Let m = degree (Q), d = norm (Q), let $XX = (\frac{a_1}{b_1}, \frac{a_2}{b_2}]$ be an interval of length h containing r real zeros, $e = \max\{|a_1|, |a_2|, b_1, b_2\}$, and if $r \geq 2$, let $\lambda = \min\{|\alpha-\beta| : Q(\alpha) = 0, Q(\beta) = 0, \alpha \neq \beta, \text{ and } \alpha, \beta \in XX\}$. Then if $F = 1$, $r \geq 2$, and SEQ ≠ 0, the maximum computing time is:

$$Q(rm^3(\ln \frac{h}{\lambda})(\ln \frac{eh}{\lambda})^2 + rm^4(\ln \frac{h}{\lambda})(\ln \frac{eh}{\lambda})(\ln md)).$$

If $F = 1$, $r \geq 2$, SEQ = 0, and the Sturm sequence constructed is normal, then the maximum computing time is:

$$O(rm^3(\ln \frac{h}{\lambda})(\ln \frac{eh}{\lambda})^2 + rm^4(\ln \frac{h}{\lambda})(\ln \frac{eh}{\lambda})(\ln md) + m^4(\ln md)^2).$$

If $F = 1$, $r \geq 2$, SEQ = 0, and the Sturm sequence constructed is non-normal, then the maximum computing time is:

$$O(rm^3(\ln \frac{h}{\lambda})(\ln \frac{eh}{\lambda})^2 + rm^4(\ln \frac{h}{\lambda})(\ln \frac{eh}{\lambda})(\ln md) + m^5(\ln md)^2).$$

If $F = 0$ or $F = 1$, $r \leq 1$, and SEQ $\neq 0$, then the maximum computing time is:

$$O(m^3(\ln e)^2 + m^4(\ln e)(\ln md)).$$

If $F = 0$ or $F = 1$, $r \leq 1$, and SEQ $= 0$ and the Sturm sequence constructed is normal, then the maximum computing time is:

$$O(m^3(\ln e)^2 + m^4(\ln e)(\ln md) + m^4(\ln md)^2).$$

If $F = 0$ or $F = 1$, $r \leq 1$, and SEQ $= 0$ and the Sturm sequence constructed is non-normal, then the maximum computing time is:

$$O(m^3(\ln e)^2 + m^4(\ln e)(\ln md) + m^5(\ln md)^2).$$

2.2 Modular Arithmetic Algorithms

Algorithm LI = CANALR(Q,Y)

Input: Q, a square-free, positive univariate polynomial of positive degree with integer coefficients; and Y, a reational number equal to or greater than zero.

Output: If Q has no real zeros, LI is set to zero. If Q has any real zeros and $Y = 0$, LI is a list $(I_1, I_2, \ldots, I_r)$, $I_1 \preceq I_2 \preceq \cdots \preceq I_r$, where each I_i, $1 \leq i \leq r$, contains exactly one real zero of Q and every real zero of Q is contained in some I_i. If $Y \neq 0$, the length of each of the I_i is equal to or less than Y.

(1) P1 ← Q; EPS ← Y; R ← RBOUND(P1); L ← RDIF(0,R); INTER ← PFL(L,PFL(R,0)); DUM ← 0; RROOTS(P1,INTER,1,DUM,ROOTS,IN); erase INTER; if IN ≠ 0 and EPS ≠ 0, go to (2); erase DUM; LI ← IN; return.

(2) LI $\leftarrow$ 0; DECAP(SEQ,DUM); DECAP(LCP1,SEQ); erase SEQ; DECAP(PLIST2, DUM); DECAP(SIGNS,DUM); erase SIGNS; DECAP(NORMS,DUM); DECAP(NORM, NORMS); erase NORMS; DECAP(PLIST1,DUM); DECAP(DEGSEQ,DUM); erase DEGSEQ.

(3) DECAP(I,IN); LI $\leftarrow$ PFL(CREFIN(P1,I,EPS,LCP1,PLIST2,NORM,PLIST1), LI); erase I; if IN $\neq$ 0, go to (3); LI $\leftarrow$ INV(LI); erase LCP1, PLIST1, PLIST2; return.

Computing Time: $O(m^{10}(\ln d)^3 + m^7 (\ln d)^2(\ln \frac{d}{y}) + m^5(\ln d)(\ln \frac{1}{y})(\ln \frac{d}{y}) + m^3(\ln \frac{1}{y})^3)$, if $0 \leq Y \leq 1$, and $O(m^{10}(\ln d)^3)$, if $1 < Y$ or $Y = 0$, where m = degree (Q) and d = norm(Q).

Algorithm S = CNPRS(P,X,Y)

Input: P, a prime; and X and Y, two non-zero univariate polynomials over GF(P) such that degree (X) $\geq$ degree(Y) $\geq$ 0.

Output: $S = (S_1, S_2, \ldots, S_r)$, where $S_1 = X$, $S_2 = Y$, and $S_1, S_2, \ldots, S_r$ is a quasi-negative subresultant p.r.s. for X and Y over GF(P).

(1) P1 $\leftarrow$ X; P1 $\leftarrow$ BORROW(BORROW(P1)); P2 $\leftarrow$ Y; P2 $\leftarrow$ BORROW(BORROW(P2)); S $\leftarrow$ PFL(P2,PFL(P1,0)); DI $\leftarrow$ 1; AIM1 $\leftarrow$ 1; NIM1 $\leftarrow$ FIRST(P1).

(2) DUM $\leftarrow$ CPREM(P,P1,P2); if DUM = 0, go to (3); R $\leftarrow$ CSPROD(P,DUM,P-1,0); erase DUM; AI $\leftarrow$ FIRST(TAIL(P2)); NI $\leftarrow$ FIRST(P2); D $\leftarrow$ NIM1-NI-1; DI $\leftarrow$ CPROD(P,DI,CPROD(P,CPOWER(P,AIM1,D), CPOWER(P,AI,D+2))); S $\leftarrow$ PFL(CSPROD(P,R,DI,0),S); erase P1; P1 $\leftarrow$ P2; P2 $\leftarrow$ R; AIM1 $\leftarrow$ AI; NIM1 $\leftarrow$ NI; go to (2).

(3) Erase P1,P2; S $\leftarrow$ INV(S); return.

<u>Computing Time</u>: $O(m^2)$, where m = degree (X) > 0.

<u>Algorithm</u> I = CREFIN(Q,IN1,E,LCP1,PLIST2,NORM,PLIST1)

<u>Input</u>: Q, a square-free univariate polynomial over the integers; IN1, an interval containing exactly one real zero of Q; E, a positive rational number; LCP1 = $(\phi^*_{p_1}(Q), \phi^*_{p_2}(Q), \ldots, \phi^*_{p_k}(Q))$, where $k \geq 0$ and degree $(\phi^*_{p_i}(Q))$ = degree (Q), $1 \leq i \leq k$; PLIST2 = $(p_1, p_2, \ldots, p_k)$; NORM = $\lceil \log_2(\text{norm}(Q)) \rceil$; PLIST1, a final segment of the list PRIME whose primes were not discarded because degree $(\phi^*_p(Q)) \neq$ degree (Q) nor placed on the list PLIST2; and PEXP = $\lfloor \log_2(\min\{\text{primes on the list PRIME}\}) \rfloor$.

<u>Output</u>: I, a subinterval of the interval IN1 containing a real zero of the polynomial Q, the length of I being equal to or less than E ; LCP1 = $(\phi^*_{p_1}(Q), \phi^*_{p_2}(Q), \ldots, \phi^*_{p_k}(Q), \ldots, \phi^*_{p_\ell}(Q))$, $k \leq \ell$, and degree $(\phi^*_{p_i}(Q))$ = degree (Q), $1 \leq i \leq \ell$; PLIST2 = $(p_1, p_2, \ldots, p_k, \ldots, p_\ell)$; and PLIST1 a final segment of the list PRIME whose primes were not discarded because degree $(\phi^*_p(Q)) \neq$ degree (Q) nor placed on the list PLIST2.

[Steps (1) - (10) refine the interval in a manner analogous to algorithm REFINE.]

(1) P1 ← Q; IN ← IN1; EPS ← E; HALF ← PFL(PFA(1,0), PFL(PFA(2,0),0)); LDCFP1 ← FIRST(TAIL(P1)); DEGP1 ← PDEG(P1); L ← BORROW(FIRST(IN)); XX ← PLIST2; R ← BORROW(FIRST(TAIL(IN))); X ← R; J ← 1; go to (11).

(2) SR ← SEVAL(LCP1,PLIST2,NPRIME,R); if SR = 0, go to (10).

(3) DIF ← RDIF(R,L); DUM ← RCOMP(DIF,EPS); erase DIF; if DUM = 1, go to (4); I ← PFL(L,PFL(R,0)); go to (9).

(4) SUM ← RSUM(L,R); MID ← RPROD(SUM,HALF); erase SUM; X ← MID; J ← 2; go to (11).

(5) SMID ← SEVAL(LCP1,PLIST2,NPRIME,MID); if SMID = 0, go to (7), if SMID = SR, go to (6); erase L; L ← MID; go to (3).

(6) Erase R; R ← MID; go to (3).

(7) Erase R.

(8) I ← PFL(MID,PFL(BORROW(MID),0)); erase L.

(9) Erase HALF; PLIST1 ← BORROW(PLIST1); erase XX; return.

(10) MID ← R; go to (8).

[Check if enough images are available to do the evaluation, if not construct some more.]

(11) If X ≠ 0, go to (12); ECEIL ← 0; go to (15).

(12) N ← IABSL(FIRST(X)); D ← (FIRST(TAIL(X)); if ICOMP(N,D) = 1, go to (13); ELPOF2(D,DUM,ECEIL); go to (14).

(13) ELPOF2(N,DUM,ECEIL)

(14) Erase N.

(15) NPRIME ← [(NORM+DEGP1*ECEIL)/PEXP] + 1; NP ← NPRIME - LENGTH(PLIST2).

(16) If NP ≤ 0, go to (2,5), J.

(17) If PLIST1 = 0, go to (18); ADV(GFP,PLIST1); if CMOD(GFP,LDCFP1) = 0, go to (17); PLIST2 ← PFA(GFP,PLIST2); LCP1 ← PFL(CPMOD(GFP,P1), LCP1); NP ← NP - 1; go to (16).

(18) Print 'INSUFFICIENT PRIMES, CREFIN'; stop.

Computing Time: $O(m^2(\ln \frac{eh}{E})((\ln d) + (\ln \frac{h}{E})) + m(\ln d)((\ln d) + (\ln \frac{h}{E})(\ln \frac{eh}{E})) + (\ln \frac{h}{E})(\ln d)^2)$, for $E < h$ and $O(m^2((\ln d)(\ln e) + (\ln e)^2) + m(\ln d)^2)$ for $E \geq h$, where $IN1 = (\frac{a_1}{b_1}, \frac{a_2}{b_2}]$, $e = \max\{|a_1|, |a_2|, b_1, b_2\}$, h is the length of IN1, m = degree(Q), d = norm(Q), and LCP1 = PLIST2 = 0.

Algorithm N = IROOTS(X,Y)

Input: X, a square-free, positive polynomial of positive degree with integer coefficients; and Y, an interval whose endpoints are either zero or whose denominators are non-negative powers of 2.

Output: N, the number of real zeros of X in the interval Y.

(1) P1 ← X; INTER ← Y; DUM ← 0; RROOTS(P1,INTER,0,DUM,N,DUM1); erase DUM; return.

Computing Time: $O(m^4(\ln md) + m^3(\ln cmd)^2)$, where $Y = (\frac{a}{b}, \frac{e}{f}]$, $c = \max\{|a|, |e|, b, f\}$, m = degree (X), and d = norm (X).

Algorithm INS = ISOLAT(X,Y)

Input: X, a square-free, positive univariate polynomial of positive degree with integer coefficients; and Y, an interval whose endpoints are either zero or whose denominators are non-negative powers of 2.

Output: $INS = (I_1, \ldots, I_r)$, $I_1 \preceq I_2 \preceq \cdots \preceq I_r$, where each I_i, $1 \leq i \leq r$, contains exactly one real zero of X and each zero of X in the interval Y is contained in one of the I_i.

(1) P1 ← X; INTER ← Y; DUM ← 0; RROOTS(P1,INTER,1,DUM,DUM1,INS);

erase DUM; return.

Computing Time: Let m = degree (X) > 0, d = norm(X), let $X = (\frac{a_1}{b_1}, \frac{a_2}{b_2}]$ be an interval

of length h containing r real zeros, $e = \max\{|a_1|, |a_2|, b_1, b_2\}$, and if

$r \geq 2$, let $\lambda = \min\{|\alpha-\beta| : X(\alpha) = 0, X(\alpha) = 0, \alpha \neq \beta$, and $\alpha, \beta \in Y\}$. Then

if $r \geq 2$, the maximum computing time is:

$$O(m^4(\ln md) + rm^3 (\ln \frac{h}{\lambda})(\ln (\frac{eh}{\lambda})\ md)^2).$$

If $r \leq 1$, the maximum computing time is:

$$O(m^4(\ln md) + m^3(\ln emd)^2).$$

Algorithm N = NROOTS(X)

Input: X, a square-free, positive univariate polynomial of positive degree with integer coefficients.

Output: N, the number of real zeros that X has.

(1) P1 ← X; R ← RBOUND(P1); L ← RDIF(0,R); INTER ← PFL(L,PFL(R,0));

N ← IROOTS(P1,INTER); erase INTER; return.

Computing Time: $O(m^4(\ln md) + m^3 (\ln md)^2)$, where m = degree(X), and d = norm(X).

Algorithm RROOTS (Q,XX,F,YY,ROOTS,ISOLAT)

Input: Q, a primitive, positive, square-free, univariate polynomial of positive degree with integer coefficients; XX, an interval; F, a FORTRAN integer equal to 0 or 1; and YY, either 0 or a list structure identical in form to the output of Algorithm STURM; and PEXP, an integer equal to $\lfloor \log_2 (\min \{\text{primes on the list PRIME}\}) \rfloor$.

Output: ROOTS, the number of real zeros that Q has in the interval XX; YY, a list structure identical in form to the output of Algorithm STURM; and if F = 1, ISOLAT = $(I_1, I_2, \ldots, I_k)$ is a list of intervals such that each I_i, $1 \le i \le k$, contains exactly one zero of Q in the interval XX, $I_1 \preceq I_2 \preceq \ldots \preceq I_k$, and every real zero of Q in the interval XX is contained in one I_i.

[Steps (1) - (13) are completely analogous to the subinterval generation scheme employed in RRTS except that before each application of VAR there is a jump to step (14).]

(1) P1 ← Q; INTER ← XX; FLAG ← F; DUM ← YY; DUM1 ← PDERIV(P1,FIRST(P1)); P2 ← PPP(DUM1); erase DUM1; LCP1 ← FIRST(TAIL(P1)); LCP2 ← FIRST (TAIL(P2)); ISOLAT ← 0; if DUM ≠ 0, go to (2); DUM ← STURM(P1,P2).

(2) DECAP(SEQ,DUM); DECAP(PLIST2,DUM); DECAP(SIGNS,DUM); DECAP(NORMS, DUM); DECAP(PLIST1,DUM); DUM4 ← PLIST1; DECAP(DEGSEQ,DUM); LDEGSQ ← LENGTH(DEGSEQ); ADV(L,INTER); ADV(R,INTER); X ← L; J ← 1; go to (14).

(3) LV ← VAR(SEQ,PLIST2,LNPRMS,SIGNS,L); erase LNPRMS; X ← R; J ← 2; go to (14).

(4) RV ← VAR(SEQ,PLIST2,LNPRMS,SIGNS,R); erase LNPRMS; ROOTS ← LV - RV; if FLAG = 0 or ROOTS = 0, go to (13); LIST ← 0; HALF ← PFL(PFA(1,0), PFL(PFA(2,0),0)); L ← BORROW(L); R ← BORROW(R).

(5) RTS ← LV −RV; if RTS = 0, go to (10); if RTS ≠ 1, go to (6); ISOLAT ← PFL(PFL(L,PFL(R,0)), ISOLAT); go to (11).

(6) SUM ← RSUM(L,R); MID ← RPROD(SUM,HALF); erase SUM; X ← MID; J ← 3; go to (14).

(7) MIDV ← VAR(SEQ,PLIST2,LNPRMS,SIGNS,MID); erase LNPRMS; if LV - MIDV = 0, go to (8); LIST ← PFA(LV,PFA(MIDV,PFA(L,PFA(BORROW(MID),LIST)))); go to (9).

(8) Erase L.

(9) L ← MID; LV ← MIDV; go to (5).

(10) Erase L,R.

(11) If LIST = 0, go to (12); DECAP(LV,LIST); DECAP(RV,LIST); DECAP(L,LIST); DECAP(R,LIST); go to (5).

(12) Erase HALF.

(13) Erase P2; YY ← PFL(NORMS,PFL(BORROW(PLIST1), PFL(DEGSEQ,0))); YY ← PFL(SEQ,PFL(PLIST2,PFL(SIGNS,YY))); erase DUM4, return.

[Steps (14) - (26) determine if enough images of the sequence exist to do the sign determinations in VAR; if not it computes some more images if there are unused primes available. Start by computing ECEIL = $\lceil \log_2 (\max \{|\text{numerator of } X|, \text{denominator of } X\})\rceil$.]

(14) If $X \neq 0$, go to (15); ECEIL ← 0; go to (18).

(15) N ← IABSL(FIRST(X)); D ← FIRST(TAIL(X)); if ICOMP(N,D) = 1, go to (16); ELPOF2 (D,DUM,ECEIL); go to (17).

(16) ELPOF2 (N,DUM,ECEIL).

(17) Erase N.

[Make up a list LNPRMS = $(n_1, n_2, \ldots, n_r)$, where n_i is the number of primes one must use to determine the sign of the i'th term of the sequence at X. Also compute NP, the number of additional images which must be constructed.]

(18) LNPRMS ← 0; DUM ← DEGSEQ; DUM2 ← NORMS; MAXNP ← 0.

(19) ADV(DEG,DUM); ADV(N,DUM2); NP ← [(N+DEG*ECEIL)/PEXP] + 1; LNPRMS ← PFA(NP,LNPRMS); if MAXNP < NP, MAXNP ← NP; if DUM ≠ 0, go to (19); LNPRMS ← INV(LNPRMS); NP ← MAXNP - LENGTH(PLIST2).

[Jump to the proper step in the algorithm if no additional images are needed.]

(20) If NP ≤ 0, go to (3, 4, 7), J.

[Compute an image of the sequence over GF(p) if there are primes remaining and ϕ_p(LCP1) ≠ 0 and ϕ_p(LCP2) ≠ 0. Reject the prime if it does not produce a maximal degree sequence; otherwise save the sequence and the prime and decrease the number, NP, of additional images needed by 1.]

(21) If PLIST1 = 0, go to (26); ADV(P,PLIST1): if CMOD(P,LCP1) = 0, go to (21); if CMOD(P,LCP2) = 0, go to (21); CP1 ← CPMOD(P,P1); CP2 ← CPMOD(P,P2); S ← CNPRS(P,CP1,CP2); erase CP1, CP2; if LDEGSQ = LENGTH(S), go to (23).

(22) Erase S; go to (21).

(23) DUM ← S; DUM2 ← DEGSEQ.

(24) ADV(DUM1,DUM); ADV(DUM3,DUM2); if FIRST(DUM1) ≠ DUM3, go to (22); if DUM ≠ 0, go to (24); DUM1 ← SEQ; DUM2 ← SEQ.

(25) DECAP(PI,S); ADV(DUM,DUM2); DUM ← PFL(PI,DUM); ALTER(DUM,DUM1); DUM1 ← DUM2; if S ≠ 0, go to (25); PLIST2 ← PFA(P,PLIST2); NP ← NP - 1; go to (20).

(26) Print 'INSUFFICIENT PRIMES,RROOTS'; stop.

<u>Computing Time</u>: Let m = degree(Q), d = norm(Q), let $XX = (\frac{a_1}{b_1}, \frac{a_2}{b_2}]$ be an interval of length h containing r real zeros, $e = \max\{|a_1|, |a_2|, b_1, b_2\}$, and if $r \geq 2$, let $\lambda = \min\{|\alpha-\beta|: Q(\alpha) = 0, Q(\beta) = 0, \alpha \neq \beta$, and $\alpha, \beta \in XX\}$. Then if F = 1, $r \geq 2$, and YY = 0 or YY $\neq$ 0, the maximum computing time is:

$$O(m^4(\ln md) + rm^3(\ln \frac{h}{\lambda})(\ln (\frac{eh}{\lambda})\ md)^2).$$

If F = 0 or F = 1, $r \leq 1$, and YY = 0 or YY $\neq$ 0, then the maximum computing time is:

$$O(m^4(\ln md) + m^3(\ln emd)^2).$$

<u>Algorithm</u> S = SEVAL(LQ,LP,NP,PT)

<u>Input</u>: LQ = $(Q_1, Q_2, \ldots, Q_k)$, where $Q_i = \phi^*_{p_i}(Q)$ and degree (Q_1) = degree (Q_i), Q being a univariate polynomial over the integers; LP = $(p_1, p_2, \ldots, p_k)$, a list of the corresponding primes; NP, a FORTRAN integer equal to or greater than the number of primes SEVAL must process to determine the sign of Q(PT), NP $\leq$ k; and PT, a rational number whose denominator is not divisible by any of the primes $p_1, p_2, \ldots, p_{NP}$.

<u>Output</u>: S, a FORTRAN integer equal to the sign of Q(PT).

(1) LPOLYS ← LQ; LPRIME ← LP; NPRIME ← NP; POINT ← PT; DEG ← FIRST (FIRST(LPOLYS)); QQ ← PFA(1,0); BB ← 0; if POINT = 0, go to (2); PTN ← FIRST(POINT); PTD ← FIRST(TAIL(POINT)).

(2) Do thru (4) for I = 1, . . . , NPRIME: ADV(P,LPRIME), ADV(POLY,LPOLYS); if POINT $\neq$ 0, go to (3); R ← CPEVAL(P,POLY,0); go to (4).

(3) $N \leftarrow$ CMOD(P,PTN); $D \leftarrow$ CMOD(P,PTD); $X \leftarrow$ CPROD(P,N,CRECIP(P,D));

$R \leftarrow$ CPROD(P,CPOWER(P,D,DEG), CPEVAL(P,POLY,X)).

(4) $C \leftarrow$ CGARN(QQ,BB,P,R); erase BB; $BB \leftarrow C$; $P \leftarrow$ PFA(P,O); $DUM \leftarrow QQ$;

$QQ \leftarrow$ IPROD(QQ,P); erase DUM, P.

(5) $S \leftarrow$ ISIGNL(BB); erase BB, QQ; return.

Computing Time: $O(m^2(\ln e)^2 + m(\ln e)(\ln d) + (\ln d)^2)$, where degree (Q) $\le m$, Q being the polynomial being evaluated; norm (Q) $\le d$; $LQ = (Q_1, \ldots, Q_k)$, where $Q_i = \emptyset_{pi}(Q)$ and degree (Q_i) = degree (Q); $LP = (p_1, \ldots, p_k)$; $PT = \frac{a}{b}$, $b > 0$, $e = \max\{|a|, b\}$; $NP \le k$ and $(\min\{p_1, p_2, \ldots, p_k\})^{NP-1} < 2 \cdot d \cdot e^m$.

Algorithm SS = STURM(XXX,YYY)

Input: XXX and YYY, two non-zero univariate polynomials over the integers, degree (XXX) > 0, degree (XXX) $\ge$ degree (YYY); PRIME, a list of primes; PEXP = $\lfloor \log_2(\min\{\text{primes on list PRIME}\}) \rfloor$.

Output: Let B_1 = XXX, B_2 = YYY, $\ldots, B_r$ be a quasi-negative subresultant p.r.s. over the integers. Let $B_1^{(i)}, B_2^{(i)}, \ldots, B_r^{(i)}$ be a quasi-negative subresultant p.r.s. over $GF(p_i)$ for which degree $(B_j^{(i)})$ = degree (B_j). Let $C_j = (B_j^{(1)}, B_j^{(2)}, \ldots, B_j^{(k)})$, $1 \le j \le r$. Then SEQ = $(C_1, C_2, \ldots, C_r)$; PLIST2 = $(p_1, p_2, \ldots, p_k)$, where $\Pi_{i=1}^{k} p_i \ge d^{m+n}$, where $B_1 \varepsilon U(d,m)$ and $B_2 \varepsilon U(d,n)$; SIGNS = $(s_1, s_2, \ldots, s_r)$, where the p.r.s. $s_1 B_1, s_2 B_2, \ldots, s_r B_r$ is a negative p.r.s. for XXX and YYY over the integers; NORMS = $(n_1, n_2, \ldots, n_r)$, where n_i is equal to or greater than the logarithm to the base 2 of the subresultant bound of the coefficients of B_i; PLIST1 is a final segment of the list PRIME consisting of all primes which were not discarded by STURM as not producing maximal degree sequences nor placed on the list PLIST2; DEGSEQ = $(d_1, d_2, \ldots, d_r)$, where d_i = degree (B_i). And SS = (SEQ,PLIST2,SIGNS,NORMS,PLIST1, DEGSEQ).

[Compute X = $\max\{\lceil \log_2(\text{norm}(P))\rceil, \lceil \log_2(\text{norm}(Q))\rceil\}$.]

(1) P ← XXX; Q ← YYY; DEGP ← PDEG(P); DEGQ ← PDEG(Q); NORMP ← PNORMF(P); NORMQ ← PNORMF(Q); LDCP ← FIRST(TAIL(P)); LDCQ ← FIRST(TAIL(Q)); ELPOF2(NORMP,DUM,R1); ELPOF2(NORMQ,DUM,S1); erase NORMP,NORMQ; if R1 ≥ S1, go to (2); X ← S1; go to (3).

(2) X ← R1.

[Compute NPRIME, the number of primes that must produce identical degree sequences to determine the degree sequence over the integers.]

(3) NPRIME ← [(X*(DEGP+DEGQ))/PEXP] + 1; PLIST1 ← PRIME; DEGSEQ ← 0; PCOUNT ← 0; PLIST2 ← 0; SEQ2 ← 0.

[Stop if list of primes is exhausted.]

(4) If PLIST1 ≠ 0, go to (5); print 'INSUFFICIENT PRIMES, STURM'; stop.

[If ϕ_{p_i} (ldcf(P)) or ϕ_{p_i} (ldcf(Q)) = 0, reject the prime; otherwise use CNPRS to compute a p.r.s., S, over GF(p_i).]

(5) ADV(GFP,PLIST1); if CMOD(GFP,LDCP) = 0 or CMOD(GFP,LDCQ) = 0, go to (4); P1 ← CPMOD(GFP,P); P2 ← CPMOD(GFP,Q); S ← CNPRS(GFP,P1,P2); erase P1,P2; DUM ← S; D ← 0.

[If the degree sequence of S is equal to the current maximal degree sequence, DEGSEQ, prefix S onto the list SEQ2 and GFP onto the list PLIST2; if the degree sequence is less than the current maximal degree sequence, erase S and reject the prime; if it is greater, erase the list of sequences with the current maximal degree sequence (the list SEQ2) and the list of associated primes (the list PLIST2) and set DEGSEQ to the new maximal degree sequence and start a new list SEQ2 with S as its only element and a new list PLIST2 with GFP as its only element.]

(6) ADV(DUM1,DUM); D ← PFA(FIRST(DUM1),D); if DUM $\neq$ 0, go to (6);
D ← INV(D); if DEGSEQ = 0, go to (10); DUM1 ← DEGSEQ; DUM2 ← D.

(7) ADV(DUM3,DUM1); ADV(DUM4,DUM2); if DUM3-DUM4, go to (10), (8), (9).

(8) If DUM1 $\neq$ 0 and DUM2 $\neq$ 0, go to (7);
if DUM1 = 0 and DUM2 = 0, go to (11);
if DUM1 = 0, go to (10).

(9) Erase D, S; go to (4).

(10) Erase DEGSEQ,PLIST2,SEQ2; DEGSEQ ← D; PLIST2 ← PFA(GFP,0);
SEQ ← PFL(S,0); PCOUNT ← 1; go to (12).

(11) SEQ2 ← PFL(S,SEQ2); PLIST2 ← PFA(GFP,PLIST2); PCOUNT ← PCOUNT + 1;
erase D.

[If NPRIME primes have not yet produced identical degree sequences, jump to process another prime. If they have, construct from SEQ2 a list SEQ whose i'th element is a list of NPRIME images of the i'th term of the p.r.s. $B_1, B_2, \ldots, B_r$.]

(12) If PCOUNT < NPRIME, go to (4); L1 ← LENGTH(DEGSEQ); SEQ2 ← INV(SEQ2);
SEQ ← 0.

(13) DUM1 ← SEQ2; DUM2 ← SEQ2; LT ← 0.

(14) ADV(S,DUM2); DECAP(PI,S); LT ← PFL(PI,LT); ALTER(S,DUM1); DUM1 ← DUM2; if DUM2 ≠ 0, go to (14); SEQ ← PFL(LT,SEQ); if S ≠ 0, go to (13); erase SEQ2; SEQ ← INV(SEQ).

[Compute a list NORMS = $(n_1, n_2, \ldots, n_r)$ such that $n_i \geq \log_2$ {subresultant bound on the coefficients of B_i}.]

(15) DUM ← DEGSEQ; NORMS ← PFA(S1,PFA(R1,0)); L ← L1 - 2; if L = 0, go to (16); ADV(DUM1,DUM); do for I = 1,...,L: [ADV(DUM1,DUM); NORMS ← PFA(R1*(DEGQ-DUM1+1) + S1*(DEGP-DUM+1), NORMS)].

(16) NORMS ← INV(NORMS).

[Make up the list BSIGNS = $(\beta_2, \ldots, \beta_{r-1})$, where β_2 = sign (ldcf(Q)) = sign (ldcf(B_2)) and, for $3 \leq k \leq r - 1$, $\beta_k = 1$ if δ_{k-1} and δ_k are both odd, and β_k = sign (ldcf(B_k)) otherwise, where for $k \geq 3$ the sign (ldcf(B_k)) is obtained using the Chinese Remainder Theorem.]

(17) SIGNS ← PFA(1,PFA(1,0); if L1 = 2, go to (24); BSIGNS ← PFA (PSIGN(Q),0); if L1 = 3, go to (21); DUM ← NORMS; DUM1 ← SEQ; DUM2 ← DEGSEQ; do for I = 1,2: [ADV(NORMI,DUM); ADV(BI,DUM1); ADV(NI,DUM2)]; ADV(NIP1,DUM2).

(18) NIM1 ← NI; NI ← NIP1; ADV(NIP1,DUM2); ADV(NORMI,DUM); ADV(BI,DUM1); if (-1) ** (NIM1-NI) = 1 or (-1) ** (NI-NIP1) = 1, go to (19); SIGN ← 1; go to (20).

(19) NP ← [NORMI/PEXP] + 1; DUM3 ← PLIST2; QQ ← PFA(1,0); BB ← 0; do for J = 1,...,NP: [ADV(GFP,DUM3); ADV(COEF,BI); COEF ← FIRST(TAIL(COEF)); C ← CGARN(QQ,BB,GFP,COEF); erase BB; BB ← C; GFP ← PFA(GFP,0); DUM4 ← QQ; QQ ← IPROD(QQ,GFP); erase DUM4,GFP]; SIGN ← ISIGNL(BB); erase BB, QQ.

(20) BSIGNS ← PFA(SIGN,BSIGNS); if DUM2 $\neq$ 0, go to (18).

[Construct the list SIGNS = $(s_1, s_2, \ldots, s_r)$, where the s_i are the sign correction factors so that $s_1B_1, s_2B_2, \ldots, s_rB_r$ is a negative p.r.s.]

(21) DUM ← DEGSEQ; ADV(NKM1,DUM); ADV(NK,DUM); DECAP(BETK,BSIGNS);
SKP1 ← BETK ** (NKM1-NK+1); SK ← 1.

(22) SIGNS ← PFA(SKP1,SIGNS); if BSIGNS = 0, go to (23); SKM1 ← SK; SK ← SKP1;
BETKM1 ← BETK; DECAP(BETK,BSIGNS); NKM1 ← NK; ADV(NK,DUM);
SKP1 ← SK * (SKM1*BETKM1*SK*BETK) ** (NKM1-NK+1); go to (22).

(23) SIGNS ← INV(SIGNS).

[Set up output list.]

(24) STURM ← PFL(NORMS,PFL(BORROW(PLIST1), PFL(DEGSEQ,0))); STURM ←
PFL(SEQ,PFL(PLIST2,PFL(SIGNS,STURM))); return.

Computing Time: $O(m^4(\ln d) + m^3(\ln d)^2)$, where m bounds the degrees of XXX and YYY, and d bounds their norm.

Algorithm V = VAR(SS,LP,LNP,LSCFS,E)

Input: Let $B_1, B_2, \ldots, B_r$ be a quasi-negative subresultant p.r.s. Let $B_1^{(i)}, B_2^{(i)}, \ldots, B_r^{(i)}$ be an image of $B_1, B_2, \ldots, B_r$ in $GF(p_i)$ such that degree $(B_j^{(i)})$ = degree (B_j). Let $C_i = (B_i^{(1)}, B_i^{(2)}, \ldots, B_i^{(k)})$. Then SS = $(C_1, C_2, \ldots, C_r)$. LP = $(p_1, p_2, \ldots, p_k)$ is a list of the corresponding primes. LNP = $(n_1, n_2, \ldots, n_r)$ where n_i is the number of primes SEVAL must use to determine the sign of $B_i(E)$, max $\{n_1, n_2, \ldots, n_r\} \leq k$. LSCFS = $(s_1, s_2, \ldots, s_r)$ is a list of sign correction factors such that $s_1B_1, s_2B_2, \ldots, s_rB_r$ is a Sturm sequence.

E is a rational number whose denominator is not divisible by any of the primes $p_1, \ldots, p_k$.

Output: V, the number of variations in sign of the sequence $s_1B_1(E), s_2B_2(E), \ldots, s_rB_r(E)$.

(1) SEQ ← SS; LPRIMS ← LP; LNPRMS ← LNP; LSIGNS ← LSCFS; PT ← E; V ← 0; LS ← 0.

(2) ADV(P,SEQ); ADV(NPRIME,LNPRMS); ADV(PSIGN,LSIGNS); S ← SEVAL (P,LPRIMS,NPRIME,PT) * PSIGN; if LS = 0, go to (3); if S = 0 or S = LS, go to (4); V ← V + 1.

(3) LS ← S.

(4) If SEQ ≠ 0, go to (2); return.

Computing Time: $O(m^3(\ln de)^2)$, where $E = \frac{a}{b}$, $b > 0$, $e = \max\{|a|, b\}$, m = degree (B_1), and d is a bound on the norms of B_1 and B_2.

2.3 Miscellaneous

Algorithm INS = ANALRR(Q,Y)

Input Q, a square-free, positive, univariate polynomial of positive degree with integer coefficients; and Y, a rational number equal to or greater than zero.

Output: If Q has no real zeros, INS is set to zero. If Q has any real zeros and Y = 0, INS is a list $(I_1, \ldots, I_r)$, $I_1 \le I_2 \le \ldots \le I_r$, where each I_i, $1 \le i \le r$ contains exactly one real zero of Q and every real zero of Q is contained in some I_i. If Y ≠ 0, the length of each of the I_i is equal to or less than Y.

(1) $P1 \leftarrow Q$; $EPS \leftarrow Y$; $R \leftarrow RBOUND(P1)$; $L \leftarrow RDIF(0,R)$; $INTER \leftarrow PFL(L, PFL(R,0))$; $IN \leftarrow ISOLAT(P1,INTER)$; erase INTER; if $IN \neq 0$ and $EPS \neq 0$, go to (2); $INS \leftarrow IN$; return.

(2) $INS \leftarrow 0$.

(3) DECAP(I,IN); $INS \leftarrow PFL(REFINE(P1,I,EPS), INS)$; erase I; if $IN \neq 0$, go to (3); $INS \leftarrow INV(INS)$; return.

Computing Time: $O(m^{10}(\ln d)^3 + m^3(\ln \frac{d}{y})(m^2(\ln d) + (\ln \frac{1}{y}))^2)$, if $0 < y \leq 1$, and $O(m^{10}(\ln d)^3)$, if $1 < y$ or $y = 0$, where m = degree (Q), and d = norm(Q)

Algorithm ELPOF2(X,EFLR,ECEIL)

Input: X, an L-integer.

Output: If $X = 0$, $EFLR = ECEIL = 0$; if $X \neq 0$, EFLR, a FORTRAN integer, equal to $\lfloor \log_2 |X| \rfloor$; and ECEIL, a FORTRAN integer, equal to $\lceil \log_2 |X| \rceil$.

(1) $V \leftarrow X$; $EFLR \leftarrow 0$; $ECEIL \leftarrow 0$; if $V = 0$, return; $Q \leftarrow IABSL(V)$; $TWO \leftarrow PFA(2,0)$; $RP \leftarrow 0$.

(2) $Z \leftarrow IQRS(Q,TWO)$; erase Q; DECAP(Q,Z); DECAP(R,Z); if $R = 0$, go to (3); $RP \leftarrow RP + 1$; erase R.

(3) If $Q = 0$, go to (4); $EFLR \leftarrow EFLR + 1$; go to (2).

(4) If $RP > 1$, $ECEIL \leftarrow 1$; $ECEIL \leftarrow EFLR + ECEIL$; erase TWO; return.

Computing Time: $O(1)$, if $X = 0$ and $O((\ln |X|)^2)$ if $X \neq 0$.

Algorithm IPRINT(UNIT,Y)

Input: Y, an interval whose endpoints are either zero or rational numbers with powers of 2 as denominators; and UNIT, an I/O unit number.

Output: IPRINT prints the left endpoint of interval Y on I/O unit UNIT as an integer if it is zero or has a denominator of 1 and as a decimal number otherwise. IPRINT then skips to the next record and repeats the process with the right endpoint.

(1) IN ← Y; FIVE ← PFA(5,0).

(2) ADV(L,IN); if L $\neq$ 0, go to (3); IWRITE(L,UNIT); go to (10).

(3) ADV(A,L); ADV(B,L); ELPOF2(B,K,DUM); if K $\neq$ 0, go to (4); IWRITE(A,UNIT); go to (10).

(4) DUM ← IQR(A,B); DECAP(Q,DUM); DECAP(R,DUM); Z ← IBTOD(Q); erase Q; DECAP(DUM,Z); FACT ← PPOWER(FIVE,K); F ← IPROD(R,FACT); erase R,FACT; FL ← IBTOD(F); erase F; DECAP(SIGN,FL); Z ← PFA(SIGN,Z); DUM ← K-LENGTH(FL); if DUM = 0, go to (5); do for I = 1,...,DUM: FL ← PFA(0,FL).

(5) FL ← PFA(43,FL); FL ← CONC(Z,FL).

(6) Do for I = 1,...,72: if FL = 0, go to (8); DECAP(RECORD(I),FL).

(7) WRITE(UNIT,RECORD); go to (6).

(8) Do for J = I,...,72: RECORD(J) ← 44.

(9) WRITE(UNIT,RECORD)

(10) If IN $\neq$ 0, go to (2); erase FIVE; return.

Algorithm N = PNORMF(X)

Input: X, a multivariate polynomial over the integers.

Output: N, the norm of X.

(1) $P \leftarrow X$, $N \leftarrow 0$; if $P = 0$, return; if TYPE(P) = 1, go to (2);
$N \leftarrow$ IABSL(P); return.

(2) $L \leftarrow 0$.

(3) $P \leftarrow$ TAIL(P); if $P = 0$, go to (5); ADV(COEF,P); if TYPE(COEF) = 1, go to (4); $Z \leftarrow$ IABSL(COEF); DUM $\leftarrow$ N; $N \leftarrow$ ISUM(N,Z); erase DUM, Z; go to (3).

(4) $L \leftarrow$ PFA(P,L); $P \leftarrow$ COEF; go to (3).

(5) If $L = 0$, return; DECAP(P,L); go to (3).

Computing Time: $O(m_1 m_2 \ldots m_r (\ln d))$, where m_i is the degree of X in the i'th variable and d is the norm of X.

Algorithm X = PPOWER(P,I)

Input: P, a multivariate polynomial over the integers; and I, a FORTRAN integer.

Output: If $I < 0$ or $P = 0$, $X = 0$; otherwise, $X = P^I$.

(1) $Z \leftarrow P$; $N \leftarrow I$; $X \leftarrow 0$; if $N < 0$ or $Z = 0$, return; if $N \neq 0$, go to (3);
$L \leftarrow$ PVLIST(Z); $X \leftarrow$ PFA(1,0).

(2) If $L = 0$, return; DECAP(Z1,L); $X \leftarrow$ PFL(Z1,PFL(X,PFA(0,0))); go to (2).

(3) $X \leftarrow$ BORROW(Z); if $N = 1$, return; $N \leftarrow N - 1$; do for $J = 1, \ldots, N$:
[DUM $\leftarrow$ X; $X \leftarrow$ PPROD(X,Z); erase DUM]; return.

Computing Time: If $I < 0$ or $P = 0$, then the maximum computing time is $O(1)$. If P is an L-integer bounded in magnitude by d, then the maximum computing time is $O(1)$ if $I = 0$ or 1, and $O(I^2 (\ln d)^2)$ if $I > 1$. If P is a polynomial in r variables of norm d and degree m_i in the ith variable, then the maximum computing time is $O(r)$ if $I = 0$, $O(1)$ if $I = 1$, and $O(m_1^2 \ldots m_r^2 \, I^{r+2} (\ln d)^2$ if $I > 1$.

Algorithm B = RBOUND(Y)

Input: Y, a non-zero univariate polynomial over the integers.

Output: B, a rational number such that all the zeros of Y lie within a circle about the origin of radius B.

(1) P ← TAIL(Y); ADV(DEN,P); DEN ← IABSL(DEN); ELPOF2(DEN,DEN2,DUM); ADV(N,P); EXP ← 1.

(2) If P = 0, go to (4); ADV(COEF,P); COEF ← IABSL(COEF); ADV(DEG,P); if ICOMP(DEN,COEF) ≥ 0, go to (3); ELPOF2(COEF,DUM,COEF2); EXPP ← [(COEF2-DEN2)/(N-DEG)] + 2; if EXP < EXPP, EXP ← EXPP.

(3) Erase COEF; go to (2).

(4) TWO ← PFA(2,0); X ← PPOWER(TWO,EXP); B ← RPOLY(X); erase TWO, DEN, X; return.

Computing Time: $O(m(\ln d)^2)$, where m = degree (Y) and d = norm(Y).

Algorithm I = RCOMP(X,Y)

Input: X and Y, two rational numbers.

Output: I, the FORTRAN integer -1 if X < Y, the FORTRAN integer 0 if X = Y, the FORTRAN integer 1 if X > Y.

(1) L ← X; R ← Y; if L ≠ 0, go to (3); if R ≠ 0, go to (2); I ← 0; return.

(2) I ← -ISIGNL(FIRST(R)); return.

(3) If R ≠ 0, go to (4); I ← ISIGNL(FIRST(L)); return.

(4) LN ← FIRST(L); LD ← FIRST(TAIL(L)); RN ← FIRST(R); RD ← FIRST(TAIL(R)); RNLD ← IPROD(RN,LD); RDLN ← IPROD(RD,LN); I ← ICOMP(RDLN,RNLD); erase RNLD, RDLN; return.

Computing Time: $O(1)$, if $X = Y = 0$; $O(\ln |a|)$, if only one of X and Y is non-zero and equal to $\frac{a}{b}$, $b > 0$; and $O((\ln e)^2)$, if $X = \frac{a}{b}$, $b > 0$, $Y = \frac{c}{d}$, $d > 0$, $e = \max\{|a|, |c|, b, d\}$.

Algorithm $X = \mathrm{SQFREE}(P)$

Input: P, a univariate polynomial with integer coefficients.

Output: if $P = 0$, then $X = 0$; otherwise X is the primitive part of the unique positive greatest square-free divisor of P.

(1) $X \leftarrow P$; if $X = 0$, return; DUM $\leftarrow$ PPP(X); DERV $\leftarrow$ PDERIV(DUM,FIRST(DUM)); GCD $\leftarrow$ PGCD(DUM,DERV); X $\leftarrow$ PQ(DUM,GCD); erase DUM, GCD, DERV; return.

Computing Time: $O(mL^2(\ln md)^2 + L^2(m-L+1)^2(\ln d)^2)$, where P is a univariate polynomial of degree m, norm d, and which has L distinct roots.

3. EMPIRICAL RESULTS

We now present four tables of empirical results of running the various algorithms. The numbers under the name of an algorithm are the times in milliseconds of UNIVAC 1108 execution time for applying the algorithm to the given polynomial. The times under ISOLT and ISOLAT are the times to isolate all the real zeros of the given polynomial P in the interval (-B,B], where B = RBOUND(P). The epsilon given in the table is the length of the refining interval, i.e. the error bound on the zeros. The random polynomials were chosen of odd degree to insure that they had at least one real zero.

Table 2.14.1: Empirical Results on Ten Random Polynomials Belonging to $U(11, 2^3 - 1)$

$\varepsilon = 10^{-9}$

No.	No. of Roots	NRTS (integer)	NROOTS (congruence)	ISOLT (integer)	ISOLAT (congruence)	REFINE (integer)	CREFIN (congruence)	ANALR (integer)	CANALR (congruence)	ANALRR (combination)
1	1	845	637	802	626	1017	1473	1819	2099	1547
2	3	677	531	840	776	2968	4026	3808	4802	3716
3	1	700	647	726	612	1038	1340	1764	1952	1651
4	3	787	637	960	967	3373	4611	4333	5578	4030
5	3	705	540	863	786	2979	4029	3842	4815	3828
6	1	610	474	610	498	1033	1420	1643	1918	1443
7	1	708	539	703	538	1018	1430	1721	1968	1569
8	3	743	605	902	858	3043	3964	3945	4822	3898
9	1	692	550	700	631	1126	1809	1826	2440	1835
10	3	805	703	1097	1082	2352	3317	3449	4399	3424

Table 2.14.2: Empirical Results on Selected Chebyshev Polynomials

$\varepsilon = 10^{-15}$

No. of Roots	NRTS (integer)	NROOTS (congruence)	ISOLT (integer)	ISOLAT (congruence)	REFINE (integer)	CREFIN (congruence)	ANALR (integer)	CANALR (congruence)	ANALRR (combination)
1	8	24	8	23	6	9	14	32	28
2	21	32	22	37	1816	2816	1838	2853	1918
3	36	42	56	78	1936	3033	1992	3111	1889
4	55	67	100	142	4327	7072	4427	7214	4647
5	84	107	265	379	4390	8014	4655	8393	4801
6	110	165	316	571	7975	13754	8291	14325	8021
7	143	257	428	931	8017	15044	8445	15975	8982
8	201	462	631	1726	12586	22632	13217	24358	14290
9	229	580	737	2425	13047	25316	13784	27714	15388
10	274	841	924	3648	18737	35564	19661	39212	22284
15	736	2683	3308	17393	43041	81977	46349	99370	57829
20	1189	6556	6573	50208	96516	177733	103089	227941	143737
25	2129	*	12823	*	158387	*	171210	*	*

*Could not be run due to insufficient computer memory.

Table 2.14.3: Empirical Results on First Ten Legendre Polynomials

$\varepsilon = 10^{-15}$

No. of Roots	NRTS (integer)	NROOTS (congruence)	ISOLT (integer)	ISOLAT (congruence)	REFINE (integer)	CREFIN (congruence)	ANALR (integer)	CANALR (congruence)	ANALRR (combination)
1	9	23	10	26	6	8	16	34	29
2	22	34	24	39	1766	2816	1790	2855	1883
3	39	47	56	79	1881	3227	1937	3306	1892
4	59	65	103	136	4345	7367	4448	7503	4404
5	88	102	219	309	4488	8186	4707	8495	4820
6	113	130	288	397	7723	13996	8011	14393	8100
7	155	224	460	791	8024	15437	8484	16228	8805
8	186	263	528	928	12310	23173	12838	24101	12794
9	229	447	760	1695	12844	25250	13604	26945	14678
10	276	573	1072	2675	18635	35988	19707	38663	21323

Table 2.14.4: Empirical Results on Three Random Polynomials Belonging to $U(21, 2^{32} - 1)$

$\varepsilon = 10^{-15}$

No.	No. of Roots	NRTS (integer)	NROOTS (congruence)	ISOLT (integer)	ISOLAT (congruence)	REFINE (integer)	CREFIN (congruence)	ANALR (integer)	CANALR (congruence)	ANALRR (combination)
1	1	62457	17681	62742	17835	7743	10049	70485	27884	24144
2	1	61998	18094	61966	18068	5716	8939	67682	27007	23749
3	1	61455	17716	61435	17359	6059	9322	67494	26681	23416

By examining the tables one can conclude that neither integer nor modular arithmetic techniques are faster in all cases for a given computation. For instance, to compute the number of zeros of the Chebyshev polynomials requires more time using modular arithmetic than integer arithmetic. However, the opposite is true with the random 21'st degree polynomials. Thus it seems that only experience will tell which algorithms to use on which polynomials.

In all the empirical tests the largest value returned by RBOUND was 16 with the typical value being 4, even though the norms of some of the polynomials are quite large.

In the empirical results of isolating and refining the zeros of both the Chebyshev and Legendre polynomials, the computing time for the polynomial of degree n and the polynomial of degree $n + 1$ is approximately the same for even n. This is because $X = 0$ is a zero of all the odd degree polynomials and this zero is found exactly on the first bisection of the interval $(-B,B]$.

An interesting point to note about ANALR, ANALRR, and CANALR is that they will find all the integer zeros of a polynomial exactly if the refining epsilon is set at 1 or less. This is because the initial interval $(-B,B]$ has integer endpoints and a length which is a power of 2. This is one reason for having RBOUND return a power of 2.

4. TEST PROGRAM

The following test program calculates the number of real zeros of the first five Chebyshev polynomials using both integer arithmetic (NRTS) and modular arithmetic (NROOTS). It also isolates and refines the real zeros of the polynomials to eight decimal places using both integer arithmetic (ANALR) and modular arithmetic (CANALR). Notice that corresponding to each real zerothere is an interval printed by IPRINT which contains the root, and hence the endpoints of the intervals have more decimal places than they are accurate to. This is due to the algorithm used by IPRINT to convert from rational to decimal notation.

If this test program is to function correctly on another system, care must be taken to use the correct values for BETA, CONS, IN, and OUT, rather than the ones given in the program. CONS is the starting value used in generating the list of primes, PRIME.

```
      INTEGER BETA,CONS,DUM,EPS,I,IN,INS,N1,N2,OUT,P,PA,PEXP,PRIME
      INTEGER SPACE,SYMLST,X
      INTEGER ANALR,CANALR,GENPR,NROOTS,NRTS,PFA,PREAD,RREAD,SQFREE
      COMMON /TR2/SYMLST
      COMMON /TR3/BETA
      COMMON /TR4/PRIME,PEXP
      DIMENSION SPACE(10000),PA(1000)
      CALL BEGIN(SPACE,10000)
      BETA=2**33
      IN=5
      OUT=6
      CONS=400000001
      PRIME=GENPR(PA,1000,CONS)
      CONS=PFA(CONS,0)
      CALL ELPOF2(CONS,PEXP,DUM)
      CALL ERLA(CONS)
      SYMLST=0
      PRINT 1
1     FORMAT(5H1EPS=)
      EPS=RREAD(IN)
      CALL RWRITE(OUT,EPS)
2     X=PREAD(IN)
      IF (X .EQ. -1) STOP
      P=SQFREE(X)
      CALL PERASE(X)
      PRINT 3
3     FORMAT(3H0P=)
      CALL PWRITE(OUT,P)
      N1=NRTS(P)
      N2=NROOTS(P)
      PRINT 4,N1,N2
4     FORMAT(4H0N1=,I2,20X,3HN2=,I2)
      INS=ANALR(P,EPS)
      PRINT 5
5     FORMAT(23H0OUTPUT OF ANALR(P,EPS))
6     IF (INS .EQ. 0) GO TO 8
      PRINT 7
7     FORMAT(1X)
      CALL DECAP(I,INS)
      CALL IPRINT(OUT,I)
      CALL ERASE(I)
      GO TO 6
8     PRINT 9
9     FORMAT(24H0OUTPUT OF CANALR(P,EPS))
      INS=CANALR(P,EPS)
10    IF (INS .EQ. 0) GO TO 11
      PRINT 7
      CALL DECAP(I,INS)
```

```
      CALL IPRINT(OUT,I)
      CALL ERASE(I)
      GO TO 10
11    CALL PERASE(P)
      GO TO 2
      END
```

INPUT TO TEST PROGRAM

```
+1 / +1000000000
(+1X**1)
(+2X**2-1X**0)
(+4X**3-3X**1)
(+8X**4-8X**2+1X**0)
(+16X**5-20X**3+5X**1)
```

OUTPUT OF TEST PROGRAM

```
EPS=
+1    /    +1000000000

P=
(+1X**1)

N1= 1                         N2= 1

OUTPUT OF ANALR(P,EPS)

+0
+0

OUTPUT OF CANALR(P,EPS)

+0
+0

P=
(+2X**2-1X**0)

N1= 2                         N2= 2

OUTPUT OF ANALR(P,EPS)

-0.707106781192123889923095703125
-0.7071067802608013153076171875

+0.7071067802608013153076171875
+0.707106781192123889923095703125

OUTPUT OF CANALR(P,EPS)
```

```
-0.70710678119212388992309570312 5
-0.707106780260801315307617187 5

+0.707106780260801315307617187 5
+0.70710678119212388992309570312 5

P=
(+4X**3-3X**1)

N1= 3                      N2= 3

OUTPUT OF ANALR(P,EPS)

-0.8660254040732979774475097656 25
-0.866025403141975402832031 25

+0
+0

+0.866025403141975402832031 25
+0.8660254040732979774475097656 25

OUTPUT OF CANALR(P,EPS)

-0.866025403141975402832031 25

+0
+0

+0.866025403141975402832031 25
+0.8660254040732979774475097656 25

P=
(+8X**4-8X**2+1X**0)

N1= 4                      N2= 4

OUTPUT OF ANALR(P,EPS)

-0.92387953307479619979858398437 5
-0.923879532143473625183105468 75

-0.38268343266099691390991210937 5
-0.3826834317296743392944335937 5

+0.3826834317296743392944335937 5
+0.38268343266099691390991210937 5

+0.923879532143473625183105468 75
+0.92387953307479619979858398437 5
```

```
OUTPUT OF CANALR(P,EPS)

-0.923879533074796199798583984375
-0.9238795321434736251831054 6875

-0.382683432660996913909912109375
-0.38268343172967433929443359375

+0.38268343172967433929443359375
+0.382683432660996913909912109375

+0.92387953214347362518310546875
+0.923879533074796199798583984375

P=
(+16X**5-20X**3+5X**1)

N1= 5                         N2= 5

OUTPUT OF ANALR(P,EPS)

-0.951056516729295253753662109375
-0.95105651579797267913818359375

-0.587785252369940280914306640625
-0.5877852514386177062988 28125

+0
+0

+0.58778525143861770629882 8125
+0.587785252369940280914306640625

+0.95105651579797267913818359375
+0.951056516729295253753662109375

OUTPUT OF CANALR(P,EPS)

-0.951056516729295253753662109375
-0.95105651579797267913818359375

-0.587785252369940280914306640625
-0.58778525143861770629882 8125

+0
+0

+0.58778525143861770629882 8125
+0.587785252369940280914306640625

+0.95105651579797267913818359375
+0.951056516729295253753662109375
```

5. FORTRAN SUBPROGRAMS

```
      INTEGER FUNCTION ANALR(Q,Y)
      INTEGER EPS,I,IN,INTER,L,P1,Q,R,Y
      INTEGER INV,ISCLT,PFL,RBOUND,RDIF,REFINE
      P1=Q
      EPS=Y
      R=RBOUND(P1)
      L=RDIF(0,R)
      INTER=PFL(L,PFL(R,0))
      IN=ISOLT(P1,INTER)
      CALL ERASE(INTER)
      IF (IN .NE. 0 .AND. EPS .NE. 0) GO TO 1
      ANALR=IN
      RETURN
1     ANALR=0
2     CALL DECAP(I,IN)
      ANALR=PFL(REFINE(P1,I,EPS),ANALR)
      CALL ERASE(I)
      IF (IN .NE. 0) GO TO 2
      ANALR=INV(ANALR)
      RETURN
      END
```

```
      INTEGER FUNCTION ANALRR(Q,Y)
      INTEGER EPS,I,IN,INTER,L,P1,Q,R,Y
      INTEGER INV,ISOLAT,PFL,RBOUND,RDIF,REFINE
      P1=Q
      EPS=Y
      R=RBOUND(P1)
      L=RDIF(0,R)
      INTER=PFL(L,PFL(R,0))
      IN=ISOLAT(P1,INTER)
      CALL ERASE(INTER)
      IF (IN .NE. 0 .AND. EPS .NE. 0) GO TO 1
      ANALRR=IN
      RETURN
1     ANALRR=0
2     CALL DECAP(I,IN)
      ANALRR=PFL(REFINE(P1,I,EPS),ANALRR)
      CALL ERASE(I)
      IF (IN .NE. 0) GO TO 2
      ANALRR=INV(ANALRR)
      RETURN
      END
```

```
      INTEGER FUNCTION CANALR(Q,Y)
      INTEGER DEGSEQ,DUM,EPS,I,IN,INTER,L,LCP1,NORM,NORMS,P1,PLIST1
      INTEGER PLIST2,Q,R,ROOTS,SEQ,SIGNS,Y
      INTEGER CREFIN,INV,PFL,RBOUND,RDIF
      P1=Q
      EPS=Y
      R=RBOUND(P1)
      L=RDIF(0,R)
      INTER=PFL(L,PFL(R,0))
      DUM=0
      CALL RROOTS(P1,INTER,1,DUM,ROOTS,IN)
      CALL ERASE(INTER)
      IF (IN .NE. 0 .AND. EPS .NE. 0) GO TO 1
      CALL ERASE(DUM)
      CANALR=IN
      RETURN
1     CANALR=0
      CALL DECAP(SEQ,DUM)
      CALL DECAP(LCP1,SEQ)
      CALL ERASE(SEQ)
      CALL DECAP(PLIST2,DUM)
      CALL DECAP(SIGNS,DUM)
      CALL ERLA(SIGNS)
      CALL DECAP(NORMS,DUM)
      CALL DECAP(NORM,NORMS)
      CALL ERLA(NORMS)
      CALL DECAP(PLIST1,DUM)
      CALL DECAP(DEGSEQ,DUM)
      CALL ERLA(DEGSEQ)
2     CALL DECAP(I,IN)
      CANALR=PFL(CREFIN(P1,I,EPS,LCP1,PLIST2,NORM,PLIST1),CANALR)
      CALL ERASE(I)
      IF (IN .NE. 0) GO TO 2
      CANALR=INV(CANALR)
      CALL ERASE(LCP1)
      CALL ERLA(PLIST1)
      CALL ERLA(PLIST2)
      RETURN
      END


      INTEGER FUNCTION CNPRS(P,X,Y)
      INTEGER AI,AIM1,D,DI,DUM,NI,NIM1,P,P1,P2,R,X,Y
      INTEGER BORROW,CPOWER,CPREM,CPROD,CSPROD,FIRST,INV,PFL,TAIL
      P1=X
      P1=BORROW(BORROW(P1))
      P2=Y
      P2=BORROW(BORROW(P2))
      CNPRS=PFL(P2,PFL(P1.0))
      DI=1
```

```
      AIM1=1
      NIM1=FIRST(P1)
1     DUM=CPREM(P,P1,P2)
      IF (DUM .EQ. 0) GO TO 2
      R=CSPROD(P,DUM,P-1,0)
      CALL ERLA(DUM)
      AI=FIRST(TAIL(P2))
      NI=FIRST(P2)
      D=NIM1-NI-1
      DI=CPROD(P,DI,CPROD(P,CPOWER(P,AIM1,D),CPOWER(P,AI,D+2)))
      CNPRS=PFL(CSPROD(P,R,DI,0),CNPRS)
      CALL ERLA(P1)
      P1=P2
      P2=R
      AIM1=AI
      NIM1=NI
      GO TO 1
2     CALL ERLA(P1)
      CALL ERLA(P2)
      CNPRS=INV(CNPRS)
      RETURN
      END


      INTEGER FUNCTION CREFIN(Q,IN1,E,LCP1,PLIST2,NORM,PLIST1)
      INTEGER D,DEGP1,DIF,DUM,E,ECEIL,EPS,GFP,HALF,IN,IN1,J,L,LCP1,MID,N
      INTEGER NORM,NP,NPRIME,P1,PEXP,PLIST1,PLIST2,PRIME,Q,R,SMID,SR,SUM
      INTEGER X,XX
      INTEGER BORROW,CMOD,CPMOD,FIRST,IABSL,ICOMP,LENGTH,PDEG,PFA,PFL
      INTEGER RCOMP,RDIF,RPROD,RSUM,SEVAL,TAIL
      COMMON /TR4/ PRIME,PEXP
      P1=Q
      IN=IN1
      EPS=E
      HALF=PFL(PFA(1,0),PFL(PFA(2,0),0))
      LDCFP1=FIRST(TAIL(P1))
      DEGP1=PDEG(P1)
      L=BORROW(FIRST(IN))
      XX=PLIST1
      R=BORROW(FIRST(TAIL(IN)))
      X=R
      J=1
      GO TO 10
1     SR=SEVAL(LCP1,PLIST2,NPRIME,R)
      IF (SR .EQ. 0) GO TO 9
2     DIF=RDIF(R,L)
      DUM=RCOMP(DIF,EPS)
      CALL RERASE(DIF)
      IF (DUM .EQ. 1) GO TO 3
      CREFIN=PFL(L,PFL(R,0))
```

```
      GO TO 8
3     SUM=RSUM(L,R)
      MID=RPROD(SUM,HALF)
      CALL RERASE(SUM)
      X=MID
      J=2
      GO TO 10
4     SMID=SEVAL(LCP1,PLIST2,NPRIME,MID)
      IF (SMID .EQ. 0) GO TO 6
      IF (SMID .EQ. SR) GO TO 5
      CALL RERASE(L)
      L=MID
      GO TO 2
5     CALL RERASE(R)
      R=MID
      GO TO 2
6     CALL RERASE(R)
7     CREFIN=PFL(MID,PFL(BORROW(MID),0))
      CALL RERASE(L)
8     CALL RERASE(HALF)
      PLIST1=BORROW(PLIST1)
      CALL ERLA(XX)
      RETURN
9     MID=R
      GO TO 7
10    IF (X .NE. 0) GO TO 11
      ECEIL=0
      GO TO 14
11    N=IABSL(FIRST(X))
      D=FIRST(TAIL(X))
      IF (ICOMP(N,D) .EQ. 1) GO TO 12
      CALL ELPOF2(D,DUM,ECEIL)
      GO TO 13
12    CALL ELPOF2(N,DUM,ECEIL)
13    CALL ERLA(N)
14    NPRIME=(NORM+DEGP1*ECEIL)/PEXP+1
      NP=NPRIME-LENGTH(PLIST2)
15    IF (NP .LE. 0) GO TO (1,4),J
16    IF (PLIST1 .EQ. 0) GO TO 17
      CALL ADV(GFP,PLIST1)
      IF (CMOD(GFP,LDCFP1) .EQ. 0) GO TO 16
      PLIST2=PFA(GFP,PLIST2)
      LCP1=PFL(CPMOD(GFP,P1),LCP1)
      NP=NP-1
      GO TO 15
17    PRINT 18
18    FORMAT(28H0INSUFFICIENT PRIMES, CREFIN)
      STOP
      END
```

```
      SUBROUTINE ELPOF2(X,EFLR,ECEIL)
      INTEGER ECEIL,EFLR,Q,R,RP,TWO,V,X,Z
      INTEGER IABSL,IQRS,PFA
      V=X
      V=X
      EFLR=0
      ECEIL=0
      IF (V .EQ. 0) RETURN
      Q=IABSL(V)
      TWO=PFA(2,0)
      RP=0
1     Z=IQRS(Q,TWO)
      CALL ERLA(Q)
      CALL DECAP(Q,Z)
      CALL DECAP(R,Z)
      IF (R .EQ. 0) GO TO 2
      RP=RP+1
      CALL ERLA(R)
2     IF (Q .EQ. 0) GO TO 3
      EFLR=EFLR+1
      GO TO 1
3     IF (RP .GT. 1) ECEIL=1
      ECEIL=EFLR+ECEIL
      CALL ERLA(TWO)
      RETURN
      END


      SUBROUTINE IPRINT(UNIT,Y)
      COMMON /TR1/AVAIL,STAK,RECORD(72)
      INTEGER AVAIL,STAK,RECORD
      INTEGER A,B,DUM,F,FACT,FIVE,FL,Q,R,SIGN,UNIT,Y,Z
      INTEGER CONC,IBTOD,IQR,LENGTH,PFA,PPOWER
      IN=Y
      FIVE=PFA(5,0)
1     CALL ADV(L,IN)
      IF (L .NE. 0) GO TO 2
      CALL IWRITE(L,UNIT)
      GO TO 10
2     CALL ADV(A,L)
      CALL ADV(B,L)
      CALL ELPOF2(B,K,DUM)
      IF (K .NE. 0) GO TO 3
      CALL IWRITE(A,UNIT)
      GO TO 10
3     DUM=IQR(A,B)
      CALL DECAP(Q,DUM)
      CALL DECAP(R,DUM)
      Z=IBTOD(Q)
      CALL ERLA(Q)
      CALL DECAP(DUM,Z)
```

```
      FACT=PPOWER(FIVE,K)
      F=IPROD(R,FACT)
      CALL ERLA(R)
      CALL ERLA(FACT)
      FL=IBTOD(F)
      CALL ERLA(F)
      CALL DECAP(SIGN,FL)
      Z=PFA(SIGN,Z)
      DUM=K-LENGTH(FL)
      IF (DUM .EQ. 0) GO TO 5
      DO 4 I=1,DUM
4     FL=PFA(0,FL)
5     FL=PFA(43,FL)
      FL=CONC(Z,FL)
6     DO 7 I=1,72
      IF (FL .EQ. 0) GO TO 8
7     CALL DECAP(RECORD(I),FL)
      CALL WRITE(UNIT,RECORD)
      GO TO 6
8     DO 9 J=I,72
9     RECORD(J)=44
      CALL WRITE(UNIT,RECORD)
10    IF (IN .NE. 0) GO TO 1
      CALL ERLA(FIVE)
      RETURN
      END


      INTEGER FUNCTION IROOTS(X,Y)
      INTEGER DUM,DUM1,INTER,P1,X,Y
      P1=X
      INTER=Y
      DUM=0
      CALL RROOTS(P1,INTER,0,DUM,IROOTS,DUM1)
      CALL ERASE(DUM)
      RETURN
      END


      INTEGER FUNCTION IRTS(X,Y)
      INTEGER DUM,DUM1,INTER,P1,X,Y
      P1=X
      INTER=Y
      DUM=0
      CALL RRTS(P1,INTER,0,DUM,IRTS,DUM1)
      CALL ERASE(DUM)
      RETURN
      END
```

```
      INTEGER FUNCTION ISOLAT(X,Y)
      INTEGER DUM,DUM1,INTER,P1,X,Y
      P1=X
      INTER=Y
      DUM=0
      CALL RROOTS(P1,INTER,1,DUM,DUM1,ISOLAT)
      CALL ERASE(DUM)
      RETURN
      END


      INTEGER FUNCTION ISOLT(X,Y)
      INTEGER DUM,DUM1,INTER,P1,X,Y
      P1=X
      INTER=Y
      DUM=0
      CALL RRTS(P1,INTER,1,DUM,DUM1,ISOLT)
      CALL ERASE(DUM)
      RETURN
      END


      INTEGER FUNCTION ISTURM(X,Y)
      INTEGER D,DUM,DUM1,DP1,DP2,P1,P2,SP2,X,Y
      INTEGER BORROW,INV,PCONT,PDEG,PFL,PNEG,PSIGN,PSQ,PSREM
      P1=X
      P2=Y
      ISTURM=PFL(BORROW(P2),PFL(BORROW(P1),0))
      DP1=PDEG(P1)
1     DP2=PDEG(P2)
      DUM=PSREM(P1,P2)
      IF (DUM .NE. 0) GO TO 2
      ISTURM=INV(ISTURM)
      RETURN
2     D=DP1-DP2
      SP2=PSIGN(P2)
      P1=P2
      DP1=DP2
      DUM1=PCONT(DUM)
      P2=PSQ(DUM,DUM1)
      CALL PERASE(DUM)
      CALL ERLA(DUM1)
      IF (SP2 .EQ. -1 .AND. SP2**D .EQ. 1) GO TO 3
      DUM=P2
      P2=PNEG(P2)
      CALL PERASE(DUM)
3     ISTURM=PFL(P2,ISTURM)
      GO TO 1
      END
```

```
      INTEGER FUNCTION IVAR(X,Y)
      INTEGER LS,P,PT,S,SEQ,X,Y
      INTEGER REVAL
      SEQ=X
      PT=Y
      IVAR=0
      LS=0
1     CALL ADV(P,SEQ)
      S=REVAL(P,PT)
      IF (LS .EQ. 0) GO TO 2
      IF (S .EQ. 0 .OR. S .EQ. LS) GO TO 3
      IVAR=IVAR+1
2     LS=S
3     IF (SEQ .NE. 0) GO TO 1
      RETURN
      END
```

```
      INTEGER FUNCTION NROOTS(X)
      INTEGER INTER,L,P1,R,X
      INTEGER IROOTS,PFL,RBOUND,RDIF
      P1=X
      R=RBOUND(P1)
      L=RDIF(0,R)
      INTER=PFL(L,PFL(R,0))
      NROOTS=IROOTS(P1,INTER)
      CALL ERASE(INTER)
      RETURN
      END
```

```
      INTEGER FUNCTION NRTS(X)
      INTEGER INTER,L,P1,R,X
      INTEGER IRTS,PFL,RBOUND,RDIF
      P1=X
      R=RBOUND(P1)
      L=RDIF(0,R)
      INTER=PFL(L,PFL(R,0))
      NRTS=IRTS(P1,INTER)
      CALL ERASE(INTER)
      RETURN
      END
```

```
      INTEGER FUNCTION PNORMF(X)
      INTEGER COEF,DUM,L,P,X,Z
      INTEGER IABSL,ISUM,PFA,TAIL,TYPE
      P=X
```

```
      PNORMF=0
      IF (P .EQ. 0) RETURN
      IF (TYPE(P) .EQ. 1) GO TO 1
      PNORMF=IABSL(P)
      RETURN
1     L=0
2     P=TAIL(P)
      IF (P .EQ. 0) GO TO 4
      CALL ADV(COEF,P)
      IF (TYPE(COEF) .EQ. 1) GO TO 3
      Z=IABSL(COEF)
      DUM=PNORMF
      PNORMF=ISUM(PNORMF,Z)
      CALL ERLA(DUM)
      CALL ERLA(Z)
      GO TO 2
3     L=PFA(P,L)
      P=COEF
      GO TO 2
4     IF (L .EQ. 0) RETURN
      CALL DECAP(P,L)
      GO TO 2
      END
```

```
      INTEGER FUNCTION RBOUND(Y)
      INTEGER COEF,COEF2,DEG,DEN,DEN2,DUM,EXP,EXPP,N,P,TWO,X,Y
      INTEGER IABSL,ICOMP,PFA,PPOWER,RPOLY,TAIL
      P=TAIL(Y)
      CALL ADV(DEN,P)
      DEN=IABSL(DEN)
      CALL ELPOF2(DEN,DEN2,DUM)
      CALL ADV(N,P)
      EXP=1
1     IF (P .EQ. 0) GO TO 3
      CALL ADV(COEF,P)
      COEF=IABSL(COEF)
      CALL ADV(DEG,P)
      IF (ICOMP(DEN,COEF) .GE. 0) GO TO 2
      CALL ELPOF2(COEF,DUM,COEF2)
      EXPP=(COEF2-DEN2)/(N-DEG)+2
      IF (EXP .LT. EXPP) EXP=EXPP
2     CALL ERLA(COEF)
      GO TO 1
3     TWO=PFA(2,0)
      X=PPOWER(TWO,EXP)
      CALL ERLA(TWO)
      CALL ERLA(DEN)
      RBOUND=RPOLY(X)
      CALL ERLA(X)
      RETURN
      END
```

```
      INTEGER FUNCTION RCOMP(X,Y)
      INTEGER L,LD,LN,R,RD,RDLN,RN,RNLD,X,Y
      INTEGER FIRST,ICOMP,IPROD,ISIGNL,TAIL
      L=X
      R=Y
      IF (L .NE. 0) GO TO 2
      IF (R .NE. 0) GO TO 1
      RCOMP=0
      RETURN
1     RCOMP=-ISIGNL(FIRST(R))
      RETURN
2     IF (R .NE. 0) GO TO 3
      RCOMP=ISIGNL(FIRST(L))
      RETURN
3     LN=FIRST(L)
      LD=FIRST(TAIL(L))
      RN=FIRST(R)
      RD=FIRST(TAIL(R))
      RNLD=IPROD(RN,LD)
      RDLN=IPROD(RD,LN)
      RCOMP=ICOMP(RDLN,RNLD)
      CALL ERLA(RNLD)
      CALL ERLA(RDLN)
      RETURN
      END


      INTEGER FUNCTION REFINE(X,Y,Z)
      INTEGER DIF,DUM,EPS,HALF,IN,L,MID,P,R,SMID,SR,SUM,X,Y,Z
      INTEGER BORROW,PFA,PFL,RCOMP,RDIF,REVAL,RPROD,RSUM
      P=X
      IN=Y
      EPS=Z
      HALF=PFL(PFA(1,0),PFL(PFA(2,0),0))
      CALL ADV(L,IN)
      CALL ADV(R,IN)
      L=BORROW(L)
      R=BORROW(R)
      SR=REVAL(P,R)
      IF (SR .EQ. 0) GO TO 7
1     DIF=RDIF(R,L)
      DUM=RCOMP(DIF,EPS)
      CALL RERASE(DIF)
      IF (DUM .EQ. 1) GO TO 2
      REFINE=PFL(L,PFL(R,0))
      GO TO 6
2     SUM=RSUM(L,R)
      MID=RPROD(SUM,HALF)
      CALL RERASE(SUM)
      SMID=REVAL(P,MID)
```

```
      IF (SMID .EQ. 0) GO TO 4
      IF (SMID .EQ. SR) G) TO 3
      CALL RERASE(L)
      L=MID
      GO TO 1
3     CALL RERASE(R)
      R=MID
      GO TO 1
4     CALL RERASE(R)
5     REFINE=PFL(MID,PFL(BORROW(MID),0))
      CALL RERASE(L)
6     CALL RERASE(HALF)
      RETURN
7     MID=R
      GO TO 5
      END



      INTEGER FUNCTION REVAL(Q,P)
      INTEGER A,B,BI,CI,EI,I,P,Q,QI,T,U
      INTEGER BORROW,FIRST,IPROD,ISIGNL,ISUM,PFA,TAIL
      REVAL=0
      IF (Q .EQ. 0) RETURN
      QI=TAIL(Q)
      IF (P .NE. 0) GO TO 2
1     CALL ADV(CI,QI)
      CALL ADV(EI,QI)
      IF (QI .NE. 0) GO TO 1
      IF (EI .EQ. 0) REVAL=ISIGNL(CI)
      RETURN
2     A=FIRST(P)
      B=FIRST(TAIL(P))
      CALL ADV(REVAL,QI)
      REVAL=BORROW(REVAL)
      CALL ADV(I,QI)
      BI=PFA(1,0)
      GO TO 4
3     T=IPROD(REVAL,A)
      CALL ERLA(REVAL)
      REVAL=T
      T=IPROD(BI,B)
      CALL ERLA(BI)
      BI=T
      IF (QI .EQ. 0) GO TO 4
      IF (FIRST(TAIL(QI)) .LT. I) GO TO 4
      CALL ADV(CI,QI)
      QI=TAIL(QI)
      T=IPROD(CI,BI)
      U=ISUM(REVAL,T)
      CALL ERLA(REVAL)
```

```
      CALL ERLA(T)
      REVAL=U
4     I=I-1
      IF (I .GE. 0) GO TO 3
      CALL ERLA(BI)
      T=ISIGNL(REVAL)
      CALL ERLA(REVAL)
      REVAL=T
      RETURN
      END


      SUBROUTINE RROOTS(Q,XX,F,YY,ROOTS,ISOLAT)
      INTEGER CP1,CP2,D,DEG,DEGSEQ,DUM,DUM1,DUM2,DUM3,DUM4,ECEIL,F,FLAG
      INTEGER HALF,INTER,ISOLAT,J,L,LCP1,LCP2,LDEGSQ,LIST,LNPRMS,LV
      INTEGER MAXNP,MID,MIDV,N,NORMS,NP,P,PEXP,PI,PLIST1,PLIST2,PRIME
      INTEGER P1,P2,Q,R,ROOTS,RTS,RV,S,SEQ,SIGNS,SUM,X,XX,YY
      INTEGER BORROW,CMOD,CNPRS,CPMOD,FIRST,IABSL,ICOMP,INV,LENGTH
      INTEGER PDERIV,PFA,PFL,PPP,RPROD,RSUM,STURM,TAIL,VAR
      COMMON /TR4/PRIME,PEXP
      P1=Q
      INTER=XX
      FLAG=F
      DUM=YY
      DUM1=PDERIV(P1,FIRST(P1))
      P2=PPP(DUM1)
      CALL PERASE(DUM1)
      LCP1=FIRST(TAIL(P1))
      LCP2=FIRST(TAIL(P2))
      ISOLAT=0
      IF (DUM .NE. 0) GO TO 1
      DUM=STURM(P1,P2)
1     CALL DECAP(SEQ,DUM)
      CALL DECAP(PLIST2,DUM)
      CALL DECAP(SIGNS,DUM)
      CALL DECAP(NORMS,DUM)
      CALL DECAP(PLIST1,DUM)
      CALL DECAP(DEGSEQ,DUM)
      DUM4=PLIST1
      LDEGSQ=LENGTH(DEGSEQ)
      CALL ADV(L,INTER)
      CALL ADV(R,INTER)
      X=L
      J=1
      GO TO 13
2     LV=VAR(SEQ,PLIST2,LNPRMS,SIGNS,L)
      CALL ERLA(LNPRMS)
      X=R
      J=2
      GO TO 13
```

```
3     RV=VAR(SEQ,PLIST2,LNPRMS,SIGNS,R)
      CALL ERLA(LNPRMS)
      ROOTS=LV-RV
      IF (FLAG .EQ. 0 .OR. ROOTS .EQ. 0) GO TO 12
      LIST=0
      HALF=PFL(PFA(1,0),PFL(PFA(2,0),0))
      L=BORROW(L)
      R=BORROW(R)
4     RTS=LV-RV
      IF (RTS .EQ. 0) GO TO 9
      IF (RTS .NE. 1) GO TO 5
      ISOLAT=PFL(PFL(L,PFL(R,0)),ISOLAT)
      GO TO 10
5     SUM=RSUM(L,R)
      MID=RPROD(SUM,HALF)
      CALL RERASE(SUM)
      X=MID
      J=3
      GO TO 13
6     MIDV=VAR(SEQ,PLIST2,LNPRMS,SIGNS,MID)
      CALL ERLA(LNPRMS)
      IF (LV-MIDV .EQ. 0) GO TO 7
      LIST=PFA(LV,PFA(MIDV,PFA(L,PFA(BORROW(MID),LIST))))
      GO TO 8
7     CALL RERASE(L)
8     L=MID
      LV=MIDV
      GO TO 4
9     CALL RERASE(L)
      CALL RERASE(R)
10    IF (LIST .EQ. 0) GO TO 11
      CALL DECAP(LV,LIST)
      CALL DECAP(RV,LIST)
      CALL DECAP(L,LIST)
      CALL DECAP(R,LIST)
      GO TO 4
11    CALL RERASE(HALF)
12    CALL PERASE(P2)
      YY=PFL(NORMS,PFL(BORROW(PLIST1),PFL(DEGSEQ,0)))
      YY=PFL(SEQ,PFL(PLIST2,PFL(SIGNS,YY)))
      CALL ERLA(DUM4)
      RETURN
13    IF (X .NE. 0) GO TO 14
      ECEIL=0
      GO TO 17
14    N=IABSL(FIRST(X))
      D=FIRST(TAIL(X))
      IF (ICOMP(N,D) .EQ. 1) GO TO 15
      CALL ELPOF2(D,DUM,ECEIL)
      GO TO 16
15    CALL ELPOF2(N,DUM,ECEIL)
```

```
16    CALL ERLA(N)
17    LNPRMS=0
      DUM=DEGSEQ
      DUM2=NORMS
      MAXNP=0
18    CALL ADV(DEG,DUM)
      CALL ADV(N,DUM2)
      NP=(N+DEG*ECEIL)/PEXP+1
      LNPRMS=PFA(NP,LNPRMS)
      IF (MAXNP .LT. NP) MAXNP=NP
      IF (DUM .NE. 0) GO TO 18
      LNPRMS=INV(LNPRMS)
      NP=MAXNP-LENGTH(PLIST2)
19    IF (NP .LE. 0) GO TO (2,3,6),J
20    IF (PLIST1 .EQ. 0) GO TO 25
      CALL ADV(P,PLIST1)
      IF (CMOD(P,LCP1) .EQ. 0) GO TO 20
      IF (CMOD(P,LCP2) .EQ. 0) GO TO 20
      CP1=CPMOD(P,P1)
      CP2=CPMOD(P,P2)
      S=CNPRS(P,CP1,CP2)
      CALL ERLA(CP1)
      CALL ERLA(CP2)
      IF (LDEGSQ .EQ. LENGTH(S)) GO TO 22
21    CALL ERASE(S)
      GO TO 20
22    DUM=S
      DUM2=DEGSEQ
23    CALL ADV(DUM1,DUM)
      CALL ADV(DUM3,DUM2)
      IF (FIRST(DUM1) .NE. DUM3) GO TO 21
      IF (DUM .NE. 0) GO TO 23
      DUM1=SEQ
      DUM2=SEQ
24    CALL DECAP(PI,S)
      CALL ADV(DUM,DUM2)
      DUM=PFL(PI,DUM)
      CALL ALTER(DUM,DUM1)
      DUM1=DUM2
      IF (S .NE. 0) GO TO 24
      PLIST2=PFA(P,PLIST2)
      NP=NP-1
      GO TO 19
25    PRINT 26
26    FORMAT(28H INSUFFICIENT PRIMES, RROOTS)
      STOP
      END
```

```
      SUBROUTINE RRTS(Q,XX,F,SEQ,ROOTS,ISOLAT)
      INTEGER DUM,F,FLAG,HALF,INTER,ISOLAT,L,LIST,LV,MID,MIDV,P1,P2,Q
      INTEGER R,ROOTS,RTS,RV,SEQ,SUM,XX
      INTEGER BORROW,FIRST,ISTURM,IVAR,PDERIV,PFA,PFL,PPP,RSUM,RPROD
      P1=Q
      INTER=XX
      FLAG=F
      IF (SEQ .NE. 0) GO TO 1
      DUM=PDERIV(P1,FIRST(P1))
      P2=PPP(DUM)
      CALL PERASE(DUM)
      SEQ=ISTURM(P1,P2)
      CALL PERASE(P2)
1     CALL ADV(L,INTER)
      CALL ADV(R,INTER)
      LV=IVAR(SEQ,L)
      RV=IVAR(SEQ,R)
      ROOTS=LV-RV
      ISOLAT=0
      IF (FLAG .EQ. 0 .OR. ROOTS .EQ. 0) RETURN
      HALF=PFL(PFA(1,0),PFL(PFA(2,0),0))
      LIST=0
      L=BORROW(L)
      R=BORROW(R)
2     RTS=LV-RV
      IF (RTS .EQ. 0) GO TO 6
      IF (RTS .NE. 1) GO TO 3
      ISOLAT=PFL(PFL(L,PFL(R,0)),ISOLAT)
      GO TO 7
3     SUM=RSUM(L,R)
      MID=RPROD(SUM,HALF)
      CALL RERASE(SUM)
      MIDV=IVAR(SEQ,MID)
      IF (LV-MIDV .EQ. 0) GO TO 4
      LIST=PFA(LV,PFA(MIDV,PFA(L,PFA(BORROW(MID),LIST))))
      GO TO 5
4     CALL RERASE(L)
5     L=MID
      LV=MIDV
      GO TO 2
6     CALL RERASE(L)
      CALL RERASE(R)
7     IF (LIST .EQ. 0) GO TO 8
      CALL DECAP(LV,LIST)
      CALL DECAP(RV,LIST)
      CALL DECAP(L,LIST)
      CALL DECAP(R,LIST)
      GO TO 2
8     CALL RERASE(HALF)
      RETURN
      END
```

```
      INTEGER FUNCTION SEVAL(LQ,LP,NP,PT)
      INTEGER BB,C,D,DEG,DUM,I,LP,LPOLYS,LPRIME,LQ,N,NP,NPRIME,P,POINT
      INTEGER POLY,PT,PTD.PTN,QQ,R,X
      INTEGER CGARN,CMOD,CPEVAL,CPOWER,CPROD,CRECIP,FIRST,IPROD,ISIGNL
      INTEGER PFA,TAIL
      LPOLYS=LQ
      LPRIME=LP
      NPRIME=NP
      POINT=PT
      DEG=FIRST(FIRST(LPOLYS))
      QQ=PFA(1,0)
      BB=0
      IF (POINT .EQ. 0) GO TO 1
      PTN=FIRST(POINT)
      PTD=FIRST(TAIL(POINT))
1     DO 4 I=1,NPRIME
      CALL ADV(P,LPRIME)
      CALL ADV(POLY,LPOLYS)
      IF (POINT .NE. 0) GO TO 2
      R=CPEVAL(P,POLY,0)
      GO TO 3
2     N=CMOD(P,PTN)
      D=CMOD(P,PTD)
      X=CPROD(P,N,CRECIP(P,D))
      R=CPROD(P,CPOWER(P,D,DEG),CPEVAL(P,POLY,X))
3     C=CGARN(QQ,BB,P,R)
      CALL ERLA(BB)
      BB=C
      P=PFA(P,0)
      DUM=QQ
      QQ=IPROD(QQ,P)
      CALL ERLA(DUM)
4     CALL ERLA(P)
      SEVAL=ISIGNL(BB)
      CALL ERLA(BB)
      CALL ERLA(QQ)
      RETURN
      END
```

```
      INTEGER FUNCTION SQFREE(P)
      INTEGER DERV,DUM,GCD,P
      INTEGER FIRST,PDERIV,PGCD,PPP,PQ
      SQFREE=P
      IF (SQFREE .EQ. 0) RETURN
      DUM=PPP(SQFREE)
      DERV=PDERIV(DUM,FIRST(DUM))
      GCD=PGCD(DUM,DERV)
      SQFREE=PQ(DUM,GCD)
      CALL PERASE(DUM)
```

```
      CALL PERASE(GCD)
      CALL PERASE(DERV)
      RETURN
      END


      INTEGER FUNCTION STURM(XXX,YYY)
      INTEGER BB,BETK,BETKM1,BI,BSIGNS,C,COEF,D,DEGP,DEGQ,DEGSEQ,DUM
      INTEGER DUM1,DUM2,DUM3,DUM4,GFP,I,J,L,LDCP,LDCQ,LT,L1,NI,NIM1
      INTEGER NIP1,NK,NKM1,NORMI,NORMP,NORMQ,NORMS,NP,NPRIME,P,PCOUNT
      INTEGER PEXP,PI,PLIST1,PLIST2,PRIME,P1,P2,Q,QQ,R1,S,SEQ,SEQ2,SIGN
      INTEGER SIGNS,SK,SKM1,SKP1,S1,X,XXX,YYY
      INTEGER BORROW,CGARN,CMOD,CNPRS,CPMOD,FIRST,INV,IPROD,ISIGNL
      INTEGER LENGTH,PDEG,PFA,PFL,PNORMF,PSIGN,TAIL
      COMMON /TR4/PRIME,PEXP
      P=XXX
      Q=YYY
      DEGP=PDEG(P)
      DEGQ=PDEG(Q)
      NORMP=PNORMF(P)
      NORMQ=PNORMF(Q)
      LDCP=FIRST(TAIL(P))
      LDCQ=FIRST(TAIL(Q))
      CALL ELPOF2(NORMP,DUM,R1)
      CALL ELPOF2(NORMQ,DUM,S1)
      CALL ERLA(NORMP)
      CALL ERLA(NORMQ)
      IF (R1 .GE. S1) GO TO 1
      X=S1
      GO TO 2
1     X=R1
2     NPRIME=(X*(DEGP+DEGQ))/PEXP+1
      PLIST1=PRIME
      DEGSEQ=0
      PCOUNT=0
      PLIST2=0
      SEQ2=0
3     IF (PLIST1 .NE. 0) GO TO 5
      PRINT 4
4     FORMAT(27H0INSUFFICIENT PRIMES, STURM)
      STOP
5     CALL ADV(GFP,PLIST1)
      IF (CMOD(GFP,LDCP) .EQ. 0 .OR. CMOD(GFP,LDCQ) .EQ. 0) GO TO 3
      P1=CPMOD(GFP,P)
      P2=CPMOD(GFP,Q)
      S=CNPRS(GFP,P1,P2)
      CALL ERLA(P1)
      CALL ERLA(P2)
      DUM=S
      D=0
```

```
6       CALL ADV(DUM1,DUM)
        D=PFA(FIRST(DUM1),D)
        IF (DUM .NE. 0) GO TO 6
        D=INV(D)
        IF (DEGSEQ .EQ. 0) GO TO 10
        DUM1=DEGSEQ
        DUM2=D
7       CALL ADV(DUM3,DUM1)
        CALL ADV(DUM4,DUM2)
        IF (DUM3-DUM4) 10,8,9
8       IF (DUM1 .NE. 0 .AND. DUM2 .NE. 0) GO TO 7
        IF (DUM1 .EQ. 0 .AND. DUM2 .EQ. 0) GO TO 11
        IF (DUM1 .EQ. 0) GO TO 10
9       CALL ERLA(D)
        CALL ERASE(S)
        GO TO 3
10      CALL ERLA(DEGSEQ)
        CALL ERLA(PLIST2)
        CALL ERASE(SEQ2)
        DEGSEQ=D
        PLIST2=PFA(GFP,0)
        SEQ2=PFL(S,0)
        PCOUNT=1
        GO TO 12
11      SEQ2=PFL(S,SEQ2)
        PLIST2=PFA(GFP,PLIST2)
        PCOUNT=PCOUNT+1
        CALL ERLA(D)
12      IF (PCOUNT .LT. NPRIME) GO TO 3
        L1=LENGTH(DEGSEQ)
        SEQ2=INV(SEQ2)
        SEQ=0
13      DUM1=SEQ2
        DUM2=SEQ2
        LT=0
14      CALL ADV(S,DUM2)
        CALL DECAP(PI,S)
        LT=PFL(PI,LT)
        CALL ALTER(S,DUM1)
        DUM1=DUM2
        IF (DUM2 .NE. 0) GO TO 14
        SEQ=PFL(LT,SEQ)
        IF (S .NE. 0) GO TO 13
        CALL ERLA(SEQ2)
        SEQ=INV(SEQ)
        DUM=DEGSEQ
        NORMS=PFA(S1,PFA(R1,0))
        L=L1-2
        IF (L .EQ. 0) GO TO 16
         ALL ADV(DUM1,DUM)
           15 I=1,L
```

```
      CALL ADV(DUM1,DUM)
15    NORMS=PFA(R1*(DEGQ-DUM1+1)+S1*(DEGP-DUM1+1),NORMS)
16    NORMS=INV(NORMS)
      SIGNS=PFA(1,PFA(1,0))
      IF (L1 .EQ. 2) GO TO 25
      BSIGNS=PFA(PSIGN(Q),0)
      IF (L1 .EQ. 3) GO TO 22
      DUM=NORMS
      DUM1=SEQ
      DUM2=DEGSEQ
      DO 17 I=1,2
      CALL ADV(NORMI,DUM)
      CALL ADV(BI,DUM1)
17    CALL ADV(NI,DUM2)
      CALL ADV(NIP1,DUM2)
18    NIM1=NI
      NI=NIP1
      CALL ADV(NIP1,DUM2)
      CALL ADV(NORMI,DUM)
      CALL ADV(BI,DUM1)
      IF ((-1)**(NIM1-NI) .EQ. 1 .OR. (-1)**(NI-NIP1) .EQ. 1) GO TO 19
      SIGN=1
      GO TO 21
19    NP=NORMI/PEXP+1
      DUM3=PLIST2
      QQ=PFA(1,0)
      BB=0
      DO 20 J=1,NP
      CALL ADV(GFP,DUM3)
      CALL ADV(COEF,BI)
      COEF=FIRST(TAIL(COEF))
      C=CGARN(QQ,BB,GFP,COEF)
      CALL ERLA(BB)
      BB=C
      GFP=PFA(GFP,0)
      DUM4=QQ
      QQ=IPROD(QQ,GFP)
      CALL ERLA(DUM4)
20    CALL ERLA(GFP)
      SIGN=ISIGNL(BB)
      CALL ERLA(BB)
      CALL ERLA(QQ)
21    BSIGNS=PFA(SIGN,BSIGNS)
      IF (DUM2 .NE. 0) GO TO 18
      DUM=DEGSEQ
      CALL ADV(NKM1,DUM)
      CALL ADV(NK,DUM)
      CALL DECAP(BETK,BSIGNS)
      SKP1=BETK**(NKM1-NK+1)
       K=1
        GNS=PFA(SKP1,SIGNS)
```

```
      IF (BSIGNS .EQ. 0) GO TO 24
      SKM1=SK
      SK=SKP1
      BETKM1=BETK
      CALL DECAP(BETK,BSIGNS)
      NKM1=NK
      CALL ADV(NK,DUM)
      SKP1=SK*(SKM1*BETKM1*SK*BETK)**(NKM1-NK+1)
      GO TO 23
24    SIGNS=INV(SIGNS)
25    STURM=PFL(NORMS,PFL(BORROW(PLIST1),PFL(DEGSEQ,0)))
      STURM=PFL(SEQ,PFL(PLIST2,PFL(SIGNS,STURM)))
      RETURN
      END
```

```
      INTEGER FUNCTION VAR(SS,LP,LNP,LSCFS,E)
      INTEGER E,LNP,LNPRMS,LP,LPRIMS,LS,LSCFS,LSIGNS,NPRIME,P,PSIGN,PT
      INTEGER S,SS,SEQ
      INTEGER SEVAL
      SEQ=SS
      LPRIMS=LP
      LNPRMS=LNP
      LSIGNS=LSCFS
      PT=E
      VAR=0
      LS=0
1     CALL ADV(P,SEQ)
      CALL ADV(NPRIME,LNPRMS)
      CALL ADV(PSIGN,LSIGNS)
      S=SEVAL(P,LPRIMS,NPRIME,PT)*PSIGN
      IF (LS .EQ. 0) GO TO 2
      IF (S .EQ. 0 .OR. S .EQ. LS) GO TO 3
      VAR=VAR+1
2     LS=S
3     IF (SEQ .NE. 0) GO TO 1
      RETURN
      END
```

```
      INTEGER FUNCTION PPOWER(P,I)
      INTEGER DUM,I,J,L,N,P,Z,Z1
      INTEGER BORROW,PFA,PFL,PPROD,PVLIST
      Z=P
      N=I
      PPOWER=0
      IF (N .LT.0 .OR. Z .EQ. 0) RETURN
      IF (N .NE. 0) GO TO 2
      L=PVLIST(Z)
```

```
      PPOWER=PFA(1,0)
1     IF (L .EQ. 0) RETURN
      CALL DECAP(Z1,L)
      PPOWER=PF'.(Z1,PFL(PPOWER,PFA(0,0)))
      GO TO 1
2     PPOWER=BORROW(Z)
      IF (N .EQ. 1) RETURN
      N=N-1
      DO 3 J=1,N
      DUM=PPOWER
      PPOWER=PPROD(PPOWER,Z)
3     CALL PERASE(DUM)
      RETURN
      END
```

6. REFERENCES

[ASA64] "FORTRAN vs. Basic FORTRAN--A Programming Language for Information Processing on Automatic Data Processing Systems," Communications of the Association for Computing Machinery, Vol. 7, No. 10 (October 1964), pp. 591-625.

[ASA65] "Appendixes to ASA FORTRANs," Communications of the Association for Computing Machinery, Vol. 8, No. 5 (May 1965), pp. 655-657.

[COG67] Collins, G.E., "The SAC-1 List Processing System," University of Wisconsin Computing Center Report (34 pages), Madison, Wisconsin, July 1967.

[COG68a] Collins, G.E., "The SAC-1 Polynomial System," University of Wisconsin Computing Center Technical Reference No. 2 (68 pages), Madison, Wisconsin January 1968.

[COG68b] Collins, G.E., "The SAC-1 Rational Function System," University of Wisconsin Computing Center Technical Reference No. 8 (32 pages), Madison, Wisconsin, July 1968.

[COG68c] Collins, G.E. and Pinkert, J.R., "The Revised SAC-1 Integer Arithmetic System," University of Wisconsin Computing Center Technical Reference No. 9 (50 pages), Madison, Wisconsin, November 1968.

[COG69a] Collins, G.E., "Computing Time Analysis for Some Arithmetic and Algebraic Algorithms," Proceedings of the 1968 Summer Institute on Symbolic Mathematical Computation, R. G. Tobey, Editor, IBM Boston Programming Center, (June 1969), pp. 195-231.

[COG69b] Collins, G.E. et al., "The SAC-1 Modular Arithmetic System," University of Wisconsin Computing Center Technical Reference No. 10 (50 pages), Madison, Wisconsin, June 1969.

[COG70a] Collins, G.E. and Horowitz, E., "The SAC-1 Partial Fraction Decomposition and Rational Function Integration System," University of Wisconsin Computer Sciences Department Technical Report No. 80 (47 pages), Madison, Wisconsin, February 1970.

[HEL70] Heindel, L. E., Algorithms for Exact Polynomial Root Calculation, Ph.D. Thesis, University of Wisconsin, August 1970.