WIS-CS-73-188
Computer Sciences Department
The University of Wisconsin
1210 West Dayton Street
Madison, Wisconsin 53706

Received July 13, 1973

ALGEBRAIC ALGORITHMS FOR COMPUTING THE COMPLEX ZEROS OF GAUSSIAN POLYNOMIALS

by

James R. Pinkert

Technical Report #188

July 1973

PREFACE

This report is a revision of a May, 1973, Ph.D. Thesis in the Department of Computer Sciences at the University of Wisconsin. The changes consist of the following: correction of a few typographical and other minor errors; rewording of several short segments; deletion of the majority of program and sample output listings; deletion of the empirical computing time tables for all except the main algorithms.

The deletions were made in this instance to shorten the document and thus facilitate mechanical preparation of the report. However, a complete listing of the Fortran subprograms will appear in a forthcoming SAC-1 manual on the system (to be published in the near future).

ALGEBRAIC ALGORITHMS FOR COMPUTING THE COMPLEX ZEROS OF GAUSSIAN POLYNOMIALS*

by

James R. Pinkert

ABSTRACT

Let G be a univariate rational polynomial or a univariate Gaussian rational polynomial (a polynomial with Gaussian rational coefficients) having m distinct zeros. Algebraic algorithms are designed and implemented which, given G and a positive rational error bound ε, use Sturm's Theorem, the Routh-Hurwitz Theorems, and infinite precision integer arithmetic or modular arithmetic to compute m disjoint squares in the complex plane, each containing one zero of G and having width less than ε. Also included are algorithms for the following operations: associating with each square the multiplicity of the unique zero of G contained in the square; determining the number of zeros of G in regions of the complex plane such as circles and rectangles; refining selected individual zeros of G, that is, given G, a square S containing a single zero of G, and a positive rational error bound ε, computing a subsquare of S which contains the zero and has width less than ε. The theoretical computing times of the algorithms are analyzed and presented along with empirical computing times.

*Research supported by the National Science Foundation (Grant GJ-30125X) and the Mathematics Research Center

ACKNOWLEDGEMENTS

I wish to express my sincere appreciation to Professor George E. Collins for his guidance and assistance throughout my graduate studies and for the immense amount of time and effort which he has devoted to the development of my thesis.

Professor Bobby F. Caviness and Professor J. Barkley Rosser have also been of great help to me.

The Mathematics Research Center has provided generous support for my graduate studies. The National Science foundation and the Wisconsin Alumni Research Foundation have provided funding for research.

The last year of research and thesis preparation was done at the Stanford University Artificial Intelligence Project, and I wish to thank those associated with the project for their assistance and for the use of their excellent facilities.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 Incentives for development

Before advancing very far into the study of mathematics, one encounters the familiar expression,

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0 x^0$$

a "polynomial". More precisely, when the a_j are elements of some algebraic system A this expression is a "polynomial over A". The x is a formal symbol whose value is indeterminate. Usually A is a commutative ring with identity, so it has the operations of addition, subtraction, and multiplication.

Common as this expression may be, one aspect of it has received a great deal of attention from the outset. At what values of the indeterminate x does the expression evaluate to zero? Closed form solutions for $0 \leq n \leq 4$ were developed, but it was proved that for $n > 4$, a closed form solution did not exist. Hence the problem became one for the so-called "numerical mathematics".

No attempt will be made to present a historical survey here. Suffice it to say that interest in this problem grew as more and more instances appeared in which the solution of this or a related problem was of prime importance. Hence the number of techniques for its solution grew correspondingly.

These techniques entered a new phase with the development of the digital computer. It enhanced methods already in use and made practical some approaches that were far too cumbersome for hand calculation.

It also presented new problems, however. Foremost among these was that the use of fixed length "integer" or fixed precision "floating point" numbers in the computer led to errors in the calculations of the zeros. In some cases, these errors were not overly detrimental. In others, the so-called "ill-conditioned" polynomials, these errors made the solutions meaningless.

Again, no survey of methods will be presented. Many numerical techniques were (and are) being tried to circumvent the computational error problem.

In 1970, however, a significant advance was published. Heindel [HEL70] developed an algebraic, rather than numerical, system which would give the real roots of a polynomial over the integers to any desired accuracy. Hence the previous problems of loss of accuracy were solved.

It is important to discuss one aspect of this system here--polynomials over the integers. Potential use of Heindel's work has sometimes not been realized because the polynomial was over the "reals" rather than integers. The quotes are used because it becomes apparent that what is usually meant is not the reals in the standard mathematical sense, but rather in the sense of being commonly represented

in a computer as a "floating point" number. With present day binary computers, however, these floating point numbers are actually rationals, not reals.

The importance of this distinction is that later on it will be shown that for a rational polynomial R, one can compute an integral polynomial R' which has the same zeros as R. Hence any of the "real" polynomials can also be handled by the integer system. This is definitely not meant to say that every application should be run with the algebraic system. One drawback, for example, is that the basic arithmetic operations are software rather than hardware, and hence the algebraic system takes longer to run. What is meant, rather, is that floating point coefficients should not immediately imply numerical techniques.

Heindel's system does, however, work only with the real roots of integral polynomials. Often the complex roots are also required. In addition, applications for finding the roots of polynomials over the Gaussian integers have also arisen.

1.2 Operations performed by the system

What was needed, then, was a completely closed computer system (that is, no manual "starting values", monitoring of intermediate values, or other human intervention) which would input a polynomial over the integers or Gaussian integers and output all of the roots of the polynomial to any pre-specified, guaranteed accuracy. This is what the system presented in this report does.

To be more precise, if the input polynomial has m distinct roots, the system outputs a set of m disjoint squares in the complex plane, each containing one root of the polynomial and having width less than the specified value.

In addition to this main algorithm, there are algorithms for performing many associated operations. It is possible to obtain the number of roots of a polynomial in certain regions of the complex plane, such as circles, rectangles, horizontal and vertical half-planes, horizontal and vertical lines, and quadrants.

Having isolated the roots into disjoint squares (or having other knowledge of the location of roots), one may wish to refine only certain roots. That is, given the polynomial and a rectangle containing one root of the polynomial, compute a subrectangle of the specified maximum dimension which contains the root. Algorithms are provided

for doing this operation.

The main algorithms assume a polynomial P with only simple roots (roots of multiplicity one). If the application requires only the location and not the multiplicity of roots, there are algorithms for producing a polynomial P' which has each distinct root of P occuring as a simple root. Hence the main algorithms can be applied to P'. If the multiplicities as well as locations are important, there are algorithms which produce, along with the disjoint squares each containing one root of P, the multiplicity of the root in the square.

1.3 Applications

In addition to its intrinsic mathematical interest, there are many processes in the physical world which are expressed in terms of polynomials and whose important characteristics are related to the locations of the zeros of these polynomials. One such application, the stability of certain mechanical and electrical differential systems, created the initial interest in pursuing the research presented in this report.

1.4 SAC-1

The algorithms presented in this report are implemented in Fortran, making extensive use of SAC-1, system for Symbolic and Algebraic Computation. The extent of this system makes it impractical to describe even briefly individual subprograms.

However, a general overview of each module will be given here. In addition, all except the most straightforward algorithms presented in this report have verbal descriptions of the methods which they employ. Finally, many of the SAC-1 routines have names which clearly describe what they do. For example, IPROD, PPROD, RPROD, GIPROD, and GPPROD compute the product of two infinite precision integers, two polynomials, two rational functions, two Gaussian integers, and two Gaussian polynomials, respectively. Hence, even those not familiar with SAC-1 should be able to follow most of the algorithms.

In order to study the material in depth, however, a thorough understanding of the following modules is required: List Processing, Integer Arithmetic, Polynomial, Rational Function, Modular Arithmetic, Real Zero, and GCD and Resultant. Manuals giving detailed descriptions of these and the other SAC-1 modules are readily available from the University of Wisconsin Computer Sciences Department.

SAC-1 is not only extremely powerful, but also very

portable and easily used. A few so-called primitives are machine dependent and usually written in assembly language. The remainder of the system, however, is written in ASA Fortran. Hence SAC-1 can be easily implemented on virtually every machine with a Fortran compiler. In addition, one uses the system by calling the subprograms in the usual Fortran fashion. Hence there is no new language to learn; one need only become familiar with the subprograms.

The basis of the system is the list processing module [COG67]. SAC-1 uses a single link reference count structure. Space is saved using only a single link, and little efficiency is lost since the majority of applications proceed sequentially through the list. The reference count approach allows the overlapping of lists, which saves considerable storage. In addition, return of cells to available space requires time proportional to the number of cells returned rather than to the number of cells in use.

Consider an integer $A \neq 0$, where for some integer $\beta > 1$. $\beta^m \leq |A| < \beta^{m+1}$. Then A can be written as $A = \sum_{j=0}^{m} a_j \beta^j$ where $a_j = 0$ or $\text{sign}(a_j) = \text{sign}(A)$, $|a_j| < \beta$, and $a_m \neq 0$. The SAC-1 integer arithmetic system [COG68a] uses this technique, representing A as the list $(\alpha_0, \alpha_1, \ldots, \alpha_m)$ where α_j is the Fortran representation of a_j. This order is

chosen since most arithmetic operations begin with the low order digit. β can be set for the particular installation, subject to a few restrictions.

A polynomial P can be written $P(x) = \sum_{j=0}^{n} p_j x^j$ where $p_n \neq 0$ and $n = \deg(P)$. In this case the p_j are infinite precision integers or polynomials in other variables. This recursive canonical form lends itself to performing operations on the polynomials using recursive techniques. Polynomial P is represented in the SAC-1 polynomial system [COG68b] as the list $(X, c_r, e_r, \ldots, c_1, e_1)$ where X is the representation of the main variable and c_j is the representation of the j-th non-zero element p_{e_j} of the sequence $\{p_0, p_1, \ldots, p_n\}$. This representation does not include the zero coefficients. The list begins with the highest degree term, as most polynomial operations proceed in this order (as opposed to arithmetic order mentioned above).

The rational function system [COG68c] provides operations on rational numbers or rational functions. P/Q is represented as the list (r,s) were r and s are the representations of R and S such that R/S = P/Q, R and S are relatively prime, and sign(S) = 1.

The modular arithmetic system [COG69a] provides for arithmetic operations in GF(p) for p a prime less than β, and for operations on multivariate polynomials over GF(p). Certain polynomial operations can be done much faster than using classical methods by using homomorphisms, the modular operations, and then the Chinese Remainder Theorem.

The real zero system [HEL70b], written by Lee Heindel, has been discussed in the first section of this chapter.

The polynomial gcd and resultant system [COG72] uses modular and evaluation homomorphisms to compute gcd's and resultants much faster than with classical methods.

The Gaussian integer and polynomial system [CAB73] is scheduled for release at approximately the same time as this report, and the manual describing it is an invaluable accompanying document for the study of the material presented here. $G = G_1 + iG_2$ is represented as the list (g_1, g_2), where g_1 and g_2 are the representations of G_1 and G_2. Operations corresponding to those in the integer arithmetic and integral polynomial modules are provided for Gaussian integers and polynomials.

1.5 Algorithms and computing times

Knuth [KND67] gives a formal definition of an algorithm in the following manner. A computational method is a quadruple (Q, I, P, f) in which Q is a set containing subsets I and P, $f:Q \rightarrow Q$, and $f(p) = p$ for all $p \in P$. Q, I, P, and f represent the state of the computation, the input, the output, and the computational rule. Each $x \in I$ gives a computational sequence $x_0, x_1, \ldots$, defined by $x_0 = x$ and $x_{k+1} = f(x_k)$ for $k \geq 0$. The sequence terminates in K steps if K is the smallest k such that $x_k \in P$. x_K is then the output for input x. Some computational sequences never terminate. An algorithm is a computational method which terminates in finitely many steps for all $x \in I$.

If f is expressed in terms of elementary operations, such as a Turing machine table, and each execution of an elementary operation corresponds to the generation of another element in the sequence $x_{k+1} = f(x_k)$, then K can be termed the computing time of algorithm A = (Q, I, P, f) given input $x \in I$, which can be written $t_A(x) = K$.

As an example, consider the following algorithm SUM for adding two positive numbers a and b represented as strings of ones. Let the input tape be of the form 1...101...10...0 and the output tape be of the form 1...10...0 . Let the

following be indices: $i, j, k > 0$; $\ell, m, n \geq 0$; $0 \leq p \leq 1$; $q, r, s, t \geq 0$; $q + r + s + t = 1$. Let c^u be a string of u characters c. Let $S \in \{A, B, C, D\}$.

ALGORITHM SUM:

```
     q i r ℓ p m s j t n k
Q: Sx 1 x 1 0 1 x 1 x 1 0

      i  j k
I: Ax1 01 0

      i j
P: Dx1 0

           ℓ  i  j k      ℓ+1  i-1  j k
f: (1)  A1 x1 01 0  → A1    x1   01 0

           i   j k      i  j+1 k
   (2)  A1 x01 0  → B1 x1   0

           i  j k       i+1  j-1 k
   (3)  B1 x1 0   → B1    x1   0

           i  j      i-1   j
   (4)  B1 x0  → C1    x10

           i   j      i  j+1
   (5)  C1 x10  → D1 x0

           i  j      i-1   j
   (6)  D1 x0  → D1    x10

           i  j k      i-1  j+1 k
   (7)  D1 x1 0  → D1    x1   0
```

Note that this is just a string encoding of a four state Turing machine to perform the addition, with x being the read-write head and the character right of x being read or written.

For computing time analysis, the transformations in f are done the following number of times:

(1) a

(2) 1

(3) b + 1

(4) 1

(5) 1

(6) 1

(7) a + b - 1 .

Hence $t_{SUM}(a,b) = 2a + 2b + 4$.

Although this illustrates formal definitions, writing algorithms for complex operations in the above manner would be impractical and incomprehensible. Hence algorithms are written using meta-languages. The following is an example of this procedure.

ALGORITHM SORT:

SORT(A,N)

Array Sort

A is a Fortran linear array of N elements. A is sorted into ascending order.

Method:

Use a bubble sort.

Description:

(1) [Initialize and check for single element.]

If N = 1, return; j ← 1.

(2) [Initialize inner loop.] k ← j + 1.

(3) [Make pass through array.] If A[j] > A[k],

(t ← A[j]; A[j] ← A[k]; A[k] ← t); k ← k + 1; if k $\leq$ N, go to (3); j ← j + 1; if j < N, go to (2); return.

Describing computing times in the formal sense discussed above is also somewhat impractical. One could analyze algorithms in terms of the number of machine cycles used. However, each implementation would then need computing times of its own. In addition, the analysis may be very difficult and the usefulness of the results limited because of their complexity.

Hence computing times are given in terms of codominance equivalence classes. If f and g are two positive real valued functions defined on a set S, then f <u>is dominated by</u> g, written $f \underline{\propto} g$, if there is a positive real number c such that $f(x) \leq cg(x)$ for all $x \in S$. If $f \underline{\propto} g$ and $g \underline{\propto} f$, then f and g are <u>codominant</u>, written $f \sim g$. If $f \underline{\propto} g$ but not $g \underline{\propto} f$, then f <u>is strictly dominated by</u> g, written $f \propto g$.

Now consider the following definition of $t_A(x)$ introduced above. Let I_A be the well defined set of valid inputs to algorithm A (the elements of I_A may be n-tuples). Let P_A be the set of outputs of algorithm A for all inputs $x \in I_A$ (the elements of P_A may be m-tuples). When algorithm A is initialized with input $x \in I_A$, it performs a certain

finite number, $t_A(x)$, of primitve actions and stops with output $y \in P_A$. t_A is the <u>computing time function of algorithm A</u>.

Using the notions of codominance equivalence classes, one obtains meaningful expressions for this computing time function which are not burdened with encumbering details.

For example, $t_{SUM}(a,b) \sim a + b$ and $t_{SORT}(A,N) \underline{\propto} N^2$.

Subsequent algorithms in this report will include computing time functions as discussed above. In all but the obvious algorithms, proofs of the stated times will also be included. These proofs will take one of several forms.

For short algorithms or for algorithms in which a small set of steps are the major factor in determining computing time, a paragraph giving the derivations will be used. For example, in algorithm SORT one might use the following. "For $N > 1$, step (2) is executed $N - 1$ times and step (3) is executed $\sum_{j=1}^{N-1} (N - j) = N(N - 1)/2 \underline{\propto} N^2$ times. Hence

$$t_{SORT}(A,N) \underline{\propto} N^2 + N - 1 \underline{\propto} N^2 .$$"

For longer algorithms, a table will be used. This table will contain three entries: the number of the step, i; the number of times, n_i, that step (i) is executed; the

computing time, t_i, of step (i). For example, SORT might have the following table.

Step	n_i	t_i
1	1	~ 1
2	$< N$	~ 1
3	$< N^2$	~ 1

Explanation of non-obvious table entries (as perhaps n_3 above) follow the table.

In certain instances it is not possible to specify for some step (i) a pair of entries n_i, t_i such that the product $n_i t_i$ accurately reflects the actual computing time bound. A common example is when the j-th execution of step (i) is a function of j, say $t_i(j)$, and $\sum_{j=1}^{n} t_i(j) \propto n \cdot \max_{1 \le j \le n} \{t_i(j)\}$. Then the table entry is left blank and a derivation for the time of that step follows the table.

Certain properties of codominance equivalence classes will be used often in the derivation of computing times. For f, g, f_1, f_2, g_1, g_2 non-negative valued functions on S, the following hold, as shown in [MUD71], page 20:

(1) $0 \underline{\propto} f$;

(2) if c is a positive constant, $cf \sim f$;

(3) if $f_1 \underline{\propto} g_1$ and $f_2 \underline{\propto} g_2$ then $f_1 + f_2 \underline{\propto} g_1 + g_2$ and

$f_1 f_2 \underline{\propto} g_1 g_2$;

(4) if $f_1 \underline{\propto} g$ and $f_2 \underline{\propto} g$ then $f_1 + f_2 \underline{\propto} g$;

(5) $\max\{f,g\} \sim f + g$;

(6) if $1 \underline{\propto} f$ and $1 \underline{\propto} g$ then $f + g \underline{\propto} fg$;

(7) if $1 \underline{\propto} f$ and c is a positive constant, then $f \sim$

$f + c$;

(8) if $S = S_1 \times S_2 \times \ldots \times S_n$ and $f \underline{\propto} g$, then $f(a_1,$

$x_2, \ldots, x_n) \underline{\propto} g(a_1, x_2, \ldots, x_n)$ for any $a_1 \in$

S_1.

1.6 Format of presentation

The development of this system evolved naturally into the development of a sequence of well-delineated subsystems. Such a sequence of subsystems is also an instructive format for presentation of the system, and hence is used in this report.

The final section of this introductory chapter gives some basic definitions used in the rest of the report.

Chapter two presents algorithms used throughout the rest of the system. Section one is the major exception to the "well-delineated subsystems" mentioned above. It consists of a potpourri of algorithms for operations on integers, rational numbers, and polynomials. Probably the best approach to this section is to skim it initially and then refer back to it as required. Many system algorithms require square-free polynomials as input. Hence section two discusses algorithms for computing the greatest square-free divisor of a polynomial. The modular Sturm sequences discussed later use approaches involving mixed radix representations. Section three discusses modular arithmetic with specific emphasis on mixed radix representations and algorithms for their use.

Fundamental to the operation of the system are Sturm sequences and Routh-Hurwitz sequences. Chapter three discusses these sequences. Theoretical background is

presented in section one. Section two discusses the integer and modular generation of the sequences. Section three gives algorithms for using the sequences to determine the number of zeros of a polynomial in three areas of the complex plane--the upper and lower half planes and the real axis.

In order to work with areas other than these three basic areas, it is necessary to construct polynomials whose roots are certain functions of the roots of the input polynomial. Chapter four discusses these techniques. Section one presents a theoretical background of root squaring and mappings, section two discusses algorithms for root squaring, and section three discusses algorithms for mappings. Section four applies the techniques to determine the number of zeros of a polynomial in regions of the complex plane such as quadrants, rectangles, and circles.

Chapter five then presents the heart of the system--algorithms for isolating and refining the roots of a polynomial.

Chapter six presents extensions of the system. Section one gives algorithms for refining individual isolated roots of a polynomial. Section two discusses the square-free factorization of a polynomial, and section three uses this factorization to isolate and refine the roots of a polynomial that have multiplicity greater than one. Section four gives special input-output algorithms for the data

structures used in the system. Section five presents algorithms for working with polynomials over the rationals.

Chapter seven gives some closing remarks, empirical observations, and a sample run of the system.

1.7 Definitions and basic material

This section defines terms occuring in this thesis which have not been used widely enough yet to assume familiarity with them. In addition, it presents some elementary properties of certain functions. These properties will be used in many derivations and proofs.

Let $A(x_1, \ldots, x_r) = \sum_{i=0}^{n} A_i(x_1, \ldots, x_{r-1})x_r^i$ be a multivariate integral polynomial, $A_n \neq 0$. The leading numerical coefficient of A, lncf(A), is A_n if A_n is an integer and lncf(A_n) otherwise. The sign of A, sign(A), is sign(lncf(A)). The sum norm and the infinity norm of an integer a, $|a|_1$ and $|a|_\infty$, are the absolute value of a, $|a|$. $|A|_1 = \sum_{i=0}^{n} |A_i|_1$ and $|A|_\infty = \max_{0 \leq i \leq n} \{|A_i|_\infty\}$ are then the sum and infinity norms of the integral polynomial A.

Let $a = a_1 + ia_2$ be a Gaussian integer. Then $|a|_1 = |a_1| + |a_2|$ and $|a|_\infty = \max\{|a_1|, |a_2|\}$. Let $A(z_1, \ldots, z_r) = \sum_{i=0}^{n} A_i(z_1, \ldots, z_{r-1})z_r^i$ be a multivariate Gaussian polynomial. Using the preceding definitions for

Gaussian integers, the norms of A are defined exactly as for integer polynomials: $|A|_1 = \sum_{i=0}^{n} |A_i|_1$ and $|A|_\infty = \max_{0 \leq i \leq n} \{|A_i|_\infty\}$. Note that $|A|_1$ is a semi-norm and therefore $|A|_1 = 0$ if and only if $A = 0$, $|A|_1 = |-A|_1$, $|A + B|_1 \leq |A|_1 + |B|_1$, and $|AB|_1 \leq |A|_1 \cdot |B|_1$. Also, $|A|_\infty = 0$ if and only if $A = 0$, $|-A|_\infty = |A|_\infty$, and $|A + B|_\infty \leq |A|_\infty + |B|_\infty$. Finally, $|AB|_\infty \leq |A|_\infty \cdot |B|_1$. See [COG73b].

The length of an integer a, L(a), is defined as follows. $L_\beta(a) = \lceil \log_\beta (|a| + 1) \rceil$ for $a \neq 0$ and $L_\beta(0) = 1$. Since $\log_\beta \sim \log_\gamma$ for any other base γ and since in most contexts β is fixed, the subscript is omitted and L(a) is used. The following properties of L(a) are given in Collins, [COG73a], pp. 5 - 6:

(1) $L(a \pm b) \preceq L(a) + L(b)$;

(2) $L(ab) \sim L(a) + L(b)$ for $a,b \neq 0$;

(3) $L([a/b]) \sim L(a) - L(b) + 1$ for $|a| \geq |b| > 0$;

(4) $L(\prod_{i=1}^{n} a_i) \preceq \sum_{i=1}^{n} L(a_i)$;

(5) $L(\prod_{i=1}^{n} a_i) \sim \sum_{i=1}^{n} L(a_i)$ for $|a_i| > 1$;

(6) $L(a^b) \sim bL(a)$ for $|a| \geq 2$ and $b > 0$.

The length function will also be used for Gaussian integers and rational numbers, based on the following definitions. If $a = a_1 + ia_2$ is a Gaussian integer, then $L(a) = L(a_1) + L(a_2)$. If $r = r_1/r_2$ is a rational number, then $L(r) = L(r_1) + L(r_2)$.

Theorem 1.7.1: For arbitrary rational numbers r and s, $L(r + s) \underline{\propto} L(r) + L(s)$ and $L(rs) \underline{\propto} L(r) + L(s)$.

Proof: Trivial for $r = 0$ or $s = 0$, so assume $r \neq 0$ with $r = r_1/r_2$ in lowest terms and $s \neq 0$ with $s = s_1/s_2$ in lowest terms.

Let $t = r + s$. The result is trivial if $t = 0$, so assume $t \neq 0$ with $t = t_1/t_2$ in lowest terms. Then $L(t_1) \underline{\propto} L(r_1 s_2 + r_2 s_1) \underline{\propto} L(r_1 s_2) + L(r_2 s_1) \underline{\propto} L(r_1) + L(s_2) + L(r_2) + L(s_1) = L(r) + L(s)$ and $L(t_2) \underline{\propto} L(r_2 s_2) \underline{\propto} L(r_2) + L(s_2) \underline{\propto} L(r) + L(s)$, so $L(t) = L(t_1) + L(t_2) \underline{\propto} L(r) + L(s)$.

Now let $t = rs$, with $r, s \neq 0$ and $t = t_1/t_2$ in lowest terms. Then $L(t_1) \underline{\propto} L(r_1 s_1) \underline{\propto} L(r_1) + L(s_1)$ and $L(t_2) \underline{\propto} L(r_2 s_2) \underline{\propto} L(r_2) + L(s_2)$ so $L(t) = L(t_1) + L(t_2) \underline{\propto} L(r_1) + L(s_1) + L(r_2) + L(s_2) = L(r) + L(s)$. ∎

Subsequent discussions will often concern polynomials which differ only by a constant, and hence a term expressing this relationship is introduced here. Two polynomials A and B are <u>similar</u>, $A \simeq B$, just in case there are non-zero constants a and b such that $aA = bB$.

CHAPTER 2: AUXILIARY ALGORITHMS

2.1 Miscellaneous algorithms

This section consists of algorithms and other material which is used in many subsequent sections, but which does not fit well into any particular subsystem. Hence it should probably be skimmed on first reading and then referenced as required.

For example, many times it is necessary to interchange the values of two variables. This is usually done by the following sequence of steps: T ← A; A ← B; B ← T (where A and B are the variables to be interchanged). When several such operations must be performed sequentially, there is a large amount of repetitious code. Hence the following algorithm performs the interchange operation.

ALGORITHM CZFLIP:

CZFLIP(A,B)

Complex Zero System, Flip Values

A and B are arbitrary variables. The values of A and B are interchanged.

Description:

(1) [Interchange.] T ← A; A ← B; B ← T; return.

Computing Time: ∿ 1.

Because of the rigid nature of Fortran array allocation, it is sometimes advantageous to represent arrays as SAC-1

lists and use the inherent dynamic storage allocation of this structure. A graphic example of this can be seen in the main algorithms PRIR and GPRIR by reconstructing them using Fortran arrays. The following algorithm computes the transpose of a two dimensional array stored as a list.

ALGORITHM CZTRAN:

B=CZTRAN(A)

Complex Zero System Transpose

A is an m by n matrix, $m,n \geq 1$, in list form, A = $((a_{1,1}, a_{1,2}, \ldots, a_{1,n}), \ldots, (a_{m,1}, a_{m,2}, \ldots, a_{m,n}))$, where all $a_{i,j}$ are atoms or all $a_{i,j}$ are lists. B is the transpose of A, also in list form, B = $((a_{1,1}, a_{2,1}, \ldots, a_{m,1}), \ldots, (a_{1,n}, a_{2,n}, \ldots, a_{m,n}))$.

Description:

(1) [Initialize.] B ← 0; $\overline{A}$ ← A; C ← 0.

(2) [Generate list of pointers to rows.] ADV(c,$\overline{A}$); C ← PFA(c,C); if $\overline{A} \neq 0$, go to (2).

(3) [Obtain type.] T ← TYPE(c).

(4) [Initialize new row.] b ← 0; $\overline{C}$ ← C.

(5) [Loop.] c ← FIRST($\overline{C}$); ADV(a,c); if T = 0, (b ← PFA(a,b); go to (6)); b ← PFL(BORROW(a),b).

(6) [Increment and check.] ALTER(c,$\overline{C}$); $\overline{C}$ ← TAIL($\overline{C}$);

if $\overline{C} \neq 0$, go to (5).

(7) [Check outer loop.] B ← PFL(b,B); if c ≠ 0, go to (4).

(8) [Finish.] B ← INV(B); erase C; return.

Computing Time: $\sim$ mn, for A an m by n matrix.

Proof:

Step	n_i	t_i
1	1	$\sim$ 1
2	m	$\sim$ 1
3	1	$\sim$ 1
4	n	$\sim$ 1
5	mn	$\sim$ 1
6	mn	$\sim$ 1
7	n	$\sim$ 1
8	1	$\sim$ n

The next set of algorithms is from Rubald's forthcoming Ph.D. thesis [RUC73]. They are included here to facilitate reading of this report, since they are used often in subsequent algorithms.

Some representations in SAC-1 use pairs of elements. Examples include rational numbers and Gaussian polynomials. Other structures are naturally dealt with in pairs, such as the coefficient and exponent in the representation of a polynomial. In order to avoid the constant use of pairs of

subprogram calls, the following algorithms are two-element versions of ADV, DECAP, and FIRST.

ADV2 returns the first two elements of the list and moves the pointer to the third element.

ALGORITHM ADV2:

ADV2(a,b,C)

Advance 2 Elements

C is the input. a and b are outputs. C is a list $(c_1, c_2, \ldots, c_n)$, with $n \geq 2$. The new value of a is c_1 and the new value of b is c_2. The new value of C is the list $(c_3, c_4, \ldots, c_n)$. No reference counts are changed.

Description:

(1) ADV(a,C); ADV(b,C); return.

Computing Time: $\sim$ 1.

DECAP2 is similar to ADV2 except that in addition to advancing the list pointer and returning the first two elements, it erases the cells which contained these elements.

ALGORITHM DECAP2:

DECAP2(a,b,C)

Decap 2 Elements

C is the input. a and b are outputs. C is a list $(c_1, c_2, c_3, \ldots, c_n)$, with $n \geq 2$. The first two cells of C are removed and returned to available space. Thus the new value of C is the list $(c_3, c_4, \ldots, c_n)$. The new value of a is c_1 and the new value of b is c_2.

Description:

(1) DECAP(a,C); DECAP(b,C); return.

Computing Time: ~ 1.

FIRST2 returns the first two elements of the list, but does not alter the list pointer.

ALGORITHM FIRST2:

FIRST2(a,b,C)

First 2 Elements

C is the input. a and b are outputs. C is a list $(c_1, c_2, \ldots, c_n)$, with $n \geq 2$. The new value of a is c_1 and the new value of b is c_2. C is unchanged. No reference counts are changed.

Description:

(1) a ← FIRST(C); b ← FIRST(TAIL(C)); return.

Computing Time: ~ 1.

Sometimes in these pair structures one is interested

only in the second element. Hence the following algorithm returns this second element.

ALGORITHM SECOND:

a=SECOND(B)

Second Element

B is a list $(b_1, b_2, \ldots, b_n)$, with $n \geq 2$. The new value of a is b_2. B is unchanged. No reference counts are changed.

Description:

(1) a ← FIRST(TAIL(B)); return.

Computing Time: ∿ 1.

The next algorithm is used when one needs to convert an L-integer to a rational number.

ALGORITHM RNINT1:

B=RNINT1(A)

Rational Number from Integer (1)

A is an L-integer. B is the rational number B = A/1.

Description:

(1) B ← 0; if A = 0, return; B ← PFL(BORROW(A), PFL(PFA(1,0),0)); return.

Computing Time: ∿ 1.

The previous algorithm converts a single L-integer to a rational number by creating a denominator of +1. In order

to create a rational number from a pair of L-integers it would be convenient to have an algorithm for L-integers analogous to PGCDCF for polynomials. The following is such an algorithm, returning the GCD and the cofactors.

ALGORITHM IGCDCF:

IGCDCF(A,B,C,$\overline{A}$,$\overline{B}$)

Integer GCD and Cofactors

A and B are L-integers. C = gcd(A,B), $\overline{A}$ = A/C, and $\overline{B}$ = B/C.

Description:

(1) C ← IGCD(A,B); if FIRST(C) = 1 and TAIL(C) = 0, ($\overline{A}$ ← BORROW(A); $\overline{B}$ ← BORROW(B); return); $\overline{A}$ ← IQ(A,C); $\overline{B}$ ← IQ(B,C); return.

Computing Time: $\sim$ 1 if A = 0 and B = 0; otherwise $\propto$ n(m - k + 1), where m ={max L(A), L(B)}, n = min{L(A), L(B)}, and k = L(gcd(A,B)).

Proof: $t_{IGCD}(A,B) \propto n(m - k + 1)$, $t_{IQ}(A,C) \propto k(L(A) - k + 1) \leq n(m - k + 1)$, and $t_{IQ}(B,C) \propto k(L(B) - k + 1) \leq n(m - k + 1)$. ∎

The last algorithm in this set from Rubald's thesis uses the previous algorithm to convert an arbitrary pair of L-integers to a canonical SAC-1 rational number.

ALGORITHM RNINT2:

C=RNINT2(A,B)

Rational Number from Integers (2)

A and B are L-integers, $\neg(A \neq 0 \text{ and } B = 0)$. If $A = 0$ then $C = 0$. Otherwise, C is the rational number $C = C_1/C_2$ such that $\gcd(C_1, C_2) = 1$, $C_2 > 0$, and $C = A/B$.

Description:

(1) [Initialize and check for zero.] If $A = 0$, ($C \leftarrow 0$; return); IGCDCF(A,B,G,$\overline{A}$,$\overline{B}$); erase G.

(2) [Check signs.] $S \leftarrow$ ISIGNL($\overline{B}$); if $S = 1$, ($C \leftarrow$ PFL($\overline{A}$, PFL($\overline{B}$,0)); return); $C \leftarrow$ PFL(INEG($\overline{A}$),PFL(INEG($\overline{B}$),0)); erase $\overline{A}$, $\overline{B}$; return.

Computing Time: ~ 1 if $A = 0$; otherwise, $\propto n(m - k + 1)$, where $m = \max\{L(A), L(B)\}$, $n = \min\{L(A), L(B)\}$, and $k = L(\gcd(A,B))$.

Proof: $t_{IGCDCF}(A,B,G,\overline{A},\overline{B}) \propto n(m - k + 1)$. Let $\ell = m - n$. Then $t_{ISIGNL}(\overline{B})$, $t_{INEG}(\overline{A})$, $t_{INEG}(\overline{B}) \propto m + n \sim m = \ell + n \leq n \cdot \ell + n = n(m - n + 1) \leq n(m - k + 1)$. ∎

The SAC-1 rational function system does not include a routine for computing the negative of a rational function R, although one can of course use RDIF(0,R). Explicit negation of a rational function is performed by the following algorithm.

ALGORITHM RNEG:

R*=RNEG(R)

Rational Function Negation

R is a rational function. R* = -R, a rational function.

Description:

(1) R* ← 0; if R = 0, return; FIRST2(R_1,R_2,R); R* ← PFL(PNEG(R_1),PFL(BORROW(R_2),0)); return.

Computing Time: $\propto(\prod_{i=1}^{n} \lambda_i)L(a)$, where $R = R_1/R_2$, $\ell_i = \deg_i(R_1)$, $\lambda_i = \ell_i + 1$, and $a = |R_1|_\infty$.

One frequently has need of simple rational numbers such as 1/2. In order to lessen the code required to generate the internal representation of such numbers, the following algorithm inputs two Fortran integers and constructs a SAC-1 rational number from them.

ALGORITHM RNUM:

r=RNUM(a,b)

Rational Number

a and b are Fortran integers, $0 < |a|,|b| < \beta$; gcd(a,b) = 1; and b > 0. r is a rational number, r = a/b.

Description:

(1) $\bar{a}$ ← PFA(a,0); $\bar{b}$ ← PFA(b,0); r ← PFL($\bar{a}$,PFL($\bar{b}$,0)); return.

Computing Time: $\sim$ 1.

Another rational function operation that is used quite often in the system is the computation of the average of two rational numbers. For example, one frequently needs to find the midpoint of a line interval or of a strip in the complex plane.

ALGORITHM RNAVER:

c=RNAVER(a,b)

Rational Number Average

a and b are rational numbers. c is the rational number $c = (a + b)/2$.

Description:

(1) d ← RNUM(1,2); e ← RSUM(a,b); c ← RPROD(d,e); erase d, e; return.

Computing Time: $\propto L(a_1)L(b_2) + L(a_2)L(b_1) + L(a_2)L(b_2)$,

where $a = a_1/a_2$ and $b = b_1/b_2$.

The last rational function operation in this set inputs two rational numbers and computes equivalent rational numbers with a common denominator. This is used in such operations as transforming a complex rational coordinate $r_{11}/r_{12} + ir_{21}/r_{22}$ to the form a/c + ib/c so that efficient homothetic operations can be used.

ALGORITHM CZRPCD:

CZRPCD(r_1,r_2,a,b,c)

Complex Zero System, Rational

Pair to Common Denominator

r_1 and r_2 are inputs. a, b, and c are outputs. r_1 and r_2 are rational numbers. If $r_1 = r_2 = 0$, then a = b = c = 0. Otherwise, a, b, and c are the unique L-integers such that $r_1 = a/c$, $r_2 = b/c$, c > 0, and gcd(a,b,c) = 1.

Method:

Let $r_1 = r_{11}/r_{12}$ and $r_2 = r_{21}/r_{22}$. If $r_1 = 0$, then a = 0, b = r_{21}, and c = r_{22}. If $r_2 = 0$, then b = 0, a = r_{11}, and c = r_{12}. Otherwise, let $d = \gcd(r_{12},r_{22})$, $\bar{r}_{12} = r_{12}/d$, and $\bar{r}_{22} = r_{22}/d$. Then $a = r_{11}\bar{r}_{22}$, $b = r_{21}\bar{r}_{12}$, and $c = r_{12}\bar{r}_{22}$.

Description:

(1) [Initialize.] a ← 0; b ← 0; c ← 0; if $r_1 = 0$ and $r_2 = 0$, return.

(2) [Check for zero.] If $r_1 \neq 0$, FIRST2(r_{11},r_{12},r_1); if $r_2 \neq 0$, FIRST2(r_{21},r_{22},r_2); if $r_1 = 0$,

(b ← BORROW(r_{21}); c ← BORROW(r_{22}); return); if $r_2 = 0$,

(a ← BORROW(r_{11}); c ← BORROW(r_{12}); return).

(3) [Compute terms.] IGCDCF(r_{12}, r_{22}, d, $\bar{r}_{12}$, $\bar{r}_{22}$); a ←

IPROD(r_{11}, $\bar{r}_{22}$); b ← IPROD(r_{21}, $\bar{r}_{12}$); c ← IPROD(r_{12},

$\bar{r}_{22}$); erase d, $\bar{r}_{12}$, $\bar{r}_{22}$; return.

Computing Time: $\propto a_{11}a_{22} + a_{21}a_{12} + a_{12}a_{22}$, where $a_{ij} = L(r_{ij})$ for $1 \leq i,j \leq 2$.

Proof: $t_{IGCDCF}(r_{12}, r_{22}) \propto L(r_{12})L(r_{22})$. The time for the products is $\propto L(r_{11})L(r_{22}) + L(r_{12})L(r_{21}) + L(r_{12})L(r_{22})$. ∎

Many operations performed by the system require computing the greatest common divisor of two polynomials. PGCDCF, contained in the SAC-1 Polynomial GCD and Resultant System [COG72], is used for this purpose. Empirically, it is much faster than other polynomial gcd algorithms, with the added advantage that it also computes the cofactors of the inputs. In analyzing the theoretical computing time, however, one encounters the problem that its maximum computing time is not what would be expected. The following discussion derives the computing time of the present version

of PGCDCF, then shows how a simple modification of one step can reduce the maximum computing time to the expected result.

Theorem 2.1.1: Let A'_1, A'_2 be univariate integral polynomials, $m = \max\{\deg(A'_1), \deg(A'_2)\}$, $\mu = m + 1$, and $d = \max\{|A'_1|_\infty, |A'_2|_\infty\}$. The number of primes processed by PGCDCF is $\propto \mu L(\mu d)$.

Proof: Let $B' = \gcd(A'_1, A'_2)$, $A_1 = pp(A'_1)$, and $A_2 = pp(A'_2)$. If p is an odd prime such that $\neg(p \mid ldcf(A_1))$, $\neg(p \mid ldcf(A_2))$, and $\deg(\gcd(A_1 \bmod p, A_2 \bmod p)) = \deg(B')$, then p is termed a lucky prime. Otherwise, p is an unlucky prime [BRW71].

Let $p_1, p_2, \ldots, p_\ell$ be the lucky primes used by PGCDCF, $Q = \prod_{i=1}^{\ell} p_i$, and $Q' = \prod_{i=1}^{\ell-1} p_i$. In the notation of the algorithm (with $a_1 = ldcf(A_1)$ and $a_2 = ldcf(A_2)$),

$$Q > h = 2\ell\bar{b} = 2 \cdot \max(|A_{1,1}|, |A_{2,1}|) \cdot \gcd(a_1, a_2) \quad (1)$$

and

$$Q > m_i = mn_i = 2|\bar{B}|_1 \cdot |C_{i,1}|, \ (i = 1,2), \quad (2)$$

but Q' fails to satisfy at least one of these conditions.

Thus,

$$Q' \leq \max(h, m_1, m_2) . \tag{3}$$

Clearly,

$$h \leq 2\mu d^2 . \tag{4}$$

Now, $\mathrm{ldcf}(\overline{C}_i) = a_i = \mathrm{ldcf}(A_i)$, so $\overline{C}_i \mid a_i A_i$ and $\overline{B} \mid \overline{b}_i A_i$ so $\overline{B} \mid a_i A_i$. By Musser's Corollary, [MUD71] p. 115, since

$$|a_i A_i|_1 = |a_i| \cdot |A_i|_1 \leq d \cdot \mu d,$$

$$|\overline{B}|_1 , |\overline{C}_1|_1 , |\overline{C}_2|_1 \leq \mu^{2m} (\mu d^2). \tag{5}$$

By (2), (3), (4), and (5),

$$Q' \leq 2\mu^{4m+2} d^4 . \tag{6}$$

Since $2^{\ell-1} \leq Q'$, $\ell - 1 \leq 1 + (4m + 2)\log_2 \mu + 4\log_2 d$ and

$$\ell \leq \mu L(\mu) + L(d). \tag{7}$$

Every unlucky prime divides $S_k(A_1, A_2)$, where $k = \deg(B)$, and $|S_k(A_1, A_2)|_\infty \leq (\mu d)^{2(m-k)}$ so if t is the number of unlucky primes then

$$t \leq (m - k)L(\mu d). \tag{8}$$

By (7) and (8),

$$\ell + t \leq \mu L(\mu d). \tag{9}$$

Theorem 2.1.2: Let A'_1 and A'_2 be univariate polynomials with $\deg(A'_1), \deg(A'_2) \leq m$; $\mu = m + 1$; and $|A'_1|_\infty, |A'_2|_\infty \leq d$. Then the time to execute PGCDCF is $\underline{\propto} \mu^3 L(\mu d)^3$.

Proof: Let N be the bound on the number of primes used by PGCDCF. As shown in the previous theorem, $N \underline{\propto} \mu L(\mu d)$.

Let $B' = \gcd(A'_1, A'_2)$ and $C'_1 = A'_1/B'$, $C'_2 = A'_2/B'$. The following bounds hold for these quantities [MUD71]:

$$|B'|_\infty, |C'_1|_\infty, |C'_2|_\infty \leq (\mu d)^{2\mu}.$$

Then the timing chart for PGCDCF is as follows.

Step	n_i	t_i
1	1	~ 1
2	1	$\underline{\propto} \mu L(d)^2$
3	1	$\underline{\propto} L(d)^2$
4	1	$\underline{\propto} \mu L(\mu d) + L(d)^2$
5	$\leq N$	$\underline{\propto} L(d)$
6	$\leq N$	$\underline{\propto} \mu L(d)$
7	$\leq N$	$\underline{\propto} \mu^2$
8	$\leq N$	~ 1
9	$\leq N$	~ 1
10	$\leq N$	$\underline{\propto} \mu$

11	$\leq N$	~ 1
12	$\leq N$	~ 1
13	$\leq N$	$\propto \mu^2 L(\mu d)$
14	$\leq N$	$\propto \mu^2 L(\mu d)^2$
15	1	------
16	1	$\propto \mu^3 L(\mu d)^2$
17	1	$\propto \mu^2 L(\mu d)^2$

In step (4), $t_{4}(A_i) \propto \mu L(\mu d)$ and $t_{\text{IPROD}}(\ell,t) \propto L(\mu d)L(d)$. So $t_4 \propto \mu L(\mu d) + L(\mu d)L(d) \sim \mu L(\mu) + \mu L(d) + L(\mu)L(d) + L(d)^2 \sim \mu L(\mu d) + L(d)^2$.

In step (7), CGCDCF is called, and it in turn calls CPGCD1 since the polynomials are univariate. Hence $t_7 \propto \mu^2$.

$t_9, t_{11} \sim 1$ since the polynomials are univariate.

In step (13), if Q is a product of unlucky primes, then $Q \mid S_k(A_1, A_2)$, so $Q \leq (\mu d)^{2(m-k)}$, $L(Q) \propto \mu L(\mu d)$, and $t_{13} \propto \mu^2 L(\mu d)$. If Q is a product of lucky primes, then $L(Q) \propto \mu L(\mu) + L(d) \propto \mu L(\mu d)$.

In step (14), remarks similar to those for step (13)

apply. $L(|\overline{B}|_\infty)$, $L(|\overline{C}_1|_\infty)$, $L(|\overline{C}_2|_\infty) \leq \mu L(\mu d)$, so $t_{PNORMF}(\overline{B})$, $t_{PNORMF}(\overline{C}_i) \leq \mu^2 L(\mu d)$ and $t_{IPROD}(m,n_i) \leq \mu^2 L(\mu d)^2$.

In step (16), $|\overline{B}|_1$, $|\overline{C}_1|_1$, $|\overline{C}_2|_1 \leq \mu^{2m}(\mu d^2)$, as noted in the proof of the previous theorem, so $t_{16} \leq \mu L(\mu^{2m+1} 2^2 d^2)$

$$\sim \mu^3 L(\mu)^2 + \mu^2 L(\mu)L(d) + \mu L(d)^2 \leq \mu^3 L(\mu d)^2 .$$

In step (17), $|B|_\infty$, $|C_1|_\infty$, $|C_2|_\infty \leq (\mu d)^{2\mu}$, so $t_{17} \leq$.

$$\mu L((\mu d)^{2\mu})L(d) \leq \mu^2 L(\mu d)^2 .$$

Hence, $\sum_{i=1}^{17} n_i t_i \leq (N + \mu)\mu^2 L(\mu d)^2 \leq \mu^3 L(\mu d)^3$. ∎

One step of PGCDCF requires the comparison of the product of two numbers with another number. This multiplication is the cause of the problem in the computing time. An alternative approach is to compare the sum of the logarithms of the two numbers with the logarithm of the other number, thus eliminating the multiplication. The following algorithm is used in computing the required logarithms.

ALGORITHM FLOG2:

n=FLOG2(A)

Floor of the Logarithm to the Base 2

A is a positive L-integer. The SAC-1 implementation has $\beta = 2^{\xi}$. $n = \lfloor \log_2 (A) \rfloor$.

Method: Let $A = \sum_{i=0}^{j} a_i \beta^i$, with $a_j > 0$. Use the recurrence relation $m_0 = 1$, $m_i = 2m_{i-1} + 1$, to compute the least k such that $2^{k+1} - 1 = m_k \geq a_j$. Then $n = \xi j + k$.

Description:

(1) [Initialize.] $n \leftarrow 0$; $B \leftarrow A$; $C \leftarrow$ TAIL(A); go to (3).

(2) [Increment n and obtain next β-digit.] $n \leftarrow n + \xi$; $B \leftarrow C$; $C \leftarrow$ TAIL(C).

(3) [Check for last β-digit.] If $C \neq 0$, go to (2).

(4) [Initialize for high-order β-digit.] $m \leftarrow 1$; $b \leftarrow$ FIRST(B); go to (6).

(5) [Compute next term and increment n.] $m \leftarrow 2m + 1$; $n \leftarrow n + 1$.

(6) [Compare.] If $m < b$, go to 5; return.

Computing Time: $\propto L(A)$.

Proof:

Step	n_i	t_i
1	1	~ 1
2	L(A)-1	~ 1

3	L(A)	~ 1
4	1	~ 1
5	$\leq \xi-2$	~ 1
6	$\leq \xi-1$	~ 1

Hence $\sum_{i=1}^{6} n_i t_i \propto L(A) + \xi \propto L(A)$, since ξ is a constant for each implementation. ∎

FLOG2 can be used in PGCDCF in the following manner. Let $\overline{q}$ = FLOG2(Q), $\overline{m}$ = FLOG2(m), and $\overline{n}_i$ = FLOG2(n_i). Then $2mn_i < 2^{\overline{m}+\overline{n}_i+3}$. Hence if $\overline{q} \geq \overline{m} + \overline{n}_i + 3$, or equivalently $\overline{q} > m + n_i + 2$, then $Q > 2mn_i$.

Theorem 2.1.3: Changing step (14) of PGCDCF to the following changes the computing time T to $T \propto \mu^3 L(\mu d)^2$.

(14) $S \leftarrow$ ICOMP(Q,h); if $S \leq 0$, go to (5); $m \leftarrow$ PNORMF($\overline{B}$); $n_1 \leftarrow$ PNORMF($\overline{C}_1$); $n_2 \leftarrow$ PNORMF($\overline{C}_2$); $S \leftarrow$ FLOG2(Q) - FLOG2(m) - 2; $t_1 \leftarrow S$ - FLOG2(n_1); $t_2 \leftarrow S$ - FLOG2(n_2); erase m, n_1, n_2; if $t_1 \leq 0$ or $t_2 \leq 0$, go to (5).

Proof: $L(Q), L(m), L(n_i) \propto \mu L(\mu d)$, $t_{FLOG2}(X) \propto \mu L(\mu d)$ for $X = Q, m, n_i$. $t_{PNORMF}(X) \propto \mu^2 L(\mu d)$ for $X = \overline{B}, \overline{C}_1, \overline{C}_2$, so

$t_{14} \leq \mu^2 L(\mu d)$ and $n_{14}\, t_{14} \leq \mu^3 L(\mu d)^2$. ∎

When working with polynomials one frequently references two of the terms. If $A(x) = \sum_{i=0}^{n} a_i x^i$ is a polynomial of degree n, the leading coefficient of A, designated ldcf(A), is a_n, the coefficient of the high-order (n-th degree) term, and the trailing coefficient of A, designated trcf(A), is a_0, the coefficient of the constant (0-th degree) term.

For example, in later modular algorithms the trailing coefficient of a congruence polynomial is needed. Since the SAC-1 modular arithmetic system does not contain an algorithm for this purpose, such an algorithm is included in this system.

ALGORITHM CPTRCF:

a=CPTRCF(A)

Congruence Polynomial Trailing Coefficient

A is a univariate polynomial over a finite field GF(p). a is the trailing coefficient of A, that is, the constant term of A, an element of GF(p).

Description:

(1) [Initialize.] a ← 0; if A = 0, return; A' ← TAIL(A).

(2) [Obtain last element of A.] ADV(a,A'); if A' ≠ 0, go to (2); return.

Computing Time: ∿ μ, where m = deg(A) and μ = m + 1.

Proof: Step (1) is executed once with time $t_1 \sim 1$.

If $A \neq 0$, step (2) is executed μ times with time $t_2 \sim 1$. ∎

The next group of algorithms perform operations on integral polynomials. Some of these operations are not provided elsewhere in the SAC-1 system. Others could be performed by more general routines already existing; however, frequency of application makes the implementation of the specialized versions worthwhile.

The first such operation is the multiplication of a polynomial by an arbitrary power of its main variable. Determination of the main variable of a polynomial will depend, of course, on how the polynomial is specified; a polynomial can be rewritten to make any of its variables the main variable. If $A(x_1, x_2, \ldots, x_r) = \sum_{i=1}^{n} A_i x_r^i$ is a polynomial in r variables $x_1, \ldots, x_r$ with the A_i polynomials in r-1 variables $x_1, \ldots, x_{r-1}$, then x_r is called the main variable. In SAC-1 the main variable is clearly determined by the structure of the list representing the polynomial.

ALGORITHM PALTEX:

B=PALTEX(A,k)

Polynomial with Altered Exponent

A is a proper integral polynomial with main variable x; k is a Fortran integer such that $x^k A(x)$ has no negative exponents. Then B is the polynomial $B(x) = x^k A(x)$.

Description:

(1) [Initialize.] B ← 0; if A = 0, return; A' ← TAIL(A).

(2) [Obtain term of A.] ADV2(a,j,A').

(3) [Compute term of B.] B ← PFA(j+k, PFL(BORROW(a), B)); if A' ≠ 0, go to (2).

(4) [Finish.] B ← PFL(PVBL(A),INV(B)); return.

Computing Time: $\propto \mu$, where m = deg(A) and $\mu = m + 1$.

Proof:

Step	n_i	t_i
1	1	~ 1
2	$\leq \mu$	~ 1
3	$\leq \mu$	~ 1
4	≤ 1	$\propto \mu$

length(B) $\leq 2\mu$ in step (4), so $t_{INV}(B) \propto 2\mu \sim \mu$. ∎

The next algorithm adds a univariate polynomial and an L-integer, without the necessity of making the L-integer into a compatible 0-degree polynomial, as would be required

if using the SAC-1 program PSUM.

ALGORITHM PISUM:

B=PISUM(A,x,c)

Polynomial Integer Sum

A is a univariate integral polynomial. If $A \neq 0$, then x is the variable of A; if $A = 0$, then x is an arbitrary variable. If $A = 0$ and $c = 0$, then $B = 0$; otherwise, B is the univariate integral polynomial, in variable x, $B(x) = A(x) + c$.

Description:

(1) [c zero?] If $c = 0$, (B ← BORROW(A); return).

(2) [A zero?] C ← PFL(BORROW(x),PFL(BORROW(c),PFA(0, 0))); if $A = 0$, (B ← C; return).

(3) [Add.] B ← PSUM(A,C); erase C; return.

Computing Time: $\propto m + L(a) + L(c)$, where $m = \deg(A)$ and $a = |A|_\infty$.

Proof:

Step	n_i	t_i
1	1	~ 1
2	≤ 1	~ 1
3	≤ 1	$\propto m + L(a) + L(c)$

Consider the time to execute PSUM(A,C) in step (3). Since $\deg(C) = 0$, PSUM will scan A and borrow coefficients until it finds the low-degree term. This will take at most m steps with time ~ 1 for each step.

If A does not have a 0-degree term, then c is borrowed. If A does have a 0-degree term, then the coefficients are added with time $t \preceq L(a_0) + L(c)$, for a_0 the coefficent of the 0-degree term. Thus the total time $t_{PSUM}(A,C) \preceq m + L(a_0) + L(c) \preceq m + L(a) + L(c)$ since $|a_0| \leq a$. ∎

A third operation used very often in this system is the multiplication of an arbitrary polynomial by a linear monic polynomial. A polynomial P is linear in its main variable if deg(P) = 1; it is monic if ldcf(P) is a unit.

ALGORITHM PMLMP:

B=PMLMP(A,c)

Polynomial Multiplied by a

Linear Monic Polynomial

A is a univariate integral polynomial. c is an L-integer. B is the univariate integral polynomial $B(x) = A(x) \cdot (x + c)$.

Description:

(1) $B_1 \leftarrow$ PALTEX(A,1); $B_2 \leftarrow$ PIP(A,c); $B \leftarrow$ PSUM(B_1,B_2); erase B_1, B_2; return.

Computing Time: $\preceq \mu L(a)L(c)$, where $m = \deg(A)$, $\mu = m + 1$, and $a = |A|_\infty$.

Proof: $t_{PALTEX}(A,1) \preceq \mu$. $t_{PIP}(A,c) \preceq \mu L(a)L(c)$. $\deg(B_1), \deg(B_2) \leq \mu$ and $L(|B_1|_\infty), L(|B_2|_\infty) \preceq L(a) + L(c)$. Hence $t_{PSUM}(B_1, B_2) \preceq \mu\{L(a) + L(c)\}$. Thus the total time $t \preceq \mu L(a)L(c)$. ∎

The next algorithm returns the sign of the trailing coefficient of a polynomial.

ALGORITHM PTCS:

s=PTCS(A)

Polynomial Trailing Coefficient Sign

A is a univariate integral polynomial. s is a Fortran integer, the sign of the constant term of A (s = 0 if A = 0).

Description:

(1) [Obtain coefficient.] c ← PTLCF(A).

(2) [Obtain sign.] s ← ISIGNL(c); erase c; return.

Computing Time: $\preceq \mu + L(a)$, where $m = \deg(A)$, $\mu = m + 1$, and $a = |A|_\infty$.

Proof:

Step	n_i	t_i
1	1	$\preceq \mu$
2	1	$\preceq L(a)$

The content of a polynomial $A(x) = \sum_{i=0}^{n} a_i x^i$, written

cont(A), is the gcd of its coefficients; cont(A) = gcd(a_0, a_1, . . . , a_n). The primitive part of a polynomial, written pp(A), is the polynomial divided by its content; pp(A) = A/cont(A). The SAC-1 system has a routine for computing the primitive part of a polynomial. However, it returns the absolute value of the primitive part. In most cases, this is the desired result; however, in computing p.r.s.'s, one needs the signed primitive part. Hence the following algorithm computes the primitive part which has the same sign as the input polynomial.

ALGORITHM PSPP:

B=PSPP(A)

Polynomial Signed Primitive Part

A is a non-zero integral polynomial. B = A/cont(A), a primitive polynomial whose sign is the same as the sign of A.

Description:

(1) [Compute content.] C ← PCONT(A).

(2) [Divide by content.] B ← PSQ(A,C); erase C; return.

Computing Time: $\leq \mu L(a)^2$, where m = deg(A), μ = m + 1, and $a = |A|_\infty$.

Proof: $t_{PCONT}(A) \leq \mu L(a)^2$. $t_{PSQ}(A,C) \leq \mu L(a)^2$ since $C \leq a$. ∎

The order of a non-zero polynomial $\sum_{i=0}^{n} a_i x^i$ is its order as a formal power series, namely the least k such that $a_k \neq 0$. The following algorithm determines the order of the input polynomial.

ALGORITHM PORD:

n=PORD(A)

Polynomial Order

A is a non-zero univariate integral polynomial. n is the order of A.

Description:

(1) [Initialize.] A' ← TAIL(A).

(2) [Loop.] A' ← TAIL(A'); ADV(n,A'); if A' ≠ 0, go to (2); return.

Computing Time: $\propto \mu$, where m = deg(A) and μ = m + 1.

Proof:

Step	n_i	t_i
1	1	~ 1
2	$\leq \mu$	~ 1

In some subsequent algorithms it is required that the input polynomial have no roots at the origin. A polynomial P(x) has a root at the origin, of course, just in case P(0) = 0. The following algorithm uses PORD to compute a polynomial which has the same roots as the input polynomial

except that the roots at the origin have been removed.

ALGORITHM PWRAZR:

B=PWRAZR(A)

Polynomial with Roots at Zero Removed

A is a univariate integral polynomial. If A = 0, then B = 0. Otherwise, B(X) = A(x)/x^k, where k is the order of A.

Description:

(1) [Initialize.] B ← 0; if A = 0, return; k ← PORD(A).

(2) [Compute B.] If k = 0, (B ← BORROW(A); return); B ← PALTEX(A,-k); return.

Computing Time: $\propto \mu$, where m = deg(A) and μ = m + 1.

Proof:

Step	n_i	t_i
1	1	$\propto \mu$
2	1	$\propto \mu$

This completes the integral polynomial set. The next two algorithms in this section perform operations on Gaussian polynomials. The first constructs a simple Gaussian polynomial from two coefficients and exponents.

ALGORITHM GPCONS:

B=GPCONS(A,c_1,e_1,c_2,e_2)

Gaussian Polynomial Construction

A(z) is a non-zero univariate Gaussian polynomial in

variable z. c_1 and c_2 are L-integers. e_1 and e_2 are non-negative Fortran integers. $B(z) = c_1 z^{e_1} + ic_2 z^{e_2}$, a univariate Gaussian polynomial compatible with A(z).

Description:

(1) [Initialize and check first segment.] $B_1 \leftarrow 0$; $B_2 \leftarrow 0$; $B \leftarrow 0$; if $c_1 \neq 0$, $B_1 \leftarrow$ PFL(GPVBL(A),PFL(BORROW(c_1),PFA(e_1,0))).

(2) [Check second segment.] If $c_2 \neq 0$, $B_2 \leftarrow$ PFL(GPVBL(A),PFL(BORROW(c_2),PFA(e_2,0))).

(3) [Assemble.] If $B_1 \neq 0$ or $B_2 \neq 0$, $B \leftarrow$ PFL(B_1, PFL(B_2,0)); return.

Computing Time: ~ 1.

The final algorithm of this section computes the symmetric part of a Gaussian polynomial, defined as follows. Let $A = A_1 + iA_2$ be a Gaussian polynomial and let $B = \gcd(A_1, A_2)$. Then B is the symmetric part of A.

ALGORITHM GPSYMP:

GPSYMP(A,B,C)

Gaussian Polynomial Symmetric Part

A is the input. B and C are outputs. A is a non-zero univariate Gaussian polynomial. B is the symmetric part of A, that is, the gcd of the real and imaginary parts of A. C = A/B. B is a univariate integral polynomial and C is a univariate Gaussian polynomial.

Description:

(1) [Compute gcd and cofactors.] FIRST2(A_1,A_2,A); $C_1 \leftarrow 0$; $C_2 \leftarrow 0$; if $A_1 = 0$, ($B \leftarrow$ PABS(A_2); $C_2 \leftarrow$ PQ(A_2,B); go to (2)); if $A_2 = 0$, ($B \leftarrow$ PABS(A_1); $C_1 \leftarrow$ PQ(A_1,B); go to (2)); PGCDCF(A_1,A_2,B,C_1,C_2).

(2) [Construct C.] $C \leftarrow$ PFL(C_1,PFL(C_2,0)); return.

Computing Time: $\propto \mu^3 L(\mu d)^2$, where $m = \deg(a)$, $\mu = m + 1$, and $d = |A|_\infty$.

Proof: Follows from the time for PGCDCF, noting that $|A_1|_\infty$, $|A_2|_\infty \leq d$ and $\deg(A_1)$, $\deg(A_2) \leq m$. ∎

2.2 Greatest square-free divisors

The concept of multiplicity mentioned in the introduction is of great importance when dealing with the roots of a polynomial. The multiplicity of a root α_i of a polynomial $A(x)$ is the largest positive k such that $(x - \alpha_i)^k | A(x)$. The concept of multiplicity is directly related to that of "square-free".

A polynomial A is square-free just in case there is no polynomial B of positive degree such that $B^2 | A$. To say that a polynomial is square-free is equivalent to saying that the roots of the polynomial are simple, that is, of multiplicity one. This is shown in the following theorem.

Theorem 2.2.1: A Gaussian polynomial is square-free if and only if its roots are all simple.

Proof: Let A be a Gaussian polynomial and assume that B is a Gaussian polynomial of positive degree such that $B^2 | A$. Let $B = b\prod_{j=1}^{n} (x - \alpha_j)^{e_j}$ and $C = A/B^2$. Then $A = Cb^2 \prod_{j=1}^{n} (x - \alpha_j)^{2e_j}$, so $\alpha_1, \alpha_2, \ldots, \alpha_n$ are multiple roots of A.

Now assume that α_1 is a multiple root of Gaussian

polynomial A. Since α_1 is an algebraic number, there is an irreducible Gaussian polynomial B of positive degree which has α_1 as a simple root. Let C = gcd(A,B). C must also have α_1 as a root, since it is a common root of A and B. Hence C is of positive degree. C|A and C|B. But B is irreducible, so since C|B and C is of positive degree, then C ≃ B. Hence B|A. Now let D = A/B. α_1 is a multiple root of A and a simple root of B, so α_1 must be a root of D. By the same argument as above, B|D. Thus $B^2|(A/B)$, so $B^2|A$. Hence A is not square-free.

Given a square-free factorization of a polynomial it is then a trivial task to determine the multiplicity of a root. A <u>square-free factorization</u> of a polynomial A is a sequence of polynomials $A_1, A_2, \ldots, A_n$ of positive degrees and a sequence of distinct positive integers $e_1, e_2, \ldots, e_n$ such that $A = \prod_{j=1}^{n} A_j^{e_j}$ and $\prod_{j=1}^{n} A_j$ is square-free. This problem will be dealt with in a later section.

If, however, one is interested only in the location of the roots, then all that is required is a polynomial A* which has all of the unique roots of A occuring as simple roots. One can obtain A* by computing a greatest

square-free divisor of A.

A square-free divisor of a polynomial A is a polynomial B such that B|A and B is square-free. A greatest square-free divisor C is a square-free divisor such that if B is another square-free divisor then B|C.

The method used to obtain a greatest square-free divisor is based on Section 2.4 of Musser [MUD71], where it is shown that A/gcd(A,A') is a greatest square-free divisor of A.

Also, using Musser's discussion and Theorem 2.2.1 above, it is easy to verify the previous assertion that a greatest square-free divisor of A has all of the unique roots of A occuring as simple roots.

The term "the" greatest square-free divisor of A will be used for a greatest square-free divisor B of A such that sign(B) = 1 if A and B are integral polynomials, and ldcf(B) is in the first quadrant or on the positive real axis if A and B are Gaussian polynomials.

The first algorithm finds the greatest square-free divisor of an integral polynomial, using the method discussed above.

ALGORITHM PGSFD:

B=PGSFD(A)

Polynomial Greatest Square-Free Divisor

A is an integral polynomial of positive degree. B is

the greatest square-free divisor of the primitive part of A.

Method:

Let $A = pp(\overline{A})$. Then $B = \overline{A}/gcd(\overline{A},\overline{A}')$.

Description:

(1) [Compute value.] $\overline{A} \leftarrow PPP(A)$; $v \leftarrow PVBL(\overline{A})$; $\overline{A}' \leftarrow$ PDERIV($\overline{A}$,v); erase v; PGCDCF($\overline{A},\overline{A}',C,B,D$); erase $\overline{A}$, $\overline{A}'$, C, D; return.

Computing Time: $\preceq \mu^3 L(\mu d)^2$ for univariate polynomials, where $m = deg(A)$, $\mu = m + 1$, and $d = |A|_\infty$.

Proof: $|\overline{A}'|_\infty \leq \mu d$, so $t_{PGCDCF}(\overline{A},\overline{A}') \preceq \mu^3 L(\mu^2 d)^2 \preceq \mu^3 L(\mu d)^2$. ∎

The following algorithm performs the same operation for a Gaussian polynomial.

ALGORITHM GPGSFD:

B=GPGSFD(A)

Gaussian Polynomial Greatest

Square-Free Divisor

A is a univariate Gaussian polynomial of positive degree. B is the greatest square-free divisor of the primitive part of A.

Method:

Let $\overline{A} = pp(A)$. Then $B = \overline{A}/gcd(\overline{A},\overline{A}')$.

Description:

(1) [Compute value.] $\overline{A} \leftarrow$ GPPP(A); v $\leftarrow$ GPVBL($\overline{A}$); $\overline{A}' \leftarrow$ GPDERV($\overline{A}$,v); erase v; GPGCDC($\overline{A}$,$\overline{A}'$,C,B,D); erase $\overline{A}$, $\overline{A}'$, C, D; return.

Computing Time: $\preceq \mu^3 L(\mu d)^2$ for univariate polynomials, where $m = \deg(A)$, $\mu = m + 1$, and $d = |A|_\infty$.

Proof: $|\overline{A}'|_\infty \leq \mu d$, so $t_{GPGCDC}(\overline{A},\overline{A}') \preceq \mu^3 L(\mu^2 d)^2 \preceq \mu^3 L(\mu d)^2$. ∎

2.3 Mixed radix representations

Many algorithms in SAC-1 generate, and subsequently must perform operations on, very large L-integers. A graphic example of this is provided by the polynomial remainder sequence which is used later. The coefficients of the polynomials in these sequences can grow very rapidly.

In order to decrease the amount of time spent on operations involving large L-integers, a great deal of attention has been focused on modular algorithms. Whereas the time to multiply two L-integers a and b is $\sim L(a)L(b)$, the time to perform a multiplication in a finite field GF(p) is $\propto L(p)^2$. Since in virtually all cases single precision primes are sufficient, $L(p)^2 \sim 1$. Corresponding comparisons can be made for other operations.

The advantages of performing the operations in GF(p) can be employed as follows. Suppose one must do an operation S op T. If it is possible to obtain a bound B on the numerical components of the result U, such as the coefficients of a polynomial, then the operation $(a \bmod p_i)$ op $(b \bmod p_i)$ can be done for sufficiently many primes p_i such that $2B < \prod_{i=1}^{n} p_i$. (An alternate approach when an a priori bound cannot be computed is indicated below.) Let

the results of these n operations be $(u_1, u_2, \ldots, u_n)$.

Then the following theorem, the Chinese Remainder Theorem, insures that there is a unique value U in the specified range which produces the given residues u_i for primes p_i.

Theorem 2.3.1: If $p_1, p_2, \ldots, p_n$ are positive integers which are pairwise relatively prime ($\gcd(p_i, p_j) = 1$ for $i \neq j$) and if $u_1, u_2, \ldots, u_n$ are integers then there exists exactly one integer U such that $-\prod_{i=1}^{n} p_i < 2U < \prod_{i=1}^{n} p_i$ and $U \equiv u_i \pmod{p_i}$ for $1 \leq i \leq n$.

There are many proofs of this theorem in the literature. For example see [KND68].

The pair of lists $(u_1, u_2, \ldots, u_n)$ and $(p_1, p_2, \ldots, p_n)$ will be termed the residue representation of integer U. Null lists () and () are taken as representative of U = 0.

Having performed the operation modulo p_i for sufficiently many primes p_i, it is necessary to have an efficient method to compute the unique integer U from the residue representation.

There are various methods to do this, one of which will

be considered here. Let p_i, $1 \leq i \leq n$, be an odd prime.

Suppose one has $Q_{n-1} = \prod_{j=1}^{n-1} p_j$ and U_{n-1} such that $|U_{n-1}| < Q_{n-1}/2$ and $U_{n-1} \equiv u_i \pmod{p_i}$ for $1 \leq i < n$. It is now desired to compute U_n given u_n and p_n such that, for $Q_n = \prod_{i=1}^{n} p_i$, $|U_n| < Q_n/2$ and $U_n \equiv u_i \pmod{p_i}$ for $1 \leq i \leq n$.

Theorem 2.3.2: Let Q_i, U_i, u_i, and p_i be defined as in the preceding paragraph. Define the following quantities:

$$q_{n-1} = Q_{n-1}^{-1} \text{ in } GF(p_n); \tag{1}$$

$$d_n^* = q_{n-1}(u_n - U_{n-1}) \pmod{p_n}; \tag{2}$$

$$d_n = \begin{cases} d_n^* \text{ if } d_n^* < p_n/2 \\ d_n^* - p_n \text{ otherwise} \end{cases}; \tag{3}$$

$$U_n = Q_{n-1} d_n + U_{n-1}. \tag{4}$$

Then $|U_n| < Q_n/2$ and $U_n \equiv u_i \pmod{p_i}$ for $1 \leq i \leq n$.

Proof: From (2) and (3), $d_n \equiv q_{n-1}(u_n - U_{n-1}) \pmod{p_n}$ and $|d_n| < p_n/2$; hence $|d_n| \leq (p_n - 1)/2$ since p_n is odd.

From the definition of Q_{n-1}, $p_i | Q_{n-1}$ for $1 \leq i < n$. Hence by (4) $U_n \equiv U_{n-1} \equiv u_i \pmod{p_i}$ for $1 \leq i < n$. $U_n \equiv Q_{n-1} d_n$

$+ U_{n-1} \equiv \dot{Q}_{n-1} q_{n-1} (u_n - U_{n-1}) + U_{n-1} \equiv u_n \pmod{p_n}$, since by (1) $q_{n-1} Q_{n-1} \equiv 1 \pmod{p_n}$. $|d_n| \leq (p_n - 1)/2$ and $|U_{n-1}| < Q_{n-1}/2$, so $|U_n| = |Q_{n-1} d_n + U_{n-1}| < Q_{n-1} (p_n - 1)/2 + Q_{n-1}/2 = p_n Q_{n-1}/2 = Q_n/2$. ∎

The method given in this theorem is used in the algorithm CCRA in the SAC-1 Polynomial GCD and Resultant System [COG72].

Note that if $Q_0 = 1$ and hence $U_0 = 0$, then $U_1 \equiv u_1 \pmod{p_1}$ and $|U_1| < p_1/2$. This gives a convenient method of initialization.

Hence given a list $(p_1, p_2, \ldots, p_n)$ of distinct odd primes and a list of residues, one can compute the desired integer U using the following recurrence relations:

$$Q_0 = 1, \quad U_0 = 0, \quad q_0 = 1.$$

$$d^*_i = q_{i-1} (u_i - U_{i-1}) \pmod{p_i},$$

$$d_i = \begin{cases} d^*_i & \text{if } d^*_i < p_i/2 \\ d^*_i - p_i & \text{otherwise} \end{cases},$$

$$U_i = Q_{i-1} d_i + U_{i-1},$$

$$\text{for } 1 \leq i \leq n.$$

$$Q_i = Q_{i-1}\, p_i,$$

$$q_i = Q_i^{-1} \text{ in } GF(p_{i+1}),$$

$$\text{for } 1 \leq i < n.$$

Then $U = U_n$.

An algorithm based on these relations, CHRA, is also given in the Polynomial GCD and Resultant System [COG72].

CHRA assumes that one has the lists $(u_1, u_2, \ldots, u_n)$ and $(p_1, p_2, \ldots, p_n)$. This would usually indicate that one had an a priori bound on the result, computed the number of primes required, did the operations for these primes, and now needs the integer result. Hence he applies CHRA.

Frequently, however, modular algorithms do not proceed in this manner. Instead, one does the operation for prime p_i, obtains the integer result U_i based on $(u_1, \ldots, u_i)$ and $(p_1, \ldots, p_i)$, then checks if sufficiently many primes have been used. This check may be computation of $p_1 p_2 \ldots p_i$ and comparison with an a priori bound. It may also be performing some operations with U_i to see if it satisfies certain conditions and hence is the desired

result. This latter method is used, for example, when it is not possible to compute an a priori bound or when empirical data indicates that the known bounds are usually much larger than necessary.

An important advantage of the method used in CCRA is that it does not require all of the residues and primes at one time. Hence it can be used in the sequential manner indicated in the preceding paragraph.

However, these forms of applying the Chinese Remainder Theorem did not prove efficient for the modular algorithms presented in this report. The reason is that only the sign of the integer U is required, not the magnitude. Szabó has shown that (c.f. [KND68], p. 255) it is essentially necessary to use all of the residues and primes to determine the sign of the final result U; one cannot obtain in a simple way this sign from subsets $(u_1, \ldots, u_i)$ and $(p_1, \ldots, p_i)$. Hence, in order to obtain the sign of the results using the methods discussed above, one would have to actually compute the integer U, then take its sign. This is inefficient.

An alternate approach is the use of the mixed radix representation of a number suggested by H. L. Garner [KND68]. Given primes $(p_1, p_2, \ldots, p_n)$ compute constants c_{ij} such that

$$c_{ij} = p_i^{-1} \bmod p_j \quad . \tag{5}$$

Now suppose that the residues $(u_1, u_2, \ldots, u_n)$ have been computed. Let v_i be defined by

$$v_{i,1} = u_i \quad , \tag{6}$$

$$v_{i,j+1} = (v_{i,j} - v_j)c_{j,i} \pmod{p_i} \quad , \quad 1 \leq j < i, \tag{7}$$

$$v_i = \begin{cases} v_{i,i} & \text{if } v_{i,i} < p_i/2 \\ v_{i,i} - p_i & \text{otherwise} \end{cases} \quad . \tag{8}$$

Construct the lists $(v_1, v_2, \ldots, v_r)$ and $(p_1, p_2, \ldots, p_r)$ where $v_r \neq 0$ and $v_{r+1}, \ldots, v_n = 0$, or the lists () and () if all $v_i = 0$ for $1 \leq i \leq n$. Then these lists will be called the mixed radix representation of a number $V = \sum_{i=1}^{r} v_i \{\prod_{j=1}^{i-1} p_j\}$, or $V = 0$ if the lists are null.

The following theorem shows that this mixed radix representation is unique.

Theorem 2.3.3: The mapping $\Phi(v_1, v_2, \ldots, v_r) = \sum_{i=1}^{r} v_i \{\prod_{j=1}^{i-1} p_j\}$ for distinct odd primes $(p_1, p_2, \ldots, p_r)$ is one-one from the set $\{(v_1, v_2, \ldots, v_r): |v_i| < p_i/2$

for $1 \leq i \leq r\}$ onto the set $\{v: |v| < (\prod_{i=1}^{r} p_i)/2\}$.

Proof: Use induction on r. r = 1 is the identity mapping.

For r - 1, assume for each v, $|v| < (\prod_{i=1}^{r-1} p_i)/2$, that there exist unique elements $v_1, \ldots, v_{r-1}$ such that

$\Phi(v_1, \ldots, v_{r-1}) = v$.

Let $v_r = (v/(\prod_{i=1}^{r-1} p_i))$, where (x) is the closest integer to x. Then for $s = v - v_r \cdot \prod_{i=1}^{r-1} p_i$, $|s| < |(\prod_{i=1}^{r-1} p_i)/2|$. Now $|v_r| < p_r/2$ since $|v| < (\prod_{i=1}^{r} p_i)/2$.

By the induction hypothesis select $v_1, \ldots, v_{r-1}$ such that $\Phi(v_1, \ldots, v_{r-1}) = s$. Then $\Phi(v_1, \ldots, v_{r-1}, v_r)$

$$= \sum_{i=1}^{r} v_i \{\prod_{j=1}^{i-1} p_j\} = v_r \prod_{j=1}^{r-1} p_j + \sum_{i=1}^{r-1} v_i \{\prod_{j=1}^{i-1} p_j\} =$$

$v_r \prod_{j=1}^{r-1} p_j + \Phi(v_1, \ldots, v_{r-1}) = v_r \prod_{j=1}^{r-1} p_j + s = v$. Hence the mapping Φ is onto.

Now assume $\Phi(v_1, \ldots, v_r) = \Phi(v'_1, \ldots, v'_r)$, with

$|v_i|, |v'_i| < p_i/2$ for $1 \leq i \leq r$. Then $(v_r - v'_r)\prod_{i=1}^{r-1} p_i =$ $\Phi(v'_1, \ldots, v'_{r-1}) - \Phi(v_1, \ldots, v_{r-1})$. By the induction hypothesis, $|\Phi(v'_1, \ldots, v'_{r-1})|, |\Phi(v_1, \ldots, v_{r-1})| <$ $(\prod_{i=1}^{r-1} p_i)/2$, so $|(v_r - v'_r)\prod_{i=1}^{r-1} p_i| < \prod_{i=1}^{r-1} p_i$ and $|v_r - v'_r|$ < 1. Hence $v_r = v'_r$ and $\Phi(v'_1, \ldots, v'_{r-1}) = \Phi(v_1, \ldots, v_{r-1})$. By the induction hypothesis, $v'_i = v_i$ for $1 \leq i \leq r-1$. Hence $(v_1, \ldots, v_r) = (v'_1, \ldots, v'_r)$. ∎

Next it is necessary to show that $V \equiv u_i \pmod{p_i}$ for $1 \leq i \leq r$. This is done by first proving a theorem about the $v_{k,\ell}$ defined above.

Theorem 2.3.4: Let $v_{k,\ell}$ be defined as in (5) through (8). Then $v_{k,\ell}\prod_{j=1}^{\ell-1} p_j \equiv \sum_{i=1}^{\ell-1}\{-v_i\prod_{j=1}^{i-1} p_j\} + u_k \pmod{p_k}$ for $1 \leq \ell \leq k \leq r$.

Proof: Obvious for $\ell = 1$, since $v_{k,1} = u_k$. Assume for $\ell - 1$. Then $v_{k,\ell} = (v_{k,\ell-1} - v_{\ell-1})c_{\ell-1,k} \pmod{p_k}$ by (7), so $v_{k,\ell}\prod_{j=1}^{\ell-1} p_j \equiv \{\prod_{j=1}^{\ell-2} p_j\}(v_{k,\ell-1} - v_{\ell-1})p_{\ell-1}c_{\ell-1,k}$

$\equiv \{\prod_{j=1}^{\ell-2} p_j\}(v_{k,\ell-1} - v_{\ell-1}) \pmod{p_k}$ since by (5) $p_{\ell-1} \mid c_{\ell-1,k}$

$\equiv 1 \pmod{p_k}$. Using the induction hypothesis for

$v_{k,\ell-1} \prod_{j=1}^{\ell-2} p_j$ gives $v_{k,\ell} \prod_{j=1}^{\ell-1} p_j \equiv (\sum_{i=1}^{\ell-2} \{-v_{i} \prod_{j=1}^{i-1} p_j\} + u_k)$

$- v_{\ell-1} \prod_{j=1}^{\ell-2} p_j \equiv \sum_{i=1}^{\ell-1} \{-v_i \prod_{j=1}^{i-1} p_j\} + u_k \pmod{p_k}$. ∎

This theorem can now be used to prove the desired result.

Theorem 2.3.5: Let v_k be defined as in (5) through (8) and let

$$V = \sum_{i=1}^{r} v_i \{\prod_{j=1}^{i-1} p_j\}. \tag{9}$$

Then $V \equiv u_i \pmod{p_i}$ for $1 \leq i \leq n$.

Proof: Consider $V \pmod{p_k}$. $V \equiv \sum_{i=1}^{k} v_i \{\prod_{j=1}^{i-1} p_j\}$

$\pmod{p_k}$ since $v_i \{\prod_{j=1}^{i-1} p_j\} \equiv 0 \pmod{p_k}$ for $i > k$. By (8) and Theorem 2.3.4, $v_k \prod_{j=1}^{k-1} p_j \equiv v_{k,k} \prod_{j=1}^{k-1} p_j \equiv \sum_{i=1}^{k-1} -v_i \{\prod_{j=1}^{i-1} p_j\}$

$+ u_k \pmod{p_k}$ so $\sum_{i=1}^{k} v_i \{\prod_{j=1}^{i-1} p_j\} \equiv v_k \prod_{j=1}^{k-1} p_j +$

$\sum_{i=1}^{k-1} v_i \{\prod_{j=1}^{i-1} p_j\} \equiv u_k \pmod{p_k}$. ∎

Note that it is possible to write a simple algorithm to obtain the integer V, employing directly the definition (9).

What is important to this report, however, is to note that one can obtain the sign of V without actually computing V--sign(V) = sign(v_r) where (v_1, v_2, . . . , v_r) and (p_1, p_2, . . . , p_r) are the mixed radix representation of V. This follows since the magnitude of $v_r p_{r-1} p_{r-2} \cdots p_1$ is greater than the sum of the other terms in (9).

In order to implement the mixed radix representation three algorithms are required. The first computes the inverses c_{ij}. Note that list storage of these inverses is used, which makes transmission between subprograms much easier than with a variably-dimensioned Fortran array.

ALGORITHM GLPINV:

M=GLPINV(L)

Generate List of Prime Inverses

L is a list of odd prime Fortran integers, L = (p_1, . . . , p_n), with $p_1 < p_2 < \cdots < p_n$. M is the list

((p_1), (p_2, p_1, c_{12}), (p_3, p_1, c_{13}, p_2, c_{23}), (p_4, p_1, c_{14}, p_2, c_{24}, p_3, c_{34}), . . . , (p_n, p_1, c_{1n}, . . . ,

p_{n-1}, $c_{n-1,n}$)), where c_{ij} is the multiplicative inverse of p_i in $GF(p_j)$.

Description:

(1) [Initialize.] M ← 0; if L = 0, return; Q ← L.

(2) [Obtain next prime.] P ← L; ADV(q,Q); M' ← PFA(q,0).

(3) [Loop on preceding primes.] ADV(p,P); if p = q, go to (4); M' ← PFA(CRECIP(q,p),PFA(p,M')); go to (3).

(4) [Generate sublist.] M ← PFL(INV(M'),M); if Q ≠ 0, go to (2).

(5) [Finish.] M ← INV(M); return.

Computing Time: $\propto \nu^2$, where n = length(L) and ν = n + 1.

Step	n_i	t_i
1	1	~ 1
2	n	~ 1
3	$\leq n^2$	~ 1
4	n	$\propto n$
5	≤ 1	$\sim n$

Since $L(p_n) = 1$ by definition, $t_3 \sim 1$. ∎

The next algorithm converts from the residue representation to the mixed radix representation.

ALGORITHM RTMR:

V=RTMR(M,U)

Residue Representation to Mixed

Radix Representation

M is a list $(p_{i_1}, \ldots, p_{i_n})$ of primes, where p_{i_j} is the i_j-th prime on the list PRIME and $i_1 < i_2 < \ldots < i_n$. U is a list $(u_1, \ldots, u_n)$, with $u_j \in GF(p_{i_j})$. Let $P_k = \prod_{j=1}^{k-1} p_{i_j}$ for $1 \leq k \leq n+1$, $P = P_{n+1}$. Let A be the unique integer such that $|A| < P/2$ and $A \equiv u_j \pmod{p_{i_j}}$ for $1 \leq j \leq n$. If $A = 0$, then $V = ()$. Otherwise, V is the unique list $(v_1, \ldots, v_r)$ such that $A = \sum_{j=1}^{r} v_j P_j$, $r \leq n$, $|v_j| < p_{i_j}/2$ for $1 \leq j \leq r$, and $v_r \neq 0$.

Method:

Let $v_{i,1} = u_i$ and $v_{i,j+1} = (v_{i,j} - v_j)c_{j,i}$ for $1 \leq j < i$, where $c_{j,i} = p_j^{-1}$ and arithmetic is performed in $GF(p_i)$. Then $v_i = v_{i,i}$ if $v_{i,i} < p_i/2$ and $v_i = v_{i,i}$

- p_i otherwise. The $c_{j,i}$ are obtained from the list PRINV.

Description:

(1) [Initialize.] V ← 0; if U = 0, return; M' ← M; U' ← U; V' ← 0; L ← PRINV.

(2) [Obtain next pair.] ADV(p,M'); ADV(u,U').

(3) [Locate prime on PRINV.] ADV(L_1,L); ADV(q,L_1); if q ≠ p, go to (3); if V = 0, go to (8).

(4) [Initialize inner loop.] V' ← INV(V); M'' ← M; V'' ← V'.

(5) [Obtain next pair.] ADV(v,V''); u ← CDIF(p,u,v); ADV(q,M'').

(6) [Locate inverse.] ADV2(q',c,L_1); if q' ≠ q, go to (6).

(7) [Compute next product.] u ← CPROD(p,u,c); if V'' ≠ 0, go to (5).

(8) [Convert to symmetric modular residue.] v ← u - p; if v + u < 0, v ← u; V ← PFA(v,INV(V')); if M' ≠ 0, go to (2).

(9) [Delete high order zeros.] If FIRST(V) ≠ 0, go to (10); DECAP(v,V); if V ≠ 0, go to (9).

(10) [Obtain inverse.] V ← INV(V); return.

Computing Time: ∿ 1 for n = 0; otherwise ∝ mn, where $m = i_n$.

Step	n_i	t_i
1	1	~ 1
2	n	~ 1
3	m	~ 1
4	n	$\underline{\propto}\ n$
5	$\underline{<}\ n^2$	~ 1
6	$\underline{<}\ mn$	~ 1
7	$\underline{<}\ n^2$	~ 1
8	n	$\underline{\propto}\ n$
9	$\underline{<}\ n$	~ 1
10	1	$\underline{\propto}\ n$

The third algorithm simply locates the high order digit v_r in a mixed radix representation in order to determine the sign of the number represented.

ALGORITHM SMRR:

s=SMRR(V)

Sign of a Number in Mixed

Radix Representation

V is a list of the mixed radix representation of a number N: V = () if N = 0; otherwise, $V = (v_1, \ldots, v_r)$, $v_r \neq 0$, for $N = \sum_{i=1}^{r} \{ v_i \prod_{j=1}^{i-1} p_j \}$ where the p_j are positive odd Fortran primes such that $p_1 < \ldots < p_{r-1}$

and $|v_j| < p_j/2$. s is the sign of N, a Fortran integer.

Description:

(1) [Initialize.] $s \leftarrow 0$; if $V = 0$, return; $V' \leftarrow V$.

(2) [Find last element.] ADV(v,V'); if $V' \neq 0$, go to (2).

(3) [Obtain sign.] $s \leftarrow 1$; if $v < 0$, $s \leftarrow -1$; return.

Computing Time: $\sim \rho$, where r = length(V) and $\rho = r + 1$.

Proof:

Step	n_i	t_i
1	1	~ 1
2	r	~ 1
3	≤ 1	~ 1

Although these three algorithms are all that are actually required for this system, it is interesting to study a fourth algorithm which performs the conversion from mixed radix representation to β-radix representation.

In order to derive the computing time of this algorithm, it is necessary to compute a bound on the magnitude of the integer represented in mixed radix form.

Theorem 2.3.6: Let $M = (p_1, p_2, \ldots, p_r)$ and $V = (v_1, v_2, \ldots, v_r)$ be the mixed radix representation of a

number N. Assume $3 \leq p_1, p_2, \ldots, p_r < \beta$. Then

$$\frac{3}{2}^{r-1} < |N| < \beta^r .$$

Proof: $|N| = |\sum_{i=1}^{r} v_i \prod_{j=1}^{i-1} p_j| \geq \prod_{j=1}^{r-1} p_j -$

$|\sum_{i=1}^{r-1} v_i \prod_{j=1}^{i-1} p_j| > \prod_{j=1}^{r-1} p_j - (1/2)\prod_{j=1}^{r-1} p_j = (1/2)\prod_{j=1}^{r-1} p_j$

$> (1/2)3^{r-1}$, by Theorem 2.3.3 . Also by Theorem 2.3.3,

$|N| < (1/2)\prod_{j=1}^{r} p_j < (1/2)\beta^r$. ∎

The following algorithm converts from mixed radix representation to β-radix representation.

ALGORITHM MRTI:

N=MRTI(M,V)

Mixed Radix to L-Integer

N is the L-integer whose mixed radix representation is given by M and V. N = 0 if V = (). Otherwise, V = $(v_1, v_2, \ldots, v_r)$, $v_r \neq 0$, and M = $(p_1, p_2, \ldots, p_r)$. Then $N = \sum_{i=1}^{r} v_i \prod_{j=1}^{i-1} p_j$.

Description:

(1) [Initialize.] N ← 0; if V = 0, return; $\overline{V} \leftarrow V$; $\overline{M} \leftarrow$ M; ADV(v,$\overline{V}$); N ← PFA(v,0); c ← PFA(1,0); go to (3).

(2) [Obtain next pair.] ADV(p,$\overline{M}$); ADV(v,$\overline{V}$); IMFI(c,p);

d ← BORROW(c); IMFI(d,v); e ← ISUM(d,N); erase d,N; N ← e.

(3) [Check for end.] If $\overline{V} \neq 0$, go to (2); erase c; return.

Computing Time: $\sim L(N)^2$.

Proof:

Step	n_i	t_i
1	1	~ 1
2	r-1	$\propto r$
3	r	~ 1

Let $T = \sum_{i=1}^{3} n_i t_i$. Then from the table $T \propto r^2$ for $r > 0$.

The time for the j-th execution of step (2) is $\sim j$ since $3^j \leq c_j < \beta^j$, where c_j is the value of c after the j-th execution of step (2). So the computing time $T \sim r^2$ for $r > 0$.

Also, $3^{r-1}/2 < |N| < \beta^r$ so $T \sim L(N)^2$.

In comparing this algorithm to CHRA, one sees that the codominance equivalence classes of the computing times are the same. However, the following chart shows that empirically MRTI is somewhat faster.

These times were obtained on a PDP-10 using primes p_i in the range $2^{31} + 1 \leq p_i < 2^{33}$ with random residues $u_i \in GF(p_i)$ for $1 \leq i \leq r$. The time unit is seconds (s).

r	MRRE (s)	MRTI (s)	Total (s)	CHRA (s)	Ratio: CHRA/Total
10	.084	.233	.317	.533	1.68
20	.383	.900	1.28	1.67	1.30
30	.700	1.60	2.30	3.67	1.60
40	1.12	2.92	4.04	5.58	1.38
50	1.77	3.82	5.59	8.78	1.57
60	2.45	5.86	8.31	11.9	1.43
70	3.42	7.30	10.7	15.8	1.48
80	4.63	10.1	14.7	20.7	1.41
90	6.12	12.4	18.5	25.8	1.39
100	6.75	15.3	22.1	31.6	1.43

CHAPTER 3: STURM SEQUENCES

3.1 Theoretical background

The basic tool for root isolation and refinement is the Sturm sequence. Sturm's theorem applies it to finding the number of real roots of a real polynomial in an interval, the Routh-Hurwitz theorems apply it to finding the number of roots of a complex polynomial in the upper and lower half-planes.

Heindel [HEL70] defines what in this report shall be called a restricted Sturm sequence as follows. Let $F_1(x)$ and $F_2(x)$ be two continuous real-valued functions of a single real variable x. A restricted 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 for all intervals (a,b], $a < b$, and $s_i(x) = \text{sign}(F_i(x))$:

(1) if $a \leq y \leq b$ and $F_1(y) = 0$, then there exists an $\varepsilon > 0$ such that $s^* = s_2(x) \neq 0$ is constant for $y - \varepsilon < x < y + \varepsilon$, while $s_1(x) = -s^*$ for $y - \varepsilon < x < y$ and $s_1(x) = s^*$ for $y < x < y + \varepsilon$;

(2) if $a \leq y \leq b$, $1 < i < r$, and $F_i(x) = 0$, then $s_{i-1}(x) = -s_{i+1}(x) \neq 0$;

(3) $F_r(x) \neq 0$ for all $a \leq x \leq b$.

He then states and proves the following generalized Sturm's theorem.

Theorem 3.1.1: Let $F_1(x)$ be a continuous real-valued function of a single real variable, $F_1, F_2, \ldots, F_r$ a restricted Sturm sequence for F_1 and F_2, and let $V(x)$ 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 the number of distinct real roots of $F_1(x)$ in the interval $(a,b]$ is $V(a) - V(b)$.

Rather than reproducing the proof here, an intuitive discussion based on Wilf [WIH62] will be given.

Let $F_1(x), F_2(x), \ldots, F_r(x)$ be a restricted Sturm sequence for $F(x)$, with $F_1(x) = F(x)$ and $F_2(x) = F'(x)$. (It is possible to generate such a sequence if F meets certain conditions specified later.) Consider how $V(x)$ behaves as x traverses the interval $(a,b]$. The only places that $V(x)$ can change are at points where one of the functions in the sequence changes sign, that is, vanishes. Suppose such a point is x_0. Now by definition, $F_r(x_0) \neq 0$. So let $F_k(x_0) = 0$ for $2 \leq k < r$. Based on the definition

of a restricted Sturm sequence, the following configurations are possible.

Left of x_0			At x_0			Right of x_0		
F_{k-1}	F_k	F_{k+1}	F_{k-1}	F_k	F_{k+1}	F_{k-1}	F_k	F_{k+1}
+	+	−	+	0	−	+	−	−
+	−	−	+	0	−	+	+	−
−	+	+	−	0	+	−	−	+
−	−	+	−	0	+	−	+	+

Note, however, that the number of variations in sign of the sequence $F_{k-1}(x)$, $F_k(x)$, $F_{k+1}(x)$ is the same at x_0^-, x_0, and x_0^+. Hence, if x_0 is a zero of $F_k(x)$ for $2 \leq k \leq r$, $V(x_0^-) = V(x_0^+)$.

Now consider a point x_0 which is a zero of $F_1(x) = F(x)$. By definition, $F_2(x_0) = F'(x_0) \neq 0$, so the following are the only possible combinations.

Left of x_0		At x_0		Right of x_0	
F_1	F_2	F_1	F_2	F_1	F_2
+	−	0	−	−	−
−	+	0	+	+	+

In this case, a sign variation is lost as x passes through

x_0. Hence $V(x_0^+) = V(x_0) = V(x_0^-) - 1$.

Therefore, as x goes from a to b in (a,b], V(x) decreases by the number of zeros of F(x) in the interval. (Note that the right hand end point b is included because, as shown in the preceding paragraph, $V(x_0) = V(x_0^+)$.)

Hence, if n is the number of zeros of F in (a,b],

$$n = V(a) - V(b) \quad . \tag{1}$$

Using $\pm\infty$ as the interval bounds, one obtains the number of real zeros of F.

Now consider the problem of determining the number of zeros of a Gaussian polynomial in the upper and lower half-planes. The following theorem of Hurwitz as presented in Marden [MAM66] gives a method for doing this.

Theorem 3.1.2: Let $G(z) = G_1(z) + iG_2(z)$ where $G_1(z)$ and $G_2(z)$ are real polynomials with $G_2 \neq 0$. As the point $z = x$ moves on the real axis from $-\infty$ to $+\infty$, let a be the number of real zeros of $G_1(z)$ at which $H(x) = G_1(x)/G_2(x)$ changes from - to + and b the number of real zeros of $G_1(z)$ at which H(x) changes from + to -. If G(z) has no real zeros, p zeros in the upper half-plane, and q zeros in the lower half-plane, then for n = deg(G)

$$p = (1/2)[n + (b - a)] \ , \tag{2}$$

$$q = (1/2)[n - (b - a)] \quad . \tag{3}$$

What is required, then, is a method for determining $b - a$. Routh shows that Sturm sequences can be used to do this.

Following Wilf [WIH62], let (a,b) be a finite or infinite interval of the real axis, and let $F_1, F_2, \ldots,$ F_r be r continuous functions defined on (a,b). $F_1, F_2,$ $\ldots, F_r$ is a Sturm sequence for F_1 and F_2 on (a,b) if the following hold:

(1) at a zero x_0 of $F_k(x)$, $F_{k+1}(x_0)$ and $F_{k-1}(x_0)$ have opposite signs and are not zero, $1 < k < r$, $a < x_0 < b$;

(2) $F_r(x) \neq 0$ for $a < x < b$.

Routh has developed the following theorem as presented in Marden [MAM66].

Theorem 3.1.3: Let $G(z) = G_1(z) + iG_2(z)$, where $G_1(z)$ and $G_2(z)$ are real polynomials and $G_2(z) \neq 0$, be a polynomial which has no real zeros, p zeros in the upper half-plane, and q zeros in the lower half-plane. Let $G_1(x), G_2(x), \ldots, G_r(x)$ be a Sturm sequence for $G_1(x)$ and $G_2(x)$. Then for $n = \deg(G)$

$$p = (1/2)[n + V(+\infty) - V(-\infty)] , \tag{4}$$

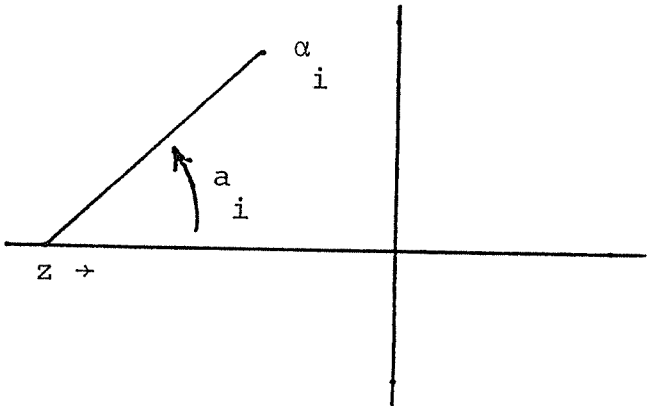
$$q = (1/2)[n - V(+\infty) + V(-\infty)] . \tag{5}$$

Marden proves both Theorems 3.1.2 and 3.1.3 in detail in [MAM66], and as with Theorem 3.1.1, the proofs will not be repeated here--rather, an intuitive discussion of them will be given, again based on Wilf [WIH62].

Consider $G(z)$ written in the form $\prod_{i=1}^{n} (z - \alpha_i)$, where the α_i are the roots of $G(z)$. (Recall that $G(z)$ has no real roots so all α_i are complex.) Then

$$\arg\{G(z)\} = \sum_{i=1}^{n} \arg\{z - \alpha_i\} . \tag{6}$$

Now observe what happens to $a_i = \arg\{z - \alpha_i\}$ as z traverses the real axis from $-\infty$ to $+\infty$. If α_i is in the upper half-plane,



then a_i goes from 0 to π. If α_i is in the lower half-plane, a_i goes from 0 to $-\pi$. Thus, by (6), $\arg\{G(z)\}$ increases by

π for each root in the upper half-plane and decreases by π for each root in the lower half-plane. Denoting the net change in arg $\{G(z)\}$ as z goes from $-\infty$ to $+\infty$ by Δ_R,

$$\Delta_R = (p - q)\pi \quad . \tag{7}$$

Since G(z) has no real zeros,

$$p + q = \deg(G) = n \quad . \tag{8}$$

Solving (7) and (8) simultaneously for p and q gives

$$p = n/2 + \Delta_R/(2\pi) \quad , \tag{9}$$

$$q = n/2 - \Delta_R/(2\pi) \quad . \tag{10}$$

If G(z) is evaluated at a real point x, then

$$G(x) = G_1(x) + iG_2(x)$$

$$= |G(x)|e^{i\Theta(x)} \tag{11}$$

where

$$\Theta(x) = \cot^{-1} H(x) \tag{12}$$

for $H(x) = G_1(x)/G_2(x)$ as in Theorem 3.1.2 . Hence

$$\arg\{G(x)\} = \Theta(x) \quad . \tag{13}$$

In order to determine p and q from (9) and (10), it is necessary to compute Δ_R. Suppose that x_i and x_j are zeros of $G_1(x)$, with $x_i < x_j$ and $G_1(x) \neq 0$ for $x_i < x < x_j$. Since G(z) has no real zeros, $G_2(x_i)$, $G_2(x_j) \neq 0$. Thus,

$$H(x_i) = H(x_j) = 0. \tag{14}$$

If $H(x)$ changes sign as x traverses the interval (x_i, x_j), only the following combinations are possible.

G_1	G_2		
+	+	0	−
+	−	0	+
−	+	0	−
−	−	0	+

(15)

Combining (14) and (15), the change in $H(x)$ and the corresponding change in argument $\Delta_{i,j}$ for x traversing $[x_i, x_j]$ are as follows.

(16)

$H(x)$	$\Delta_{i,j}$
$0/+ \to +/+ \to +/0 \to +/- \to 0/-$	$\pi/2 \to 0 \to -\pi/2$
$0/- \to +/- \to +/0 \to +/+ \to 0/+$	$-\pi/2 \to 0 \to \pi/2$
$0/+ \to -/+ \to -/0 \to -/- \to 0/-$	$\pi/2 \to \pi \to 3\pi/2$
$0/- \to -/- \to -/0 \to -/+ \to 0/+$	$3\pi/2 \to \pi \to \pi/2$

Hence if $H(x)$ changes from + to − then $\Delta_{i,j} = -\pi$, and if $H(x)$ changes from − to + then $\Delta_{i,j} = \pi$.

Let $S(x) = \text{sign}\{H(x)\}$. Then on the interval (x_i, x_j),

$$\Delta_{i,j} = (\pi/2)\{S(x_j^-) - S(x_i^+)\}. \tag{17}$$

If x_1 is the algebraically smallest root of $G_1(x)$, then on the interval $(-\infty, x_1^-)$,

$$\Delta_{0,1} = (\pi/2)S(x_1^-) \quad , \tag{18}$$

since $G_1(x) \neq 0$ and $\text{sign}\{G_1(x)\}$ cannot change. Similarly, if x_n is the algebraically largest root of $G_1(x)$ then on $(x_n^+, +\infty)$,

$$\Delta_{n,n+1} = (-\pi/2)S(x_n^+) \quad . \tag{19}$$

Adding up these quantities for all of the roots gives

$$\Delta_R = \sum_{k=0}^{n} \Delta_{k,k+1} \tag{20}$$

$$= (\pi/2)\left[\sum_{k=1}^{n-1} \{S(x_{k+1}^-) - S(x_k^+)\} + S(x_1^-) - S(x_n^+)\right]$$

$$= (\pi/2)\sum_{k=1}^{n} \{S(x_k^-) - S(x_k^+)\} \quad . \tag{21}$$

Thus the sum increases by π if $H(x)$ goes from + to - as x goes from x_k^- to x_k^+, and decreases by π for - to +. Hence Δ_R/π is the excess of + to - changes over - to + changes as x goes from $-\infty$ to $+\infty$.

In order to compute this excess, recall that the only

place V(x) can change is at x_0 a zero of $G_1(x)$. Consider the possible combinations.

x_0^-		x_0^+	
G_1	G_2	G_1	G_2
+	−	−	−
+	+	−	+
−	−	+	−
−	+	+	+

(22)

These give the following values for H(x) and V(x).

H(x)		$V(x_0^+) - V(x_0^-)$
x_0^-	x_0^+	
−	+	−1
+	−	+1
+	−	+1
−	+	−1

(23)

Note, then that the excess of + to − changes over − to + changes is given by $V(+\infty) - V(-\infty)$. Thus,

$$p = (1/2)\{n + V(+\infty) - V(-\infty)\} \quad , \tag{24}$$

$$q = (1/2)\{n - V(+\infty) + V(-\infty)\} \quad . \tag{25}$$

Hence, given an integral polynomial P(x) one can compute a restricted Sturm sequence with P(x), P'(x) and use

V(x) to compute the number of zeros of P(x) in (a,b] or the number of real zeros of P(x). Given a Gaussian polynomial $G(z) = G_1(z) + iG_2(z)$ with no real zeros one can compute a Sturm sequence with $G_1(z), G_2(z)$ and use V(x) to compute the number of zeros of G(z) in the upper and lower half-planes.

3.2 Generation of Sturm sequences

This section develops algorithms for two methods of generating the Sturm sequences discussed in Section 3.1--integer and modular.

Consider polynomials A_1 and A_2 over S, where $A_1, A_2 \neq 0$. Let $A_1, A_2, \ldots, A_{r+1}$ be a sequence of polynomials over S defined by

$$e_i A_i = Q_i A_{i+1} + f_i A_{i+2}, \quad 1 \leq i < r, \tag{1}$$

where $e_i, f_i \in S$; $A_{i+2} = 0$ or $\deg(A_{i+2}) < \deg(A_{i+1})$; and $A_{r+1} = 0$ is the first identically zero polynomial in the sequence. Then this sequence is called a polynomial remainder sequence (p.r.s.) for A_1 and A_2.

If S is a field, then it is possible to set all e_i, f_i to units and do the division directly. In this system, however, S is the integers, and hence not a field. Thus suitable e_i must be computed, and the method used relates to the technique of pseudo-division. Given two integral polynomials A and $B \neq 0$, in most cases there are not two integral polynomials Q and R with $R = 0$ or $\deg(R) < \deg(B)$ such that $A = BQ + R$. However, if $b = \mathrm{ldcf}(B)$, $m = \deg(A)$, and $n = \deg(B)$, then there are two integral polynomials Q* and R* with $R^* = 0$ or $\deg(R^*) < \deg(B)$ such

that $b^{m-n+1} A = BQ^* + R^*$. Q^* and R^* are then the pseudo-quotient and pseudo-remainder.

Given non-zero integral polynomials A_1 and A_2, this pseudo-division can be used to generate an integral p.r.s. for A_1 and A_2. Let $a_i = \mathrm{ldcf}(A_i)$, $n_i = \deg(A_i)$, and

$$e_i = a_{i+1}^{\delta_i + 1}, \text{ where } \delta_i = n_i - n_{i+1}, \tag{2}$$

for $1 \leq i \leq r - 2$. Then selecting any f_i such that $f_i \mid (e_i A_{i-1} - Q_i A_{i+1})$ will give a p.r.s. over the integers for A_1 and A_2.

A polynomial $A(x) = \sum_{i=0}^{n} a_i x^i$ is primitive just in case $\gcd(a_0, a_1, \ldots, a_n) = 1$. A primitive p.r.s., then, is a p.r.s. in which A_i is primitive for $3 \leq i \leq r$.

If $e_i f_i < 0$ for $1 \leq i \leq r - 2$, then the p.r.s. defined by (1) is called a negative p.r.s.

If e_i is as in (2) and $f_i = -\mathrm{sign}(e_i) \cdot \mathrm{cont}(e_i A_i - Q_i A_{i+1})$ then the resulting p.r.s. will be a negative primitive p.r.s. Heindel [HEL70] states and proves the following theorem about this negative primitive p.r.s.

Theorem 3.2.1: Let A_1 be a primitive, positive square-free univariate polynomial of positive degree with integer coefficients, $A_2 = pp(A_1')$, and $A_1, A_2, \ldots, A_r$ be a negative primitive p.r.s. Then $A_1, A_2, \ldots, A_r$ is a restricted Sturm sequence for A_1 and A_1'.

A similar result can be shown for Sturm sequences.

Theorem 3.2.2: Let A_1, A_2 be integral polynomials with $\deg(A_1) \geq \deg(A_2)$ and $\gcd(A_1, A_2) = 1$. Let $A_1, A_2, \ldots, A_r$ be a negative primitive p.r.s. Then $A_1, A_2, \ldots, A_r$ is a Sturm sequence for A_1 and A_2.

Proof: $e_i A_i = Q_i A_{i+1} + f_i A_{i+2}$ where $e_i f_i < 0$, so if $A_{i+1}(x_0) = 0$ then $A_i(x_0)$ and $A_{i+2}(x_0)$ have opposite signs unless they are zero. However, by the recurrence relation this would imply $A_k(x_0) = 0$ for all k, $1 \leq k \leq r$, which is impossible since $A_r = \pm \gcd(A_1, A_2) = \pm 1$. Hence condition (1) of the definition of Sturm sequence is satisfied. Since $\gcd(A_1, A_2) = 1$, it follows that $A_r(x) \neq 0$ for any x in $(a,b]$ and hence condition (2) is satisfied. ∎

The following algorithm computes a negative primitive p.r.s. and hence can be used to generate a Sturm sequence or restricted Sturm sequence.

There is one aspect to note about its output. The algorithms in this report need only $V(\pm\infty)$ and/or $V(0)$. Computing $V(\pm\infty)$ requires only the signs of the leading coefficients and the degrees, and computing $V(0)$ requires only the signs of the trailing coefficients. Hence the algorithm constructs lists of these degrees and signs, rather than storing each polynomial in the sequence in its entirety.

ALGORITHM CZNPRS:

CZNPRS(A,B,I,N,S,T)

Complex Zero System, Negative

Primitive P.R.S.

A, B, and I are inputs; N, S, and T are outputs. A and B are non-zero univariate integral polynomials, with $\deg(A) \geq \deg(B)$. I = 0 or I = 1, a Fortran integer. Let $A_1, A_2, \ldots, A_{r+1} = 0$ be any negative p.r.s. for A and B. Let $n_i = \deg(A_i)$, $s_i = \text{sign}(\text{ldcf}(A_i))$ and $t_i = \text{sign}(\text{trcf}(A_i))$. Then N = $(n_1, n_2, \ldots, n_r)$, S = $(s_1, s_2, \ldots, s_r)$, T = $(t_1, t_2, \ldots, t_r)$ if I = 1 and T = 0 if I = 0.

Method:

A negative primitive p.r.s. is computed.

Description:

(1) [Initialize.] $A_1 \leftarrow$ PSPP(A_1); $A_2 \leftarrow$ PSPP(A_2); $N \leftarrow 0$; $S \leftarrow 0$; $T \leftarrow 0$; $n_1 \leftarrow$ PDEG(A_1); $n_2 \leftarrow$ PDEG(A_2); $s_1 \leftarrow$ PSIGN(A_1); $s_2 \leftarrow$ PSIGN(A_2); if $I \neq 0$, ($t_1 \leftarrow$ PTCS(A_1); $t_2 \leftarrow$ PTCS(A_2)).

(2) [Compute next term.] $A_3 \leftarrow$ PSREM(A_1, A_2); erase A_1; if $A_3 = 0$, go to (3); $A_1 \leftarrow A_2$; $A_2 \leftarrow$ PSPP(A_3); erase A_3; if $s_2 > 0$ or MOD($n_1 - n_2$, 2) > 0, ($A_3 \leftarrow$ PNEG(A_2); erase A_2; $A_2 \leftarrow A_3$); $N \leftarrow$ PFA(n_1, N); $S \leftarrow$ PFA(s_1, S); if $I \neq 0$, $T \leftarrow$ PFA(t_1, T); $n_1 \leftarrow n_2$; $n_2 \leftarrow$ PDEG(A_2); $s_1 \leftarrow s_2$; $s_2 \leftarrow$ PSIGN(A_2); if $I \neq 0$, ($t_1 \leftarrow t_2$; $t_2 \leftarrow$ PTCS(A_2)); go to (2).

(3) [Finish.] Erase (A_2); $N \leftarrow$ INV(PFA(n_2, PFA(n_1, N))); $S \leftarrow$ INV(PFA(s_2, PFA(s_1, S))); if $I \neq 0$, $T \leftarrow$ INV(PFA(t_2, PFA(t_1, T))); return.

Computing Time: $\leq \mu^3 \nu^{1+k} L(\mu d)^2$, where $m = \deg(A)$, $\mu = m + 1$, $n = \deg(B)$, $\nu = n + 1$, $m \geq n$, and $|A|_\infty$, $|B|_\infty \leq$

d; k = 0 if the p.r.s. is normal, k = 1 otherwise.

Proof: Let $\overline{A}_{i+2} = \text{prem}(A_i, A_{i+1})$. The time to compute $\overline{A}_3$ is $\preceq \mu^3 L(\mu d)^2$. $|\overline{A}_3|_\infty \leq (\mu d)^{m-n+2}$ so the time to compute A_3 from $\overline{A}_3$ is $\preceq \nu(m-n+2)^2 L(\mu d)^2 \preceq$ $\mu^2 \nu L(\mu d)^2$.

For $2 \leq i < r$, $|A_i|_\infty, |A_{i+1}|_\infty \leq (\mu d)^{m+n}$ so the time to compute $\overline{A}_{i+2}$ from A_i and A_{i+1} is $\preceq (n_i + 1)(\delta_i + 1)^2 (L((\mu d)^{m+n}))^2 \preceq \mu^2 \nu L(\mu d)^2 \delta_i^2$, where $\delta_i = n_i - n_{i+1}$. Note that $\delta_i + 1 \sim \delta_i$ since $\delta_i > 0$. Also, $|\overline{A}_{i+2}|_\infty \leq ((\mu d)^{m+n})^{\delta_i+2}$ so the time to compute A_{i+2} from $\overline{A}_{i+2}$ for $2 \leq i < r-1$ is $\preceq (n_{i+2}+1)^2 \mu^2 L(\mu d)^2 \delta_i^2 \preceq$ $\mu^2 \nu L(\mu d)^2 \delta_i^2$.

Now $\sum_{i=2}^{r-1} \mu^2 \nu L(\mu d)^2 \delta_i^2 = \mu^2 \nu L(\mu d)^2 \sum_{i=2}^{r-1} \delta_i^2$. In general, $\sum_{i=2}^{r-1} \delta_i^2 \leq (\sum_{i=2}^{r-1} \delta_i)^2 = (n_2 - n_r)^2 \leq n_2^2$. Hence $\mu^2 \nu L(\mu d)^2 \delta_i^2 \preceq \mu^2 \nu^3 L(\mu d)^2$ and the total time is $\preceq$

$$\mu^3 L(\mu d)^2 + \mu^2 \nu^2 L(\mu d)^2 \underline{\propto} \mu^3 \nu^2 L(\mu d)^2 .$$

If the p.r.s. is normal and each $\delta_i = 1$, then $\sum_{i=2}^{r-1} \delta_i = \sum_{i=2}^{r-1} 1 \leq n$. Hence $\mu^2 \nu L(\mu d)^2 \sum_{i=2}^{r-1} \delta_i \underline{\propto} \mu^2 \nu^2 L(\mu d)^2$ and the total time is $\mu^3 L(\mu d)^2 + \mu^2 \nu^2 L(\mu d)^2 \underline{\propto} \mu^3 \nu L(\mu d)^2$. ∎

Modular generation of negative p.r.s.'s is more involved, and requires some additional definitions and theorems.

Consider the polynomial remainder sequences defined by $e_i A_i = Q_i A_{i+1} + f_i A_{i+2}$ with $A_{i+2} = 0$ or $\deg(A_{i+2}) < \deg(A_{i+1})$. A <u>natural p.r.s.</u> is a p.r.s. satisfying $e_i = f_i = 1$ for all i. A <u>positive p.r.s.</u> is a p.r.s. satisfying $e_i f_i > 0$ for all i. A <u>negative p.r.s.</u> is a p.r.s. satisfying $e_i f_i < 0$ for all i.

A <u>pseudo-subresultant p.r.s.</u> is a p.r.s. $B_1, B_2, \ldots, B_r, B_{r+1} = 0$ such that, for $3 \leq i \leq r$, $B_i = \mathcal{S}_{i-1}(B_{n-1}, B_1, B_2)$ or $B_i = -\mathcal{S}_{i-1}(B_{n-1}, B_1, B_2)$.

Recall that $\mathcal{S}_i(B_1, B_2)$ is the i-th subresultant of B_1 and B_2 [BRW71b]. A theorem is now presented which gives the relationship between a pseudo-subresultant p.r.s. and a natural p.r.s.

Theorem 3.2.3: If $A_1, A_2, \ldots, A_r, A_{r+1} = 0$ is a natural p.r.s., $a_i = \mathrm{ldcf}(A_i)$, $n_i = \deg(A_i)$, $\delta_i = n_i - n_{i+1}$, $b_1 = b_2 = 1$,

$$b_k = \{\Pi_{i=2}^{k-2} a_i^{\delta_{i-1}+\delta_i}\} a_{k-1}^{\delta_{k-2}+1} \qquad (1)$$

for $3 \leq k \leq r+1$, and $D_k = b_k A_k$, then $D_1, D_2, \ldots, D_r$, $D_{r+1} = 0$ is a pseudo-subresultant p.r.s.

Proof: In the Fundamental Theorem of Polynomial Remainder Sequences [BRW71b] let $e_i = f_i = 1$. Then

$$\left| \mathcal{S}_{n_{k-1}}(A_{-1}, A_1, A_2) \right| = |b_k A_k| = |D_k| . \blacksquare$$

Let D_k be defined as in Theorem 3.2.3. D_k is called the natural pseudo-subresultant p.r.s. of A_1 and A_2.

The following algorithm uses Theorem 3.2.3 to compute a natural pseudo-subresultant p.r.s. over GF(p). For reasons explained in connection with CZNPRS, this algorithm saves

only degrees and leading and trailing coefficients rather than the entire p.r.s.

ALGORITHM CZCPRS:

CZCPRS(p,A,B,I,N,C,D)

Complex Zero System, Congruence Natural

Pseudo-Subresultant P.R.S.

p, A, B, and I are inputs. N, C, and D are outputs. p is a prime number, a Fortran integer. A and B are non-zero univariate polynomials over GF(p), with $\deg(A) \geq \deg(B)$. I = 0 or I = 1, a Fortran integer. Let $S_1, S_2, \ldots, S_r, S_{r+1} = 0$ be the natural pseudo-subresultant p.r.s. with $S_1 = A$ and $S_2 = B$. Let $n_i = \deg(S_i)$, $c_i = \mathrm{ldcf}(S_i)$, and $d_i = \mathrm{trcf}(S_i)$. Then $N = (n_1, n_2, \ldots, n_r)$, $C = (c_1, c_2, \ldots, c_r)$, $D = (d_1, d_2, \ldots, d_r)$ if I = 1 and D = 0 if I = 0.

Method:

Let $A_1, A_2, \ldots, A_r$ be the natural p.r.s. Use the recurrence relation $b_{k+1} = a_{k-1}^{\delta_{k-1}-1} a_k^{\delta_{k-1}+1} b_k$ for $k \geq 3$ with $b_1 = b_2 = 1$, $b_3 = a_2^{\delta_1+1}$. $S_i = b_i A_i$ is then the

desired sequence.

Description:

(1) [Initialize.] $A_1 \leftarrow$ BORROW(A); $A_2 \leftarrow$ BORROW(B); $n_1 \leftarrow$ CPDEG(A_1); $n_2 \leftarrow$ CPDEG(A_2); $N \leftarrow$ PFA(n_2,PFA(n_1,0)); $a_1 \leftarrow$ CPLDCF(A_1); $a_2 \leftarrow$ CPLDCF(A_2); $C \leftarrow$ PFA(a_2,PFA(a_1,0)); $D \leftarrow$ 0; if $I \neq 0$, $D \leftarrow$ PFA(CPTRCF(A_2),PFA(CPTRCF(A_1),0)); $b_2 \leftarrow 1$; $k \leftarrow 2$; $e_1 \leftarrow 1$.

(2) [Compute next term.] $A_3 \leftarrow$ CPREM(p,A_1,A_2); erase A_1; if $A_3 = 0$, go to (3); $\delta_1 \leftarrow n_1 - n_2$; if $k > 2$, $e_1 \leftarrow$ CPOWER(p,a_1,$\delta_1 - 1$); $e_2 \leftarrow$ CPOWER(p,a_2,$\delta_1 + 1$); $e_1 \leftarrow$ CPROD(p,e_1,e_2); $b_3 \leftarrow$ CPROD(p,e_2,b_2); $n_3 \leftarrow$ CPDEG(A_3); $a_3 \leftarrow$ CPLDCF(A_3); $c_3 \leftarrow$ CPROD(p,a_3,b_3); $N \leftarrow$ PFA(n_3,N); $C \leftarrow$ PFA(c_3,C); if $I \neq 0$, ($d_3 \leftarrow$ CPROD(p,CPTRCF(A_3),b_3); $D \leftarrow$ PFA(d_3,D)); $A_1 \leftarrow A_2$; $A_2 \leftarrow A_3$; $n_1 \leftarrow n_2$; $n_2 \leftarrow n_3$; $a_1 \leftarrow a_2$; $a_2 \leftarrow a_3$; $b_2 \leftarrow b_3$; $k \leftarrow k + 1$; go to (2).

(3) [Finish.] Erase A_2; $N \leftarrow$ INV(N); $C \leftarrow$ INV(C); $D \leftarrow$ INV(D); return.

Computing Time: $\propto \mu\nu$, where $m = \deg(A)$, $n = \deg(B)$, $\mu = m + 1$, and $\nu = n + 1$.

Proof:

Step	n_i	t_i
1	1	$\underline{\propto}\ \mu$
2	r-1	
3	1	$\sim r$

Let $S_1, S_2, \ldots, S_r, S_{r+1} = 0$ be the p.r.s. and $\nu_j = \deg(S_j)$. Then the time for the j-th execution of step (2) is $\sim \nu_{j+1}(\nu_j - \nu_{j+1} + 1)$. Summing over all executions,

$$\sum_{j=1}^{r-1} \nu_{j+1}(\nu_j - \nu_{j+1} + 1) \leq \nu_2 \sum_{j=1}^{r-1} (\nu_j - \nu_{j+1} + 1) = \nu(\nu_1 - \nu_r + r - 1) \leq \nu(\nu_1 + n + 1) = \nu(\mu + \nu) \sim \mu\nu. \blacksquare$$

In most cases the p.r.s. will have to be computed over $GF(p_i)$ for several primes p_i in order to construct it over the integers. Hence the following algorithm applies CZCPRS a designated number of times and combines the results into a convenient structure. Note that the number of applications of CZCPRS is specified, not the primes with which it is to be applied. In general the primes used are taken sequentially off the list generated in the main program. However, CZSPRS does not use primes p such that for input polynomials A and B, $p \mid ldcf(A)$ or $p \mid ldcf(B)$, since

the p.r.s.'s computed over GF(p) for such primes p are not applicable. Also, after applying CZCPRS, CZSPRS checks whether or not the degree sequence of the returned p.r.s. is maximal with respect to the prior p.r.s.'s computed, rejecting those p.r.s.'s with non-maximal degree sequences. (The meaning of non-maximal here is relative to the lexicographical ordering of all finite sequences of non-negative integers; see Heindel [HEL70], Section 2.9, for a discussion of this concept.) A non-maximal degree sequence is produced when the prime p over which the p.r.s. was generated divides the leading coefficient of one of the polynomials in the pseudo-subresultant p.r.s.

The theorem following the algorithm bounds the number of primes processed, and hence is used in the proof of computing time.

ALGORITHM CZSPRS:

CZSPRS(A,B,ℓ,I,N,P, $\mathcal{C}$, $\mathcal{D}$)

Complex Zero System, Pseudo-Subresultant P.R.S.

A, B, ℓ, and I are inputs. N, P, $\mathcal{C}$, and $\mathcal{D}$ are outputs. A and B are non-zero univariate integral polynomials, deg(A) $\geq$ deg(B). ℓ and I are Fortran integers. ℓ is a positive integer such that if p_1, p_2, . . . , p_ℓ are any primes in the list PRIME, and S is any subresultant of A and B, then $|S|_\infty <$

$(1/2)\prod_{i=1}^{\ell} p_i$. I = 0 or I = 1. Let $S_1, S_2, \ldots,$ $S_r, S_{r+1} = 0$ be the complete pseudo-subresultant p.r.s. of A and B. Let $n_i = \deg(S_i)$. Then N = $(n_1, n_2, \ldots, n_r)$. P is a list $(p_1, p_2, \ldots, p_\ell)$ where $p_1 < p_2 < \ldots < p_\ell$ and each p_k is contained in the list PRIME. $\mathcal{C}$ is a list $(C_1, C_2, \ldots, C_\ell)$ with $C_k = (c_{1,k}, c_{2,k}, \ldots, c_{r,k})$ where $c_{j,k} = \phi_{p_k}(\mathrm{ldcf}(S_j))$. If I = 0 then $\mathcal{D} = 0$. If I = 1, then $\mathcal{D} = (D_1, D_2, \ldots, D_\ell)$ with $D_k = (d_{1,k}, d_{2,k}, \ldots, d_{r,k})$ where $d_{j,k} = \phi_{p_k}(\mathrm{trcf}(S_j))$.

Method:

Let $p_1, p_2, \ldots, p_\ell$ be the first ℓ primes p in the list PRIME for which $\phi_p(\mathrm{ldcf}(A)) \neq 0$ and $\phi_p(\mathrm{ldcf}(B)) \neq 0$, and the degree sequence of $(\phi_p(A), \phi_p(B))$, produced by CZCPRS, is maximal relative to all other primes among the first j primes on list PRIME, where p_ℓ is the j-th prime on list PRIME. N is

the common degree sequence for $p_1, p_2, \ldots, p_\ell$. C_k is the output of leading coefficients from CZCPRS for p_k. If I = 1, D_k is the output of trailing coefficients from CZCPRS for p_k.

Description:

(1) [Initialize.] N ← 0; P ← 0; $\mathcal{C}$ ← 0; $\mathcal{D}$ ← 0; P' ← PRIME; k ← 0; a ← PLDCF(A); b ← PLDCF(B).

(2) [Leading coefficient test.] If P' = 0, (print "OUT OF PRIMES--CZSPRS"; stop); ADV(p,P'); if CMOD(p,a) = 0 or CMOD(p,b) = 0, go to (2).

(3) [Apply CZCPRS and compare degree sequences.] A* ← CPMOD(p,A); B* ← CPMOD(p,B); CZCPRS(p,A*,B*,I,N*,C,D); erase A*,B*; S ← LEXORD(N,N*); if S < 0, go to (5); if S > 0, go to (6).

(4) [N = N*] Erase N*; P ← PFA(p,P); $\mathcal{C}$ ← PFL(C,$\mathcal{C}$); if I ≠ 0, $\mathcal{D}$ ← PFL(D,$\mathcal{D}$); k ← k + 1; if k < ℓ, go to (2); go to (7).

(5) [N < N*] Erase N; N ← N*; erase P, $\mathcal{C}$, $\mathcal{D}$; P ← PFA(p,0); $\mathcal{C}$ ← PFL(C,0); $\mathcal{D}$ ← 0; if I ≠ 0, $\mathcal{D}$ ← PFL(D, 0); k ← 1; if k < ℓ, go to (2); go to (7).

(6) [N > N*] Erase N*; C, D; go to (2).

(7) [Finish.] P ← INV(P); $\mathcal{C}$ ← INV($\mathcal{C}$); $\mathcal{D}$ ← INV($\mathcal{D}$); erase a, b; return.

Computing Time: $\propto \{\ell + \nu^2 L(a) + \mu\nu L(b)\}\cdot\{\mu\nu + \mu L(a) + \nu L(b)\}$, where $a = |A|_1$, $b = |B|_1$, $m = \deg(A)$, $n = \deg(B)$, $\mu = m + 1$, $\nu = n + 1$.

Proof: Let N be the total number of primes processed, as given by Theorem 3.2.4 . Then the table is as follows.

Step	n_i	t_i
1	1	~ 1
2	$\leq N$	$\propto L(a) + L(b)$
3	$\leq N$	$\propto \mu\nu + \mu L(a) + \nu L(b)$
4	$\leq N$	~ 1
5	$\leq N$	~ 1
6	$\leq N$	~ 1
7	1	$\propto \ell$

The time follows immediately from the table. ∎

Theorem 3.2.4: The number N of primes processed by CZSPRS is $N \propto \ell + \nu^2 L(a) + \mu\nu L(b)$, where $a = |A|_1$, $b = |B|_1$, $m = \deg(A)$, $\mu = m + 1$, $n = \deg(B)$, $\nu = n + 1$.

Proof: A prime p is rejected in step (2) if and only if, for $\bar{a} = \mathrm{ldcf}(A)$ and $\bar{b} = \mathrm{ldcf}(B)$, $p|\bar{a}$ or $p|\bar{b}$; that is, $p|\bar{a}\bar{b}$. Since $\bar{a} \leq a$, $\bar{b} \leq b$, and $p > 2$, the number N_1 of such

primes is $N_1 \underline{\propto} L(ab) \sim L(a) + L(b)$.

A prime p is rejected in steps (5) or (6) if and only if it produces a non-maximal degree sequence. Let $S_1, S_2, \ldots, S_{r+1}$ be a pseudo-subresultant p.r.s. for A and B. Then the degree sequence for p is non-maximal if and only if, for $s_i = \mathrm{ldcf}(S_i)$, $p \mid s_i$ for some i, $3 \leq i \leq r$; that is, $p \mid \prod_{i=3}^{r} s_i$. Then $\prod_{i=3}^{r} s_i \leq (a^n b^m)^{r-2} \leq (a^n b^n)$ so $L(\prod_{i=3}^{r} s_i) \underline{\propto} \nu^2 L(a) + \mu\nu L(b)$. Since $p > 2$, the number N_2 of primes rejected in steps (5) or (6) is $N_2 \underline{\propto} L(\prod_{i=3}^{r} s_i) \underline{\propto} \nu^2 L(a) + \mu\nu L(b)$. The number of primes not rejected is given by input ℓ. Hence the total number N of primes is bounded by $N \underline{\propto} \ell + N_1 + N_2 \underline{\propto} \ell + \nu^2 L(a) + \mu\nu L(b)$. ∎

In the next theorem, a method for obtaining a positive pseudo-subresultant p.r.s. from a natural pseudo-subresultant p.r.s. is presented.

Theorem 3.2.5: If $D_1, D_2, \ldots, D_{r+1}$ is a natural pseudo-subresultant p.r.s., b_i defined as in Theorem 3.2.3, $u_i = \mathrm{sign}(b_i)$ and $C_i = u_i D_i$, then $C_1, C_2, \ldots, C_{r+1}$ is a

positive pseudo-subresultant p.r.s.

Proof: $C_i = u_i D_i = u_i b_i A_i = |b_i|\cdot A_i$. Since $A_1, A_2, \ldots, A_r$ is a natural p.r.s., $A_i = Q_i A_{i+1} + A_{i+2}$.

Multiplying both sides by $|b_i|\cdot|b_{i+1}|\cdot|b_{i+2}|$ gives

$$|b_{i+1}|\cdot|b_{i+2}|\cdot C_i = Q_i\cdot|b_i|\cdot|b_{i+2}|\cdot C_{i+1} + |b_i|\cdot|b_{i+1}|\cdot C_{i+2}.$$

Hence $e_i C_i = Q_i' C_{i+1} + f_i C_{i+2}$ where $e_i = |b_{i+1}|\cdot|b_{i+2}|$, $f_i = |b_i|\cdot|b_{i+1}|$, and $Q_i' = |b_i|\cdot|b_{i+2}|\cdot Q_i$, so $e_i f_i > 0$ and $C_1, C_2, \ldots, C_{r+1}$ is a positive p.r.s. Since $D_i = \pm \mathrm{pp}(A_{i-1}, A_i)$ and $C_i = \pm D_i$, it is also a pseudo-subresultant p.r.s. ∎

The following theorem gives a recurrence relation for computing the u_i of Theorem 3.2.5.

Theorem 3.2.6: Let A_i, b_i, and D_i be as in Theorem 3.2.3, $d_i = \mathrm{ldcf}(D_i)$, $s_i = \mathrm{sign}(d_i)$, and $u_i = \mathrm{sign}(b_i)$ as in Theorem 3.2.5. Then $u_1 = u_2 = 1$,

$$u_3 = s_2^{\delta_1+1}, \tag{2}$$

$$u_{i+2} = (u_i s_i u_{i+1} s_{i+1})^{\delta_i+1} u_{i+1}, \quad 2 \leq i \leq r-2. \quad (3)$$

Proof: $b_3 = a_2^{\delta_1+1} = d_2^{\delta_1+1}$, so $u_3 = \text{sign}(b_3) = s_2^{\delta_1+1}$.

Since $\text{sign}(a_i) = \text{sign}(d_i/b_i) = \text{sign}(d_i)/\text{sign}(b_i) = s_i/u_i = s_i u_i$, then, for $i \geq 2$, $u_{i+2}/u_{i+1} = \text{sign}(b_{i+2})/\text{sign}(b_{i+1}) = \text{sign}(b_{i+2}/b_{i+1}) = \text{sign}(a_i^{\delta_i-1} a_{i+1}^{\delta_i+1}) = \{\text{sign}(a_i)\}^{\delta_i-1} \cdot \{\text{sign}(a_i)\}^{\delta_i+1} = \{\text{sign}(a_{i+1})\}^{\delta_i+1} = \{\text{sign}(a_i)\}^{\delta_i+1} \cdot \{\text{sign}(a_{i+1})\}^{\delta_i+1} =$

$(u_i s_i u_{i+1} s_{i+1})^{\delta_i+1}$. Hence $u_{i+2} = (u_i s_i u_{i+1} s_{i+1})^{\delta_i+1} u_{i+1}$. ∎

A negative p.r.s. is required for a Sturm sequence, so the next theorem shows how to compute a negative p.r.s. from a positive p.r.s. by using appropriate sign corrections.

Theorem 3.2.7: If $A_1, A_2, \ldots, A_{r+1}$ is a positive p.r.s., $\sigma_j = 1$ for $j \equiv 1$ or $j \equiv 2 \pmod 4$ and $\sigma_j = -1$ for $j \equiv 3$ or $j \equiv 0 \pmod 4$, and $B_j = \sigma_j A_j$, then $B_1, B_2, \ldots, B_{r+1}$ is a negative p.r.s.

Proof: $e_j A_j = Q_j A_{j+1} + f_j A_{j+2}$. Multiplying by $\sigma_{j+2}\sigma_{j+1}$ gives $\sigma_{j+2} e_j B_j = (\sigma_{j+2}\sigma_{j+1}\sigma_j)Q_j B_{j+1} + \sigma_{j+2} f_j B_{j+2}$ or $\bar{e}_j B_j = \bar{Q}_j B_{j+1} + \bar{f}_j B_{j+2}$. Then $\bar{e}_j \bar{f}_j = \sigma_{j+2}\sigma_{j+1} e_j f_j < 0$ since $e_j f_j > 0$. ∎

The next two theorems combine Theorem 3.2.6 and Theorem 3.2.7 to develop a method for obtaining a negative pseudo-subresultant p.r.s. from a natural pseudo-subresultant p.r.s.

Theorem 3.2.8: Let σ_j be as in Theorem 3.2.7, $\alpha_j = -\sigma_j\sigma_{j+1}$, u_j and s_j be as in Theorem 3.2.6, $v_j = \sigma_j u_j$, and $w_j = v_j s_j$. Then

$$v_{j+2} = \alpha_{j+1}(\alpha_j w_j w_{j+1})^{\delta_j+1} v_{j+1}. \tag{4}$$

Proof: $v_{j+2} = \sigma_{j+2} u_{j+2} = \sigma_{j+2}(u_j s_j u_{j+1} s_{j+1})^{\delta_j+1} u_{j+1}$

$= \sigma_{j+2}(\sigma_j v_j s_j \sigma_{j+1} v_{j+1} s_{j+1})^{\delta_j+1} \sigma_{j+1} v_{j+1} =$

$\alpha_{j+1}(\alpha_j w_j w_{j+1})^{\delta_j+1} v_{j+1}$. ∎

Theorem 3.2.9: If $D_1, D_2, \ldots, D_{r+1}$ is a natural pseudo-subresultant p.r.s., v_j as defined in Theorem 3.2.8,

and $E_j = v_j D_j$, then $E_1, E_2, \ldots, E_{r+1}$ is a negative pseudo-subresultant p.r.s.

Proof: $E_j = v_j D_j = \sigma_j u_j D_j$. By Theorem 3.2.5 $u_1 D_1, u_2 D_2, \ldots, u_{r+1} D_{r+1}$ is a positive pseudo-subresultant p.r.s. By Theorem 3.2.7 $\sigma_1 u_1 D_1, \sigma_2 u_2 D_2, \ldots, \sigma_{r+1} u_{r+1} D_{r+1}$ is a negative pseudo-subresultant p.r.s. ∎

This completes the theoretical tools necessary for computing a negative pseudo-subresultant p.r.s. Before considering the algorithm for this operation, however, one other algorithm is presented to simplify initial construction of the natural pseudo-subresultant p.r.s. It uses mixed radix methods to compute the signs of a set of integers in residue representation (such as the set of coefficients in residue representation produced by CZSPRS). An input list allows a different number of primes and residues to be used in computing each integer. This is useful in the p.r.s. application because the coefficients of various polynomials in the p.r.s. have widely differing bounds.

ALGORITHM CZCLS:

S=CZCLS(C,P,L)

Complex Zero System, Construct

List of Signs

P is a list $(p_{i_1}, p_{i_2}, \ldots, p_{i_k})$ where p_{i_j} is the j-th element of PRIME and $i_1 < i_2 < \ldots < i_k$. C is a list $((c_{1,1}, c_{1,2}, \ldots, c_{1,k}), \ldots, (c_{r,1}, c_{r,2}, \ldots, c_{r,k}))$, where $c_{i,j} \in GF(p_{i_j})$. L is a list $(\ell_1, \ell_2, \ldots, \ell_r)$ of positive integers, with $1 \leq \ell_i \leq k$. S is the list $(s_1, s_2, \ldots, s_r)$ where s_i is the sign of the integer a_i of least absolute value such that $a_i \equiv c_{i,j} \pmod{p_{i_j}}$ for $1 \leq j \leq \ell_i$.

Method:

Let $p^* = p_{i_m}$. Construct the lists $P^* = (p^*_1, p^*_2, \ldots, p^*_{\ell_j})$ and $c^* = (c_{j,1}, c_{j,2}, \ldots, c_{j,\ell_j})$.

Then $a_j = RTMR(P^*, c^*)$ and $s_j = SMRR(a_j)$.

Description:

(1) [Initialize.] S ← 0; if C = 0, return; C' ← C; L' ← L.

(2) [Obtain next coefficient residues.] ADV(c,C'); ADV(ℓ,L'); P' ← P; c' ← c; P* ← 0; c* ← 0.

(3) [Obtain ℓ residues and primes.] ADV(d,c'); ADV(p,P'); c* ← PFA(d,c*); P* ← PFA(p,P*); ℓ ← ℓ - 1; if $\ell > 0$, go to (3).

(4) [Obtain sign of coefficient.] P* ← INV(P*); c* ← INV(c*); a ← RTMR(P*,c*); s ← SMRR(a); erase P*, c*, a; S ← PFA(s,S); if C' $\neq 0$, go to (2).

(5) [Finish.] S ← INV(S); return.

Computing Time: $\underline{\propto} rki_k + 1$.

Proof:

Step	n_i	t_i
1	1	~ 1
2	r	~ 1
3	rk	~ 1
4	r	$\underline{\propto} ki_k$
5	1	$\sim r$

The final algorithm in this section, then, computes a negative p.r.s. using modular methods. It computes an overall bound on the coefficients and uses CZSPRS with this bound. Individual bounds on each polynomial of the p.r.s. are computed, and these bounds and the results of CZSPRS are input to CZCLS to construct the signs of the leading and trailing coefficients of the natural pseudo-subresultant p.r.s. Theorem 3.2.5 through Theorem 3.2.9 are then applied to generate sign correction factors, which are used

to alter the natural pseudo-subresultant p.r.s. to a negative pseudo-subresultant p.r.s.

ALGORITHM CZNMRS:

CZNMRS(A,B,I,N,S,T)

Complex Zero System, Negative Modular

Polynomial Remainder Sequence

A, B, and I are inputs; N, S, and T are outputs. A and B are non-zero univariate integral polynomials with deg(A) $\geq$ deg(B); I = 0 or I = 1, a Fortran integer. Let $A_1, A_2, \ldots, A_r, A_{r+1} = 0$ be any negative p.r.s. for A and B. Let $n_i = \deg(A_i)$, $s_i = \text{sign}(\text{ldcf}(A_i))$, and $t_i = \text{sign}(\text{trcf}(A_i))$. Then $N = (n_1, n_2, \ldots, n_r)$, $S = (s_1, s_2, \ldots, s_r)$, and $T = (t_1, t_2, \ldots, t_r)$ if I = 1 and T = 0 if I = 0.

Method:

Let $a = |A|_1$, $b = |B|_1$, $m = \deg(A)$, $n = \deg(B)$, $h = \lceil \log_2(a) \rceil$, $k = \lceil \log_2(b) \rceil$, $\ell = hn + km + 1$, and $\ell' = \lceil \ell/\text{PEXP} \rceil$. CZSPRS is applied to A, B, and ℓ', obtaining N, P, $\mathcal{C}$, $\mathcal{D}$. Let $N = (n_1, n_2, \ldots, n_r)$, so that $m = n_1$, $n = n_2$. Let $S_1, S_2, \ldots, S_r, S_{r+1} = 0$ be the natural pseudo-subresultant p.r.s. of A and B.

For $3 \leq i \leq r$, let $\ell_i = h(n - n_{i-1} + 1) + k(m - n_{i-1} + 1) + 1$ and $\ell'_i = \lceil \ell_i / \overline{PEXP} \rceil$. Then $|S_i|_\infty \leq (1/2)(2^{\ell_i})$, so ldcf($S_i$) is determined by the i-th terms of the first ℓ'_i elements of $\mathcal{C}$, and similarly for trcf(S_i) and $\mathcal{D}$ if I = 1. Applying CZCLS yields $S = (s_1, s_2, \ldots, s_r)$ and, if I = 1, $T = (t_1, t_2, \ldots, t_r)$. (Note that for i=1,2, s_i and t_i are obtained directly from $A = S_1$ and $B = S_2$.) Theorems 3.2.5 to 3.2.9 are then used to obtain sign correction terms for S, and for T if I = 1.

Description:

(1) [Initialize.] a ← PNORMF(A); b ← PNORMF(B); m ← PDEG(A); n ← PDEG(B); ELPOF2(a,t,h); ELPOF2(b,t,k); $\ell \leftarrow hn + km + 1$; $\overline{P} \leftarrow PEXP - 1$; $\ell' \leftarrow (\ell + \overline{P})/PEXP$; erase a,b; CZSPRS(A,B,$\ell'$,I,N,P,$\mathcal{C}$,$\mathcal{D}$); $\overline{r} \leftarrow$ LENGTH(N) - 2; T ← 0; L ← 0; if $\overline{r} = 0$, go to (3); N' ← TAIL(N).

(2) [Compute number of primes for each coefficient.] For i ← 1,2,...,$\overline{r}$, do: (ADV(n_1,N'); $\ell_2 \leftarrow h(n - n_1 + 1) + k(m - n_1 + 1) + 1$; $\ell'_2 \leftarrow (\ell_2 + \overline{P})/PEXP$; L ← PFA($\ell'_2$,L)); L ← INV(L).

(3) [Compute list of TRCF signs.] If I = 0, go to (4); $t_1 \leftarrow$ PTCS(A); $t_2 \leftarrow$ PTCS(B); $\mathcal{Q}_1 \leftarrow$ CZTRAN($\mathcal{Q}$); erase $\mathcal{Q}$; $\mathcal{Q}_2 \leftarrow$ TAIL(TAIL($\mathcal{Q}_1$)); T' $\leftarrow$ CZCLS($\mathcal{Q}_2$,P,L); T $\leftarrow$ PFA(t_1, PFA(t_2,T')); erase $\mathcal{Q}_1$.

(4) [Compute list of LDCF signs.] $s_1 \leftarrow$ PSIGN(A); $s_2 \leftarrow$ PSIGN(B); $\mathcal{C}_1 \leftarrow$ CZTRAN($\mathcal{C}$); erase $\mathcal{C}$; $\mathcal{C}_2 \leftarrow$ TAIL(TAIL($\mathcal{C}_1$)); S' $\leftarrow$ CZCLS($\mathcal{C}_2$,P,L); S $\leftarrow$ PFA(s_1, PFA(s_2,S')); erase $\mathcal{C}_1$, L, P; if $\overline{r} = 0$, return; $n_2 \leftarrow$ FIRST(N); N' $\leftarrow$ TAIL(N); Q $\leftarrow$ 0.

(5) [Compute list of deltas.] For i $\leftarrow$ 1,2,...,$\overline{r}$, do: ($n_1 \leftarrow n_2$; ADV(n_2,N'); $\delta \leftarrow n_1 - n_2$; Q $\leftarrow$ PFA(δ + 1,Q)); Q $\leftarrow$ INV(Q).

(6) [Initialize for sign correction.] S' $\leftarrow$ S; ADV2(s_1, s_2,S'); if I = 1, T' $\leftarrow$ TAIL(TAIL(T)); $w_1 \leftarrow s_1$; $w_2 \leftarrow s_2$; $\alpha_1 \leftarrow 1$; $\alpha_2 \leftarrow -1$.

(7) [Compute first correction.] DECAP(q_1,Q); $v_3 \leftarrow -s_2$; if MOD(q_1,2) = 0, $v_3 \leftarrow -1$; go to (9).

(8) [Compute next correction.] DECAP(q_1,Q); $v_3 \leftarrow \alpha_1 w_1 w_2$;

if MOD(q_1,2) = 0, $v_3 \leftarrow 1$; $v_3 \leftarrow \alpha_2 v_3 v_2$.

(9) [Correct sign if necessary.] $s_3 \leftarrow$ FIRST(S'); $w_3 \leftarrow s_3 v_3$; if $v_3 < 0$, ALTER(w_3,S'); if I = 0, go to (10);

if $v_3 < 0$, ALTER(-FIRST(T'),T'); T' $\leftarrow$ TAIL(T').

(10) [Increment and check for end.] S' $\leftarrow$ TAIL(S');

$w_1 \leftarrow w_2$; $w_2 \leftarrow w_3$; $v_2 \leftarrow v_3$; $\alpha_1 \leftarrow \alpha_2$; $\alpha_2 \leftarrow -\alpha_2$;

if S' $\neq$ 0, go to (8); return.

Computing Time: $\preceq \mu^2 \nu^2 L(d)^2$, where $a = |A|_1$, $b = |B|_1$, $d = \max\{a,b\}$, $m = \deg(A)$, $n = \deg(b)$, $\mu = m + 1$, and $\nu = n + 1$.

Proof:

Step	n_i	t_i
1	1	$\preceq \{\mu\nu + \mu L(a) + \nu L(b)\} \cdot \{\nu^2 L(a) + \mu\nu L(b)\}$
2	≤ 1	$\preceq \nu$
3	1	$\preceq \nu^2 \{\nu L(a) + \mu L(b)\}^2$
4	1	$\preceq \nu^2 \{\nu L(a) + \mu L(b)\}^2$
5	1	$\preceq \nu$
6	1	~ 1
7	1	~ 1
8	$\leq \nu$	~ 1

9	$\leq \nu$	~ 1
10	$\leq \nu$	~ 1

t_1 follows from noting that $\ell \propto \nu L(a) + \mu L(b)$.

The maximum length r of the p.r.s. is $r \propto \nu$. The maximum number of primes used for determining a coefficient's sign, ℓ, is $\ell \propto \nu L(a) + \mu L(b)$. The maximum index of a prime, i, is given by the total number of primes used in CZSPRS, $i \propto \nu^2 L(a) + \mu\nu L(b)$.

Hence, $t_{CZCLS} \propto \nu\{\nu L(a) + \mu L(b)\}\cdot\{\nu^2 L(a) + \mu\nu L(b)\} =$

$\nu^2\{\nu L(a) + \mu L(b)\}^2$.

Summing the table entries gives $\sum_{i=1}^{10} n_i t_i \propto$

$\nu\{\nu L(a) + \mu L(b)\}\cdot\{\nu^2 L(a) + \mu\nu L(b) + \mu\nu + \mu L(a) + \nu L(b)\}$

$\propto \nu\{\nu L(a) + \mu L(b)\}\cdot\{\mu L(a) + \nu^2 L(a) + \mu\nu L(b)\} \propto$

$\mu^2\nu^2 L(d)^2$. ∎

3.3 Application of Sturm sequences

This section applies the algorithms of Section 3.2 to determine the number of zeros of a polynomial in basic regions of the plane, using techniques discussed in Section 3.1 . The regions referred to as "basic" are the upper and lower half-planes and the real axis.

The first algorithm generates a Sturm sequence (using either modular or integer methods), then computes the number of sign variations, V(x), of this sequence either at $x = \pm\infty$ or at $x = \pm\infty$ and $x = 0$. These values of V(x) are used by subsequent algorithms in this section in determining root location.

ALGORITHM CZNVSS:

CZNVSS(A,B,I,h,k,ℓ)

Complex Zero System, Number of

Variations in a Sturm Sequence

A, B, and I are inputs; h, k, and ℓ are outputs. A and B are non-zero univariate integral polynomials, with $\deg(A) \geq \deg(B)$. I = 0 or I = 1, a Fortran integer. h, k, and ℓ are Fortran integers. Let S be any Sturm sequence for A and B (by definition, a negative p.r.s. $A_1, A_2, \ldots, A_r$ with $A_1 \simeq A$, $A_2 \simeq B$, $\text{sign}(A_1) = \text{sign}(A)$, $\text{sign}(A_2) = \text{sign}(B)$, and $\text{psrem}(A_{r-1}, A_r) = 0$).

h, k, and ℓ are the number of variations in S at $-\infty$,

0, and $+\infty$, respectively, except that if I = 0, then k = 0.

Description:

(1) [Obtain degrees and coefficient signs.]

If METHOD = 0, (CZNPRS(A,B,I,N,S,T); go to (2)); CZNMRS(A,B,I,N,S,T).

(2) [Initialize.] $h \leftarrow 0$; $k \leftarrow 0$; $\ell \leftarrow 0$; $s_1 \leftarrow 0$; $t_1 \leftarrow 0$; $u_1 \leftarrow 0$.

(3) [Process next term.] DECAP(n_2,N); DECAP(s_2,S); $t_2 \leftarrow 0$; if $I \neq 0$, DECAP(t_2,T); $u_2 \leftarrow s_2$; if MOD(n_2,2) $\neq 0$, $u_2 \leftarrow -u_2$; if $s_1 + s_2 = 0$, $\ell \leftarrow \ell + 1$; $s_1 \leftarrow s_2$; if $t_2 \neq 0$, (if $t_1 + t_2 = 0$, $k \leftarrow k + 1$; $t_1 \leftarrow t_2$); if $u_1 + u_2 = 0$, $h \leftarrow h + 1$; $u_1 \leftarrow u_2$; if N $\neq 0$, go to (3); return.

Computing Time: $\propto \mu^2 \nu^2 L(\mu d)^2$ if modular methods are used; $\propto \mu^3 \nu^{1+k} L(\mu d)^2$ if integer methods are used; m = deg(A), n = deg(B), $\mu = m + 1$, $\nu = n + 1$, and $|A|_\infty$, $|B|_\infty \leq d$; k = 0 if the p.r.s. is normal, k = 1 otherwise.

Proof: $n_2 = 1$ and $t_2 \sim 1$. $n_3 \leq$ length of p.r.s. $\propto$

ν and $t_3 \sim 1$. Hence the time for the algorithm is dominated by the time to compute the p.r.s. ∎

As shown previously, the number of real roots of a polynomial A is $V(-\infty) - V(+\infty)$, using a Sturm sequence for A and A'. The following algorithm computes the number of real roots of a polynomial using this method. In addition, it computes the number of roots in the upper half-plane by noting that any roots of an integral polynomial which are not real must occur as complex conjugate pairs, one in the upper and one in the lower half-plane.

ALGORITHM CZNRNA:

CZNRNA(A,h,k)

Complex Zero System, Number of Real Roots and Number Above Real Axis

A is the input; h and k are outputs. A is a non-zero, square-free univariate integral polynomial. h is the number of real roots of A and k is the number of roots of A above the real axis.

Description:

(1) [Initialize.] n ← PDEG(A); if n = 0, (h ← 0; k ← 0; return).

(2) [Compute derivative.] A' ← PDERIV(A,FIRST(A)).

(3) [Compute location of roots.] CZNVSS(A,A',0,ℓ_1, ℓ_2, ℓ_3); erase A'; h ← $\ell_3 - \ell_1$; k ← (n - h)/2; return.

Computing Time: $\propto \mu^{4+k} L(\mu d)^2$, where $m = \deg(A)$, $\mu = m + 1$, and $|A|_\infty = d$; $k = 0$ if modular methods are used or the p.r.s. is normal, $k = 1$ otherwise.

Proof:

Step	n_i	t_i
1	1	~ 1
2	≤ 1	$\propto \mu L(d)$
3	≤ 1	$\propto \mu^{4+k} L(\mu d)^2$

$L(\mu|A'|_\infty) \leq L(\mu^2 d) \sim L(\mu d)$. ∎

In later instances it will be necessary to know, in addition to the number of real roots, how many of these real roots are in $(-\infty,0]$ and how many in $(0,+\infty)$. The following algorithm computes these values, using $V(-\infty) - V(0)$ and $V(0) - V(+\infty)$, respectively. It also returns the number of roots in the upper half-plane, employing the same techniques for this as CZNRNA.

ALGORITHM CZNNPA:

CZNNPA(A,h,k,ℓ)

Complex Zero System, Number of Non-positive Real Roots, Positive Real Roots, and Roots Above the Real Axis

A is the input; h, k, and ℓ are outputs. A is a non-zero square-free univariate integral polynomial. h is the

number of non-positive real roots of A, k is the number of positive real roots of A, and ℓ is the number of roots of A above the real axis.

Description:

(1) [Initialize.] n ← PDEG(A); if n = 0, (h ← 0; k ← 0; ℓ ← 0; return).

(2) [Compute values.] A' ← PDERIV(A,FIRST(A)); CZNVSS(A, A',1,h_1,h_2,h_3); erase A'; $h_1 \leftarrow h_1 - h_2$; $k \leftarrow h_2 - h_3$; $\ell \leftarrow (n - h - k)/2$; return.

Computing Time: $\propto \mu^{4+k} L(\mu d)^2$, where m = deg(A), $\mu = m + 1$, and $d = |A|_\infty$; k = 0 if modular methods are used or the p.r.s. is normal, k = 1 otherwise.

Proof:

Step	n_i	t_i
1	1	~ 1
2	≤ 1	$\propto \mu^{4+k} L(\mu d)^2$

$L(\mu|A'|_\infty) \leq L(\mu^2 d) \sim L(\mu d)$. ∎

Before going to algorithms for Gaussian polynomials, one additional aspect not discussed previously must be considered. The algorithms which generate a Sturm sequence for A_1 and A_2 assume $\deg(A_1) \geq \deg(A_2)$. Hence if the

Gaussian polynomial is $A(z) = A_1(z) + iA_2(z)$ and $\deg(A_1) < \deg(A_2)$, then those algorithms could not be used to generate the required Sturm sequence $A_1, A_2, \ldots, A_r$. The following theorem, however, shows how the desired values can be computed from the sequence $B_1 = A_2, B_2 = A_1, \ldots,$ B_r so that an additional special algorithm need not be implemented.

Theorem 3.3.1: Let $A = A_1 + iA_2$, with $A_1, A_2 \neq 0$, be a univariate Gaussian polynomial with no real zeros. Let S_2 be any Sturm sequence for A_2 and A_1, and let $V_2(x)$ be the number of variations in sign of S_2 at x. Then the number of zeros of A in the upper half-plane is $\{\deg(A) + V_2(-\infty) - V_2(+\infty)\}/2$.

Proof: Let $g(A_1, A_2)$ be the number of real zeros, x_0, of A_1 such that $A_1'(x_0)\cdot A_2(x_0) < 0$ minus the number such that $A_1'(x_0)\cdot A_2(x_0) > 0$. (Note that $g(A_1, A_2) = (b - a)$, where $(b - a)$ is the quantity described in Theorem 3.2.2 .) Let S_1 be any Sturm sequence for A_1 and A_2 and let $V_1(x)$ be the number of variations in sign of S_1 at x.

Section 3.1 shows that the number of zeros of A in the upper half-plane is $\{\deg(A) + g(A_1, A_2)\}/2 = \{\deg(A) + V_1(+\infty) - V_1(-\infty)\}/2$.

Since A and iA have the same zeros, the number of zeros of A in the upper half-plane is the same as the number of zeros of iA in the upper half-plane. $iA = -A_2 + iA_1$. It is immediate from the definition that $g(-A_2, A_1) = -g(A_2, A_1)$.

Hence, the number of zeros of iA, and also of A, in the upper half-plane is $\{\deg(A) + g(-A_2, A_1)\}/2 = \{\deg(A) - g(A_2, A_1)\}/2 = \{\deg(A) + V_2(-\infty) - V_2(+\infty)\}/2$. ∎

The following algorithm computes the number of zeros of a Gaussian polynomial in the upper and lower half-planes, assuming that the polynomial has no real zeros. The results of Theorem 3.3.1 are used when necessary.

ALGORITHM CZABRA:

CZABRA(A,h,k)

Complex Zeros Above and

Below the Real Axis

A is the input; h and k are outputs. A is a non-zero univariate Gaussian polynomial having no real zeros. h and k are Fortran integers, the number of zeros of A above and below the real axis, respectively.

Description:

(1) [Initialize.] FIRST2(A_1,A_2,A); $n_1 \leftarrow$ PDEG(A_1); $n_2 \leftarrow$ PDEG(A_2); $n \leftarrow$ MAX0(n_1,n_2); if $A_1 = 0$ or $A_2 = 0$, ($h \leftarrow n/2$; $k \leftarrow h$; return); if $n_1 < n_2$, ($T \leftarrow A_1$; $A_1 \leftarrow A_2$; $A_2 \leftarrow T$).

(2) [Compute number of sign changes.] CZNVSS(A_1,A_2,0, ℓ_1,ℓ_2,ℓ_3); $\ell \leftarrow \ell_3 - \ell_1$; if $n_1 < n_2$, $\ell \leftarrow -\ell$; $h \leftarrow (n + \ell)/2$; $k \leftarrow n - h$; return.

Computing Time: $\propto \mu^{4+k} L(\mu d)^2$, where $m = \deg(A)$, $\mu = m + 1$, and $d = |A|_\infty$; $k = 0$ if modular methods are used or the p.r.s. is normal, $k = 1$ otherwise.

Proof:

Step	n_i	t_i
1	1	~ 1
2	1	$\propto \mu^{4+k} L(\mu d)^2$

$|A_1|_\infty, |A_2|_\infty \leq d$; $\deg(A_1), \deg(A_2) \leq m$. The time for CZNVSS follows immediately from this. ∎

In order to handle a Gaussian polynomial with real zeros, one need only note the following. Let $A = A_1 + iA_2$,

where A_1 and A_2 are integral polynomials. Then at any real point x, $A_1(x)$ and $A_2(x)$ are real values. Hence at x_0 a real zero of A, $A_1(x_0) = a + i0$ is pure real and $iA_2(x_0) = 0 + ib$ is pure imaginary. Thus $A_1(x_0) + iA_2(x_0) = a + ib$ can be zero only if $a = b = A_1(x_0) = A_2(x_0) = 0$. This means that x_0 is a zero of the gcd of A_1 and A_2, and hence all real zeros of A are real zeros of $\gcd(A_1, A_2)$.

The following algorithm splits a Gaussian polynomial $A = A_1 + iA_2$ into $A = B(\overline{A}_1 + i\overline{A}_2)$, where $B = \gcd(A_1, A_2)$, $\overline{A}_1 = A_1/B$, and $\overline{A}_2 = A_2/B$. Then B is an integral polynomial so CZNRNA is applied to it. $\overline{A}_1 + i\overline{A}_2$ is a Gaussian polynomial with no real zeros, so CZABRA can be applied to it. These results are then combined to give the number of zeros of a Gaussian polynomial in the upper and lower half-planes and on the real axis.

ALGORITHM CZNZHP:

CZNZHP(A,h,k)

Complex Zero System, Number of

Zeros in Upper Half-Plane

A is the input; h and k are outputs. A is a non-zero square-free univariate Gaussian polynomial. h is the

number of real zeros of A, k is the number of zeros of A in the upper half-plane.

Description:

(1) [Initialize.] FIRST2(A_1,A_2,A); if $A_1 = 0$, (CZNRNA(A_2,h,k); return); if $A_2 = 0$, (CZNRNA(A_1,h,k); return).

(2) [Compute number for real and complex factors.] PGCDCF(A_1,A_2,B,$\overline{A}_1$,$\overline{A}_2$); $\overline{A} \leftarrow$ PFL($\overline{A}_1$,PFL($\overline{A}_2$,0)); CZNRNA(B,h,k_1); erase B; CZABRA($\overline{A}$,k_2,ℓ); erase $\overline{A}$; $k \leftarrow k_1 + k_2$; return.

Computing Time: $\propto \mu^{4+k}(\mu L(\mu) + L(d))^2$, where $m = \deg(A)$, $\mu = m + 1$, and $d = |A|_\infty$; $k = 0$ if modular methods are used or the p.r.s.'s are normal, $k = 1$ otherwise.

Proof: $t_1 \propto \mu^5 L(\mu d)^2$. $|\overline{A}_1|_\infty, |\overline{A}_2|_\infty, |B|_\infty \leq \mu^{2m}(\mu d)^{2m+1} = \mu^{2m+1} d$ since $\deg(A_1), \deg(A_2) \leq m$; $|A_1|_\infty, |A_2|_\infty \leq d$; and hence $|A_1|_1, |A_2|_1 \leq \mu d$. Thus $t_{CZNRNA}(B) \propto \mu^{4+k}(2\mu^2 L(\mu^2 d)) \propto \mu^{4+k}(\mu L(\mu) + L(d))^2$. Similar relations hold for $t_{CZABRA}(\overline{A})$. $t_{PGCDCF}(A_1,A_2) \propto \mu^3 L(\mu d)^2$. ∎

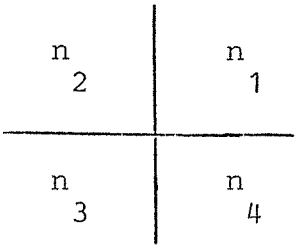
CHAPTER 4: ROOT SQUARING AND MAPPINGS

4.1 Theoretical background

This section presents a theoretical background of root squaring and mappings, which will be used to expand the algorithms in Section 3.3 in order to determine the number of roots of a polynomial in a variety of other areas in the complex plane.

Root squaring, as the name implies, is simply a process which derives from a polynomial A with roots $\alpha_1, \alpha_2, \ldots, \alpha_n$ a polynomial A* with roots $\alpha_1^2, \alpha_2^2, \ldots, \alpha_n^2$. Although it is a fairly common concept, its application here will be somewhat different.

Suppose a polynomial A has n_i roots in quadrant Q_i, with no roots on the axes:



Root squaring is used to establish a relationship between n_1 and n_3 and between n_2 and n_4 in terms of the number of zeros of another polynomial in the upper and lower

half-planes.

Consider root α_i of A in quadrant q_i:

$$0 < \arg(\alpha_1) < \pi/2; \tag{1}$$

$$\pi/2 < \arg(\alpha_2) < \pi; \tag{2}$$

$$\pi < \arg(\alpha_3) < 3\pi/2; \tag{3}$$

$$3\pi/2 < \arg(\alpha_4) < 2\pi. \tag{4}$$

If root squaring is applied to A, giving a polynomial A*, then the roots α_i^2 of A* corresponding to α_i of A have the following arguments:

$$0 < \arg(\alpha_1^2) < \pi; \tag{5}$$

$$\pi < \arg(\alpha_2^2) < 2\pi; \tag{6}$$

$$2\pi < \arg(\alpha_3^2) < 3\pi; \tag{7}$$

$$3\pi < \arg(\alpha_4^2) < 4\pi. \tag{8}$$

Hence, roots of A in quadrants one and three correspond to roots of A* in the upper half-plane, and roots of A in quadrants two and four correspond to roots of A* in the lower half-plane. Using the methods discussed in Section 3.3, one can determine the number of roots of A* in the

upper and lower half-planes, say u and l. Then $n_1 + n_3 =$ u and $n_2 + n_4 = l$.

The following theorem gives a method for performing root squaring.

Theorem 4.1.1: Let A be a univariate polynomial over C, the field of complex numbers, of degree $n > 0$. Let $A(x) = \sum_{i=0}^{n} a_i x^i = a_n \prod_{i=1}^{n} (x - \alpha_i)$ with $\alpha_i \in C$ for $1 \leq i \leq n$.

Let $A_1(x) = \sum_{\substack{i=0 \\ i \text{ even}}}^{n} a_i x^i$, $A_2(x) = \sum_{\substack{i=0 \\ i \text{ odd}}}^{n} a_i x^i$, $B = A_1^2 - A_2^2$, and $A^*(x^2) = B(x)$. Then $A^*(x) = (-1)^n a_n^2 \prod_{i=1}^{n} (x - \alpha_i^2)$.

Proof: $A_1(-x) = A_1(x)$ and $A_2(-x) = -A_2(x)$, so $B(x) = A_1^2(x) - A_2^2(x) = \{A_1(x) + A_2(x)\} \cdot \{A_1(x) - A_2(x)\} =$

$A(x)\{A_1(-x) + A_2(-x)\} = A(x)A(-x) = \{a_n \prod_{i=1}^{n} (x - \alpha_i)\} \cdot \{a_n \prod_{i=1}^{n} (-x - \alpha_i)\} = (-1)^n a_n^2 \prod_{i=1}^{n} (x^2 - \alpha_i^2)$ and $A^*(x) = (-1)^n a_n^2 \prod_{i=1}^{n} (x - \alpha_i^2)$. ∎

Mappings are used with the same basic idea as the preceding application of root squaring, although a different

approach is employed. Given a polynomial A and a method for finding the number of zeros of polynomials in region U, one desires a method for finding the zeros of A in some other region W.

Consider a polynomial A and a mapping Φ: U $\xrightarrow{1-1}$ W. Let $C(z) = A(\Phi(z))$. $C(z) = 0$ just in case $A(\Phi(z)) = A(w) = 0$ for $\Phi(z) = w \in W$. Hence the number of zeros of C in U is the number of zeros of A in W.

Furthermore, suppose $D(u) \neq 0$ for $u \in U$. Then $B(u) = D(u)C(u) = 0$ for $u \in U$ just in case $C(u) = 0$. Hence B(z) has the same number of zeros in U as does C(z). Thus the number of zeros of B in U is the number of zeros of A in W. Therefore, given a method for determining the number of zeros of B in U one can determine the number of zeros of A in W.

As an example, CZNNPA computes the number of zeros of an integral polynomial A in $(-\infty,0]$. Suppose one is interested in the number of zeros of A in $(-\infty,r]$ instead, where r is a non-zero rational number r_1/r_2. Let U = $(-\infty,0]$ and W = $(-\infty,r]$. Then $\Phi(z) = z + r$ is a mapping Φ: U $\xrightarrow{1-1}$ W, so let $C(z) = A(z + r)$. CZNNPA cannot be applied to C(z), however, since it is a rational polynomial.

Hence, suppose $A(z) = \sum_{i=0}^{n} a_i z^i$. Then C(z) =

$\sum_{i=0}^{n} a_i (z + r)^i = \sum_{i=0}^{n} a_i \{(r_2 z + r_1)/r_2\}^i$, so let $D(z) = r_2^n$. Then $D(z) \neq 0$ for z in $(-\infty,0]$. Hence $B(z) = D(z)C(z)$

$= \sum_{i=0}^{n} a_i r_2^{n-i} (r_2 z + r_1)^i$ has the same number of roots in $(-\infty,0]$ as $A(z)$ has in $(-\infty,r]$. Since $B(z)$ is an integral polynomial, CZNNPA can be applied.

4.2 Root squaring

This section presents algorithms for implementing Theorem 4.1.1 . To facilitate exposition, the following terminology is introduced.

A polynomial $A(x) = \sum_{i=1}^{n} a_i x^{e_i}$ will be called <u>even</u> if e_i is even for $1 \leq i \leq n$ and <u>odd</u> if e_i is odd for $1 \leq i \leq n$.

The <u>even part</u> and the <u>odd part</u> of a polynomial A will be polynomials A_e and A_o, respectively, such that $A = A_e + A_o$; A_e is an even polynomial, and A_o is an odd polynomial. The <u>halve</u> of an even polynomial $A(x) = \sum_{i=1}^{n} a_i x^{e_i}$ will be the polynomial $A(x) = \sum_{i=1}^{n} a_i x^{(e_i/2)}$.

The method presented in the theorem can then be stated as follows: split the input polynomial A into even and odd parts, A_e and A_o; square these, giving $B_e = A_e^2$ and $C_e = A_o^2$ (note that the square of an even polynomial or an odd polynomial is even); compute $D_e = B_e - C_e$ (note that the difference of two even polynomials is even); compute the

halve of even polynomial D_e, the result of this operation being the desired polynomial.

The first two algorithms, then, split input polynomials into even and odd parts.

ALGORITHM PSPLT:

PSPLT(A,A_1,A_2)

Polynomial Split into Even and Odd Exponent Terms

A is the input. A_1 and A_2 are outputs. A is a proper integral polynomial, $A(x) = \sum_{i=0}^{n} a_i x^i$, where n = deg(A). Then $A_1(x) = \sum_{\substack{i=0 \\ i \text{ even}}}^{n} a_i x^i$ and $A_2(x) = \sum_{\substack{i=0 \\ i \text{ odd}}}^{n} a_i x^i$.

Description:

(1) [Initialize.] $A_1 \leftarrow 0$; $A_2 \leftarrow 0$; if A = 0, return;

A' $\leftarrow$ TAIL(A).

(2) [Obtain term of A.] ADV2(a,j,A').

(3) [Check if even or odd.] If MOD(j,2) = 1, go to (5).

(4) [Even; prefix to A_1.] $A_1 \leftarrow$ PFA(j,PFL(BORROW(a),A_1));

go to (6).

(5) [Odd; prefix to A_2.] $A_2 \leftarrow$ PFA(j,PFL(BORROW(a),A_2)).

(6) [Test for end.] If A' $\neq$ 0, go to (2).

(7) [Finish.] If $A_1 \neq 0$, $A_1 \leftarrow PFL(PVBL(A), INV(A_1))$; if $A_2 \neq 0$, $A_2 \leftarrow PFL(PVBL(A), INV(A_2))$; return.

Computing Time: $\underline{\propto} \mu$, where $m = deg(A)$ and $\mu = m + 1$.

Proof:

Step	n_i	t_i
1	1	~ 1
2	$\underline{\leq} \mu$	~ 1
3	$\underline{\leq} \mu$	~ 1
4	$\underline{\leq} \mu$	~ 1
5	$\underline{\leq} \mu$	~ 1
6	$\underline{\leq} \mu$	~ 1
7	$\underline{\leq} 1$	$\underline{\propto} \mu$

$t_7 \underline{\propto} \mu$ since $deg(A_1) \underline{\leq} deg(A)$ and $deg(A_2) \underline{\leq} deg(A)$. ∎

ALGORITHM GPSPLT:

GPSPLT(A, A_1, A_2)

Gaussian Polynomial Split into

Even and Odd Exponent Terms

A is the input. A_1 and A_2 are outputs. A is a proper Gaussian polynomial, $A(z) = \sum_{i=0}^{n} a_i z^i$, where n =

deg(A). Then $A_1(z) = \sum_{i=0,\ i\ \text{even}}^{n} a_i z^i$ and $A_2(z) = \sum_{i=0,\ i\ \text{odd}}^{n} a_i z^i$.

Description:

(1) [Initialize.] $A_1 \leftarrow 0$; $A_2 \leftarrow 0$; if A = 0, return; FIRST2(A',A'',A).

(2) [Split parts.] PSPLT(A',A'_1,A'_2); PSPLT(A'',A''_1,A''_2).

(3) [Assemble A_1.] If $A'_1 \neq 0$ or $A''_1 \neq 0$, $A_1 \leftarrow$ PFL(A'_1,PFL(A''_1,0)).

(4) [Assemble A_2.] If $A'_2 \neq 0$ or $A''_2 \neq 0$, $A_2 \leftarrow$ PFL(A'_2,PFL(A''_2,0)); return.

Computing Time: $\underline{\propto}\ \mu$, where m = deg(A) and $\mu = m + 1$.

Proof:

Step	n_i	t_i
1	1	~ 1
2	≤ 1	$\underline{\propto}\ \mu$
3	≤ 1	~ 1
4	≤ 1	~ 1

$t_2 \underline{\propto}\ \mu$ since deg(A') $\leq$ deg(A) and deg(A'') $\leq$ deg(A). ∎

The next two algorithms compute the halves of the input polynomials.

ALGORITHM PHAEX:

B=PHAEX(A)

Polynomial with Halved Exponent

A is a proper integral polynomial, $A(x) = \sum_{i=0}^{n} a_i x^{2i}$,

where 2n = deg(A). B is the integral polynomial $B(x) = \sum_{i=0}^{n} a_i x^i$.

Description:

(1) [Initialize.] B ← 0; if A = 0, return; A' ← TAIL(A).

(2) [Obtain term of A.] ADV2(a,j,A').

(3) [Compute term of B.] B ← PFA(j/2,PFL(BORROW(a),B)); if A' ≠ 0, go to (2).

(4) [Finish.] B ← PFL(PVBL(A),INV(B)); return.

Computing Time: $\propto \mu$, where m = deg(A) and $\mu = m + 1$.

Proof:

Step	n_i	t_i
1	1	~ 1
2	$\leq \mu$	~ 1
3	$\leq \mu$	~ 1
4	≤ 1	$\propto \mu$

Steps(2) and (3) are executed once for each term of A and thus $n_2 \leq \mu$ and $n_3 \leq \mu$. Length(B) = length(A) $\leq 2\mu + 1$ and thus $t_4 \propto \mu$. ∎

ALGORITHM GPHAEX:

B=GPHAEX(A)

Gaussian Polynomial with Halved Exponent

A is a proper Gaussian polynomial, $A(z) = \sum_{i=0}^{n} a_i z^{2i}$,

where 2n = deg(A). B is the Gaussian polynomial $B(z) = \sum_{i=0}^{n} a_i z^i$.

Description:

(1) [Initialize.] B ← 0; if A = 0, return;

FIRST2(A_1, A_2, A).

(2) [Compute parts.] $B_1 \leftarrow$ PHAEX(A_1); $B_2 \leftarrow$ PHAEX(A_2).

(3) [Finish.] B ← PFL(B_1, PFL(B_2, 0)); return.

Computing Time: $\propto \mu$, where m = deg(A) and μ = m + 1.

Proof:

Step	n_i	t_i
1	1	~ 1
2	≤ 1	$\propto \mu$
3	≤ 1	~ 1

$t_2 \propto \mu$ since $deg(A_1) \leq deg(A)$ and $deg(A_2) \leq deg(A)$. ∎

A large part of the time in the root squaring algorithms will be the time to square the even and odd

polynomials. Hence, rather than use the standard product routines, the following algorithms for squaring univariate polynomials are implemented.

ALGORITHM PSQR:

B=PSQR(A)

Polynomial Squared

A is a univariate integral polynomial. B is the univariate integral polynomial $B = A^2$.

Method:

If $A = 0$, then $B = 0$. Otherwise, let $A(x) = \sum_{i=0}^{n} a_i x^i$, where $n = \deg(A)$. Let $B_{-1}(x) = 0$ and $B_j(x) = B_{j-1}(x) + a_j^2 x^{2j} + 2a_j x^j \sum_{k=j+1}^{n} a_k x^k$ for $0 \leq j \leq n$. Then $B = B_n$.

Description:

(1) [Initialize.] B ← 0; if A = 0, return; A' ← CINV(TAIL(A)); c ← PFA(2,0).

(2) [Obtain first term of A'.] DECAP2(j,a,A').

(3) [Compute first term of B'.] B' ← PFL(IPROD(a,a), PFA(2j,0)); if A' = 0, (erase a; go to (7)).

(4) [Initialize inner loop.] $\bar{a}$ ← IPROD(a,c); erase a; A'' ← A'.

(5) [Obtain next term of A'.] ADV2(k,a,A'').

(6) [Compute next term of B'.] B' ← PFL(IPROD(a,$\bar{a}$), PFA(j+k,B')); if A'' ≠ 0, go to (5); erase $\bar{a}$.

(7) [Add.] B' ← PFL(PVBL(A),B'); B'' ← PSUM(B,B'); erase B, B'; B ← B''; if A' ≠ 0, go to (2).

(8) [Finish.] Erase c; return.

Computing Time: $\underline{\propto} \mu^2 L(a)^2 + \mu^2 L(\mu)$, where m = deg(A), μ = m + 1, and a = $|A|_\infty$.

Proof:

Step	n_i	t_i
1	1	$\underline{\propto} \mu$
2	$\underline{\leq} \mu$	~ 1
3	$\underline{\leq} \mu$	$\underline{\propto} L(a)^2$
4	$\underline{\leq} \mu$	$\underline{\propto} L(a)$
5	$\underline{\leq} \mu^2$	~ 1
6	$\underline{\leq} \mu^2$	$\underline{\propto} L(a)^2$
7	$\underline{\leq} \mu$	$\underline{\propto} \mu L(\mu a)$
8	$\underline{\leq} 1$	~ 1

$|B'|_\infty \underline{\leq} 2a^2$ and thus, by induction, $|B|_\infty \underline{\leq} 2(j-1)a^2$ at the beginning of the j-th execution of step(7).

Hence $L(|B'|_\infty) \underline{\leq} L(2a^2) \sim L(a)$ and $L(|B|_\infty) \underline{\leq} L(2\mu a^2) \sim L(\mu a)$ for all executions of step (7). Thus $t_7 \underline{\propto} \mu L(\mu a)$.

Summing over all table entries $\sum_{i=1}^{8} n_i t_i \propto \mu^2 L(a)^2 +$

$\mu^2 L(\mu a) \sim \mu^2 L(a)^2 + \mu^2 L(\mu)$. ∎

ALGORITHM GPSQR:

B=GPSQR(A)

Gaussian Polynomial Squared

A is a univariate Gaussian polynomial. B is the univariate Gaussian polynomial $B = A^2$.

Method:

If A = 0 then B = 0. Otherwise, let $A = A_1 + iA_2$.

Let $B_1 = (A_1^2 - A_2^2)$ and $B_2 = 2A_1 A_2$. Then $B = B_1 + iB_2$.

Description:

(1) [Initialize.] B ← 0; if A = 0, return; FIRST2(A_1, A_2,A); c ← PFA(2,0).

(2) [Compute real part.] B' ← PSQR(A_1); B'' ← PSQR(A_2); B_1 ← PDIF(B',B''); erase B', B''.

(3) [Compute complex part.] B' ← PPROD(A_1,A_2); B_2 ← PIP(B',c); erase B', c.

(4) [Finish.] B ← PFL(B_1,PFL(B_2,0)); return.

Computing Time: $\propto \mu^2 L(a)^2 + \mu^2 L(\mu)$, where $m = \deg(A)$, $\mu = m + 1$, and $a = |A|_\infty$.

Proof:

Step	n_i	t_i
1	1	~ 1
2	≤ 1	$\propto \mu^2 L(a)^2 + \mu^2 L(\mu) + \mu L(\mu a^2)$
3	≤ 1	$\propto \mu^2 L(a)^2 + \mu^2 L(\mu) + \mu L(\mu a^2)$
4	≤ 1	~ 1

The table entries follow since $|A_1|_\infty$, $|A_2|_\infty \leq |A|_\infty$; $\deg(A_1)$, $\deg(A_2) \leq \deg(A)$; $\deg(A_1^2)$, $\deg(A_2^2) \leq 2(\mu - 1)$; $|A_1^2|_\infty$, $|A_2^2|_\infty \leq \mu a^2$; $|A_1 A_2|_\infty \leq \mu a^2$.

The theorem follows by observing that $\mu L(\mu a^2) \sim \mu L(\mu) + \mu L(a) \propto \mu^2 L(\mu) + \mu^2 L(a)^2$. ∎

As an illustration of speed gain, the following test runs were made on randomly generated polynomials using a PDP-10 with $\beta = 2^{31}$. The units for $L(|A|_\infty)$ are β-digits (βd) and the units of time are seconds (s).

deg(A)	$L(\|A\|_\infty)$ (βd)	PPROD (s)	PSQR (s)	GPPROD (s)	GPSQR (s)
5	2			2.90	1.43
10	2			10.1	5.18
15	2			19.8	10.4
20	2			37.0	18.4
5	3	.753	.383	3.43	1.70
10	3	3.33	1.60	11.1	6.03
15	3	7.13	3.53	26.5	15.1
20	3	10.4	5.35	47.2	22.7
5	4	1.10	.584		
10	4	3.95	2.00		
15	4	8.30	4.48		
20	4	14.5	7.67		

The following algorithms, then, use the previous algorithms of this section and the method of Theorem 4.1.1 to perform root squaring.

ALGORITHM PRSQR:

B=PRSQR(A)

Polynomial with Roots Squared

A is a univariate integral polynomial of positive degree n, $A(x) = \{ldcf(A)\}\prod_{i=1}^{n}(x - \alpha_i)$. B is the univariate integral polynomial B(x) =

$$(-1)^n \{ldcf(A)\} \prod_{i=1}^{2n} (x - \alpha_i).$$

Method:

Let $A(x) = \sum_{i=0}^{n} a_i x^i$ where $n = deg(A)$. Let $A_1(x) = \sum_{\substack{i=0 \\ i \text{ even}}}^{n} a_i x^i$ and $A_2(x) = \sum_{\substack{i=0 \\ i \text{ odd}}}^{n} a_i x^i$. Let $B' = A_1^2 - A_2^2$.

Then $B(x^2) = B'(x)$.

Description:

(1) [Split A.] PSPLT(A,A_1,A_2).

(2) [Compute B'.] A_1' ← PSQR(A_1); A_2' ← PSQR(A_2);

erase A_1, A_2; B' ← PDIF(A_1',A_2'); erase A_1', A_2'.

(3) [Finish.] B ← PHAEX(B'); erase B'; return.

Computing Time: $\propto \mu^2 L(a)^2 + \mu^2 L(\mu)$, where $m = deg(A)$, $\mu = m + 1$, and $a = |A|_\infty$.

Proof:

Step	n_i	t_i
1	1	$\propto \mu$
2	1	$\propto \mu^2 L(a)^2 + \mu^2 L(\mu) + \mu L(\mu a^2)$
3	1	$\propto \mu$

$|A_1|_\infty^2 \leq \mu a^2$ and $|A_2|_\infty^2 \leq \mu a^2$. The theorem follows by

observing that $\mu L(\mu a^2) \sim \mu L(\mu) + \mu L(a) \leq \mu^2 L(\mu) + \mu^2 L(a)$. ∎

ALGORITHM GPRSQR:

B=GPRSQR(A)

Gaussian Polynomial with Roots Squared

A is a univariate Gaussian polynomial of positive degree n, $A(z) = \{ldcf(A)\}\prod_{i=1}^{n} (z - \alpha_i)$. B is the univariate Gaussian polynomial B(z) =

$$(-1)^n \{ldcf(A)\}^2 \prod_{i=1}^{n} (z - \alpha_i^2).$$

Method:

Let $A(z) = \sum_{i=0}^{n} a_i z^i$, where n = deg(A). Let $A_1(z) = \sum_{\substack{i=0 \\ i \text{ even}}}^{n} a_i z^i$ and $A_2(z) = \sum_{\substack{i=0 \\ i \text{ odd}}}^{n} a_i z^i$. Let $B' = A_1^2 - A_2^2$.

Then $B(z^2) = B'(z)$.

Description:

(1) [Split A.] GPSPLT(A,A_1,A_2).

(2) [Compute B'.] A_1' ← GPSQR(A_1); A_2' ← GPSQR(A_2); erase A_1, A_2; B' ← GPDIF(A_1',A_2'); erase A_1', A_2'.

(3) [Finish.] B ← GPHAEX(B'); erase B'; return.

Computing Time: $\underline{\propto} \mu^2 L(a)^2 + \mu^2 L(\mu)$, where $m = \deg(A)$, $\mu = m + 1$, and $a = |A|_\infty$.

Proof:

Step	n_i	t_i
1	1	$\underline{\propto} \mu$
2	1	$\underline{\propto} \mu^2 L(a)^2 + \mu^2 L(\mu) + \mu L(\mu a^2)$
3	1	$\underline{\propto} \mu$

$|A_1|_\infty \leq \mu a^2$, $|A_2|_\infty \leq \mu a^2$. ∎

4.3 Mappings

This section presents algorithms for mappings which are used by later algorithms in order to locate roots in areas other than the basic areas of upper and lower half-planes and the real axis.

In most cases it will be obvious that the criteria for suitable mappings as discussed in Section 4.1 are satisfied; in those where it is not, an explanation will be given.

The first four algorithms may appear to implement somewhat trivial Φ's. However, their major importance lies not in the Φ per se, which simply maps an area to its expanded or contracted image (such as the unit circle to a circle of radius r). Rather, as shown later, they make it possible to work with rational, as well as integral, expressions for other Φ's.

Let $A(x) = \sum_{i=0}^{n} a_i x^i$ be a univariate integral or Gaussian polynomial. Homothetic expansion by a non-zero integer c will be the polynomial $B(x) = A(cx) = \sum_{i=0}^{n} a_i c^i x^i$. Thus the mapping function Φ is $\Phi(x) = cx$.

Strictly speaking, it would seem that corresponding contraction should be $\Phi(x) = x/c$. Note, however, that the resulting $A(\Phi(x))$ is a rational polynomial. Hence, using

the terminology of Section 4.1, let $D(x) = c^n$ and $B(x) = D(x)A(\Phi(x)) = \sum_{i=0}^{n} a_i c^{n-i} x^i$, an integral polynomial. Thus, homothetic contraction by a non-zero integer c will be the polynomial $B(x) = c^n A(x/c)$ for $n = \deg(A)$.

The first two algorithms, then, perform homothetic expansion.

ALGORITHM PHOME:

B=PHOME(A,c)

Polynomial Homothetic Expansion

A is a univariate integral polynomial. c is an L-integer. B is the univariate integral polynomial $B(x) = A(cx)$.

Method:

If $A = 0$, then $B = 0$. Otherwise, if $A(x) = \sum_{i=0}^{m} a_i x^i$, then $B(x) = \sum_{i=0}^{m} (c^i a_i) x^i$.

Description:

(1) [Initialize.] B ← 0; if A = 0, return; A' ← CINV(TAIL(A)); i ← 0; d ← PFA(1,0).

(2) [Obtain term of A.] DECAP2(j,a,A').

(3) [Compute power of c.] If $i \geq j$, go to (5).

(4) [Multiply.] d' ← IPROD(d,c); erase d; d ← d'; i ← i + 1; go to (3).

(5) [Compute term of B.] b ← IPROD(d,a); erase a; B ← PFL(b,PFA(j,B)); if A' ≠ 0, go to (2).

(6) [Finish.] Erase d; B ← PFL(PVBL(A),B); return.

Computing Time: $\preceq \mu^2 L(c)^2 + \mu^2 L(a)L(c)$, where m = deg(A), μ = m + 1, and $a = |A|_\infty$.

Proof:

Step	n_i	t_i
1	1	$\preceq \mu$
2	$\leq \mu$	~ 1
3	$\leq 2\mu$	~ 1
4	$\leq \mu$	$\preceq \mu L(c)^2$
5	$\leq \mu$	$\preceq \mu L(a)L(c)$
6	1	~ 1

The i-th execution of Step (4) computes $c^{i-1} \cdot c$, thus the time $t_3 \preceq L(c^{i-1})L(c) \preceq L(c^{\mu})L(c) \preceq \mu L(c)^2$.

Let $A(x) = \sum_{j=0}^{m} a_j x^j$ for m = deg(A). Then the i-th execution of Step (5) computes $a_j c^j$ for some j, $0 \leq j \leq m$, and hence $t_4 \preceq L(a_j)L(c^j) \preceq L(a_j)L(c^{\mu}) \preceq \mu L(a)L(c)$. ∎

ALGORITHM GPHOME:

B=GPHOME(A,c)

Gaussian Polynomial Homothetic Expansion

A is a univariate Gaussian polynomial. c is an L-integer. B is the univariate Gaussian polynomial $B(x) = A(cx)$.

Description:

(1) [Initialize.] $B \leftarrow 0$; if $A = 0$, return.

(2) [Obtain terms of A.] FIRST2(A_1,A_2,A).

(3) [Compute terms of B.] $B_1 \leftarrow$ PHOME(A_1,c); $B_2 \leftarrow$ PHOME(A_2,c).

(4) [Finish.] $B \leftarrow$ PFL(B_1,PFL(B_2,0)); return.

Computing Time: $\propto \nu^2 L(c)^2 + \nu^2 L(c)L(a)$, where $n = \deg(A)$, $\nu = n + 1$, and $a = |A|_\infty$.

Proof:

Step	n_i	t_i
1	1	~ 1
2	≤ 1	~ 1
3	≤ 1	$\propto \nu^2 L(c)^2 + \nu^2 L(a)L(c)$
4	≤ 1	~ 1

Let $A = A_1 + iA_2$. Then t_3 follows directly since

$|A_1|_\infty$, $|A_2|_\infty \leq |A|_\infty$ and $\deg(A_1)$, $\deg(A_2) \leq \deg(A)$. ∎

The next two algorithms perform homothetic contraction. The method employed, as discussed previously, can be viewed less formally by simply recalling that multiplying a polynomial by a non-zero constant does not affect its roots. Hence the integral polynomial $c^n A(x/c)$ has the same roots as the rational polynomial $A(x/c)$.

ALGORITHM PHOMC:

B=PHOMC(A,c,m)

Polynomial Homothetic Contraction

A is a univariate integral polynomial. c is a non-zero L-integer. m is a Fortran integer, with $m \geq \deg(A)$. B is the univariate integral polynomial $B(x) = c^m A(x/c)$.

Method:

If A = 0, then B = 0. Otherwise, let $A(x) = \sum_{i=0}^{n} a_i x^i$ where $n = \deg(A)$. Then $B(x) = \sum_{i=0}^{n} (c^{m-i} a_i) x^i$.

Description:

(1) [Initialize.] B ← 0; if A = 0, return; A' ← TAIL(A); i ← m; d ← PFA(1,0).

(2) [Obtain term of A.] ADV2(a,j,A').

(3) [Compute power of c.] If $i \leq j$, go to (5).

(4) [Multiply.] d' ← IPROD(d,c); erase d; d ← d';

$i \leftarrow i - 1$; go to (3).

(5) [Compute term of B.] $b \leftarrow IPROD(d,a)$; $B \leftarrow PFA(j, PFL(b,B))$; if $A' \neq 0$, go to (2).

(6) [Finish.] Erase d; $B \leftarrow PFL(PVBL(A),INV(B))$; return.

Computing Time: $\underline{\propto} \mu^2 L(c)^2 + \mu\nu L(a)L(c)$, where $\mu = m + 1$, $n = \deg(A)$, $\nu = n + 1$, and $a = |A|_\infty$.

Proof:

Step	n_i	t_i
1	1	~ 1
2	$\leq \nu$	~ 1
3	$\leq \mu + \nu$	~ 1
4	$\leq \mu$	$\underline{\propto} \mu L(c)^2$
5	$\leq \nu$	$\underline{\propto} \mu L(a)L(c)$
6	1	$\underline{\propto} \nu$

The i-th execution of Step (4) computes $c^{i-1} \cdot c$ and hence $t_4 \underline{\propto} L(c^{i-1})L(c) \underline{\leq} L(c^{\mu})L(c) \underline{\propto} \mu L(c)^2$.

Let $A(x) = \sum_{j=0}^{n} a_j x^j$ for $n = \deg(A)$. Then the i-th execution of Step (5) computes $a_j c^{m-j}$ for some j, $0 \leq j \leq n$. Hence $t_4 \underline{\propto} L(c^{m-j})L(a_j) \underline{\leq} L(c^{\mu})L(a) \underline{\propto} \mu L(c)L(a)$.

ALGORITHM GPHOMC:

B=GPHOMC(A,c)

Gaussian Polynomial Homothetic Contraction

A is a univariate Gaussian polynomial. c is a non-zero L-integer. Let n = deg(A). Then B is the univariate Gaussian polynomial $B(x) = c^n A(x/c)$.

Description:

(1) [Initialize.] B ← 0; if A = 0, return; n ← GPDEG(A).

(2) [Obtain terms of A.] FIRST2(A_1, A_2, A).

(3) [Compute terms of B.] B_1 ← PHOMC(A_1,c,n); B_2 ← PHOMC(A_2,c,n).

(4) [Finish.] B ← PFL(B_1,PFL(B_2,0)); return.

Computing Time: $\propto \nu^2 L(c)^2 + \nu^2 L(c)L(a)$, where n = deg(A), $\nu = n + 1$, and $a = |A|_\infty$.

Proof:

Step	n_i	t_i
1	1	~ 1
2	≤ 1	~ 1
3	≤ 1	$\propto \nu^2 L(c)^2 + \nu^2 L(c)L(a)$
4	≤ 1	~ 1

Let $A = A_1 + iA_2$. Then t_3 follows because $|A_1|_\infty$,

$|A_2|_\infty \leq |A|_\infty$ and $\deg(A_1), \deg(A_2) \leq \deg(A)$. ∎

Combining these two concepts of expansion and contraction, one can let $\Phi(x) = bx/c$ and $D(x) = c^n$ for $n = \deg(A)$, giving the rational homothetic operation by $r = b/c$ on $A(x) = \sum_{i=0}^{n} a_i x^i$ as the polynomial $B(x) = c^n A(bx/c) = \sum_{i=0}^{n} a_i b^i c^{n-i} x^i$. The following algorithm implements this operation for Gaussian polynomials.

ALGORITHM GPRHOM:

B=GPRHOM(A,r)

Gaussian Polynomial Rational

Homothetic Operation

A is a univariate Gaussian polynomial; r is a non-zero rational number, $r = r_1/r_2$. B is the univariate Gaussian polynomial $B(z) = r_2^n A(rz)$, where $n = \deg(A)$.

Description:

(1) [Initialize.] B ← 0; if A = 0, return;

FIRST2(r_1,r_2,r).

(2) [Compute polynomials.] A_1 ← GPHOMC(A,r_2); B ← GPHOME(A_1,r_1); erase A_1; return.

Computing Time: $\leq \mu^2 L(r)^2 + \mu^2 L(r)L(a) + \mu^3 L(r_1)L(r_2)$,

where $m = \deg(A)$, $\mu = m + 1$, $a = |A|_\infty$, and $r = r_1/r_2$.

Proof: $t_{GPHOMC}(A, r_2) \leq \mu^2 L(r_2)^2 + \mu^2 L(r_2)L(a)$. a_1

$= |A_1|_\infty \leq |r_2|^m a$ so $t_{GPHOME}(A_1, r_1) \leq \mu^2 L(r_1)^2 +$

$\mu^2 L(r_1)L(r_2^m a) \leq \mu^2 L(r_1)^2 + \mu^3 L(r_1)L(r_2) + \mu^2 L(r_1)L(a)$

$\leq \mu^2 L(r)^2 + \mu^2 L(r)L(a) + \mu^3 L(r_1)L(r_2)$. ∎

Note that $|B|_\infty \leq |r_1 r_2|^m a$. This fact will be used in later derivations.

The names of the algorithms attempt to indicate the mapping function which they implement. As an example, consider a point p in the complex plane expressed in polar form, $p = be^{i\theta}$. The following algorithm performs a "rotation" in the sense that such a point p is mapped onto point $p^* = be^{i(\theta \pm \pi/2)}$ and hence in effect the plane is rotated $\pm 90^o$ about the origin. The mapping used is $\Phi(z) = \pm iz$.

ALGORITHM GPROT:

B=GPROT(A,s)

Gaussian Polynomial Rotation

A is a univariate Gaussian polynomial. s = 1 or s = -1, a Fortran integer. B is the univariate Gaussian polynomial B(z) = A(siz).

Description:

(1) [Initialize.] $B \leftarrow 0$; if $A = 0$, return; $n \leftarrow GPDEG(A)$; if $n = 0$, ($B \leftarrow BORROW(A)$; return); $FIRST2(A_1, A_2, A)$; if $A_1 \neq 0$, $A_1 \leftarrow TAIL(A_1)$; if $A_2 \neq 0$, $A_2 \leftarrow TAIL(A_2)$; $B_1 \leftarrow 0$; $B_2 \leftarrow 0$.

(2) [Obtain a_1.] $a_1 \leftarrow 0$; if $A_1 = 0$, go to (3); $n_1 \leftarrow SECOND(A_1)$; if $n > n_1$, go to (3); $ADV(a_1, A_1)$; $A_1 \leftarrow TAIL(A_1)$.

(3) [Obtain a_2.] $a_2 \leftarrow 0$; if $A_2 = 0$, go to (4); $n_2 \leftarrow SECOND(A_2)$; if $n > n_2$, go to (4); $ADV(a_2, A_2)$; $A_2 \leftarrow TAIL(A_2)$.

(4) [Compute $k = n \bmod 4$ and branch.] $k \leftarrow MOD(n,4)$; $k' \leftarrow k + 1$; go to (5,6,7,8),k'.

(5) [k = 0.] $b_1 \leftarrow BORROW(a_1)$; $b_2 \leftarrow BORROW(a_2)$; go to (9).

(6) [k = 1.] If s < 0, ($b_1 \leftarrow$ BORROW(a_1); $b_2 \leftarrow$ INEG(a_2); go to (9)); if s > 0, ($b_1 \leftarrow$ INEG(a_1); $b_2 \leftarrow$ BORROW(a_2); go to (9)).

(7) [k = 2.] $b_1 \leftarrow$ INEG(a_1); $b_2 \leftarrow$ INEG(a_2); go to (9).

(8) [k = 3.] If s < 0, ($b_1 \leftarrow$ INEG(a_1); $b_2 \leftarrow$ BORROW(a_2); go to (9)); if s > 0, ($b_1 \leftarrow$ BORROW(a_1); $b_2 \leftarrow$ INEG(a_2)).

(9) [Prefix terms of B.] If $b_1 \neq 0$, $B_1 \leftarrow$ PFA(n,PFL(b_1, B_1)); if $b_2 \neq 0$, $B_2 \leftarrow$ PFA(n,PFL(b_2,B_2)); $n \leftarrow n - 1$; if $n \geq 0$, go to (2).

(10) [Finish.] If $B_1 \neq 0$, $B_1 \leftarrow$ PFL(GPVBL(A),INV(B_1)); if $B_2 \neq 0$, $B_2 \leftarrow$ PFL(GPVBL(A),INV(B_2)); $B \leftarrow$ PFL(B_1, PFL(B_2,0)); return.

Computing Time: $\propto \nu L(a)$, where $n = \deg(A)$, $\nu = n + 1$, and $a = |A|_\infty$.

Proof:

Step	n_i	t_i
1	1	~ 1
2	$\leq \nu$	~ 1
3	$\leq \nu$	~ 1
4	$\leq \nu$	~ 1

5	$\leq \nu$	~ 1
6	$\leq \nu$	$\propto L(a)$
7	$\leq \nu$	$\propto L(a)$
8	$\leq \nu$	$\propto L(a)$
9	$\leq \nu$	~ 1
10	≤ 1	$\propto \nu$

Translation is an often-used mapping, with $\Phi(x) = x + a$ for some integer a. The following two algorithms perform this translation.

ALGORITHM PTRAN:

B=PTRAN(A,c)

Polynomial Translation

A is a univariate integral polynomial. c is an L-integer. B is the univariate integral polynomial $B(x) = A(x + c)$.

Method:

If deg(A) = 0 or c = 0, then B = A. Otherwise, Horner's algorithm is used as follows. Let $A(x) = \sum_{i=0}^{n} a_i x^i$ with deg(A) = n. Let $B_0(x) = a_n$ and $B_{i+1}(x) = B_i(x) \cdot (x + c) + a_{n-i-1}$ for $0 \leq i < n$. Then $B = B_n$.

Description:

(1)[Deg(A) or c zero?] If PDEG(A) = 0 or c = 0, (B ← BORROW(A); return).

(2)[Initialize.] x ← FIRST(A); A' ← TAIL(A);

ADV2(a,n,A'); B ← PFL(PVBL(A),PFL(BORROW(a), PFA(0,0))); i ← 0.

(3) [Multiply by x + c.] B' ← PMLMP(B,c); erase B; B ← B'; i ← i + 1.

(4) [Add coefficients.] If A' = 0, go to (5); j ← SECOND(A'); if $j \neq n - i$, go to (5); ADV(a,A'); A' ← TAIL(A'); B' ← PISUM(B,x,a); erase B; B ← B'.

(5) [Test for end.] If $i < n$, go to (3); return.

Computing Time: $\propto \nu^2 L(a)L(c) + \nu^3 L(c)^2$, where n = deg(A), $\nu = n + 1$, and $a = |A|_\infty$.

Proof:

Step	n_i	t_i
1	1	~ 1
2	≤ 1	~ 1
3	$\leq \nu$	$\propto \nu L(a)L(c) + \nu^2 L(c)^2$
4	$\leq \nu$	$\propto \nu L(c) + L(a)$
5	$\leq \nu$	~ 1

Let $A(x) = \sum_{i=0}^{n} a_i x^i$ where n = deg(A). Define $B_0(x) = a_n$ and $B_{j+1}(x) = B_j(x) \cdot (x + c) + a_{n-j-1}$ for $0 \leq j \leq n$. Thus B_j is the value of B at the beginning of the (j + 1)-th execution of Step (3). Then $B_{j+1}(x)$

$= B_j(x)\cdot(x + c) + a_{n-j-1} = \{B_j(x)\cdot x + a_{n-j-1}\} + c\cdot B_j(x)$. Clearly $|B_j(x)\cdot x + a_{n-j-1}|_\infty \leq \max\{|B_j(x)\cdot x|_\infty, |a_{n-j-1}|\} \leq \max\{|B_j(x)|_\infty, a\}$, and $|c\cdot B_j(x)|_\infty = c^*|B_j(x)|_\infty$ for $c^* = |c|$. Thus $|B_{j+1}(x)|_\infty \leq \max\{|B_j(x)\cdot x|_\infty, a\} + c^*|B_j(x)|_\infty$. Hence if $|B_j(x)|_\infty \leq a(c^* + 1)^j$, then $|B_{j+1}(x)|_\infty \leq a(c^* + 1)^j + c^*a(c^* + 1)^j = a(c^* + 1)^{j+1}$. Since $\deg(B_j) = j$ and $0 \leq j < \nu$, then

$$t_3 \propto \nu L(a\{c^* + 1\}^\nu)L(c) \propto \nu L(a)L(c) + \nu L(\{c^* + 1\}^\nu)L(c) \propto \nu L(a)L(c) + \nu^2 L(c)^2.$$

Similarly, $t_4 \propto \nu + L(a) + L(a\{c^* + 1\}^\nu) \propto \nu L(c) + L(a)$. ∎

ALGORITHM GPTRAN:

B=GPTRAN(A,c)

Gaussian Polynomial Translation

A is a univariate Gaussian polynomial. c is an L-integer. B is the univariate Gaussian polynomial $B(x) = A(x + c)$.

Description:

(1) [Deg(A) or c zero?] If GPDEG(A) = 0 or c = 0, (B ← BORROW(A); return).

(2) [Obtain terms of A.] FIRST2(A_1, A_2, A).

(3) [Compute terms of B.] B_1 ← PTRAN(A_1, c); B_2 ← PTRAN(A_2, c).

(4) [Finish.] B ← PFL(B_1, PFL(B_2, 0)); return.

Computing Time: $\propto \nu^2 L(c)L(a) + \nu^3 L(c)^2$, where n = deg(A), $\nu = n + 1$, and $a = |A|_\infty$.

Proof:

Step	n_i	t_i
1	1	~ 1
2	≤ 1	~ 1
3	≤ 1	$\propto \nu^2 L(a)L(c) + \nu^3 L(c)^2$
4	≤ 1	~ 1

Let $A = A_1 + iA_2$. Then t_3 follows directly since $|A_1|_\infty$, $|A_2|_\infty \leq |A|_\infty$ and $\deg(A_1)$, $\deg(A_2) \leq \deg(A)$. ∎

When considering a Gaussian polynomial, it is possible to do a "double translation" using $\Phi(z) = z + a + bi$. (Note, of course, one could also do this with an integral

polynomial. However, the result would be a Gaussian polynomial, and such modification of input to output polynomial forms is avoided. Similarly, there is no PROT. If necessary, one can easily form the Gaussian polynomial A* = A + 0i from integral polynomial A and use A*.)

The following algorithm implements double translation of a Gaussian polynomial. Notable simplification is achieved by considering Φ as a composition of functions rather than forming z + a + bi directly. The functions used are $\Phi_1(z) = iz$, $\Phi_2(z) = z + b$, $\Phi_3(z) = -iz$, and $\Phi_4(z) = z + a$. Then $\Phi(z) = \Phi_1(\Phi_2(\Phi_3(\Phi_4(z)))) = \Phi_1(\Phi_2(\Phi_3(z + a)))$ $= \Phi_1(\Phi_2(-i(z + a))) = \Phi_1(-i(z + a) + b) = i(-i(z + a) + b)$ $= z + a + bi$.

ALGORITHM GPDTR:

B=GPDTR(A,a,b)

Gaussian Polynomial Double Translation

A is a univariate Gaussian polynomial. a and b are L-integers. B is the univariate Gaussian polynomial B(z) = A(z + a + bi).

Description:

(1)[Complex translation.] $A_1 \leftarrow$ GPROT(A_1,1); $A_2 \leftarrow$ GPTRAN(A_1,b); erase A_1; $A_3 \leftarrow$ GPROT(A_2,-1); erase A_2.

(2)[Real translation.] $B \leftarrow$ GPTRAN(A_3,a); erase A_3;

return.

Computing Time: $\underline{\propto} \mu^3 L(ab)^2 + \mu^2 L(ab)L(d)$, where $m = \deg(A)$, $\mu = m + 1$, and $d = |A|_\infty$.

Proof: $t_{GPROT}(A,1) \underline{\propto} \mu L(d)$ and $d_1 = |A_1|_\infty = d$.

$t_{GPTRAN}(A_1,b) \underline{\propto} \mu^2 L(b)L(d) + \mu^3 L(b)^2$ and $d_2 \underline{\leq} (|b| + 1)^m d$. $t_{GPROT}(A_2,1) \underline{\propto} \mu L(d_2) \underline{\propto} \mu^2 L(b) + \mu L(d)$ and $d_3 = |A_3|_\infty = d_2$. $t_{GPTRAN}(A_3,a) \underline{\propto} \mu^2 L(d_3)L(a) + \mu^3 L(a)^2 \underline{\propto} \mu^3 L(a)L(b) + \mu^2 L(a)L(d) + \mu^3 L(a)^2 \sim \mu^3 L(a)L(ab) + \mu^2 L(a)L(d)$. So $t_{GPDTR}(A,a,b) \underline{\propto} \mu^2 L(b)L(d) + \mu^3 L(b)^2 + \mu^2 L(a)L(d) + \mu^3 L(a)L(ab) \sim \mu^3 L(ab)^2 + \mu^2 L(ab)L(d)$. ∎

This double translation is naturally extended to a rational complex point a/b + ic/d. The first step is to obtain a common denominator and put the coordinates into the form (e + if)/g. Then a composition of functions can again be used, $\Phi_1(z) = z/g$, $\Phi_2(z) = z + e + if$, and $\Phi_3(z) = gz$.

ALGORITHM GPDRTR:

B=GPDRTR(A,r_1,r_2)

Gaussian Polynomial Double

Rational Translation

A is a univariate Gaussian polynomial. r_1 and r_2 are rational numbers. Let $r_1 = r_{11}/r_{12}$ and $r_2 = r_{21}/r_{22}$, $c = \mathrm{lcm}(r_{12}, r_{22})$, and $n = \deg(A)$. Then $B(z) = c^n A(z + r_1 + ir_2)$, a Gaussian polynomial.

Description:

(1) [Initialize.] If GPDEG(A) = 0 or ($r_1 = 0$ and $r_2 = 0$), (B ← BORROW(A); return); CZRPCD(r_1,r_2,a,b,c).

(2) [Translate.] A_1 ← GPHOMC(A,c); A_2 ← GPDTR(A_1,a,b); erase A_1, a, b; B ← GPHOME(A_2,c) erase A_2, c; return.

Computing Time: $\propto \mu^3\{L(r_1) + L(r_2)\} + \mu^2\{L(r_1) + L(r_2)\}L(d)$, where $m = \deg(A)$, $\mu = m + 1$, and $d = |A|_\infty$.

Proof: Let $\rho_i = L(r_i)$, $\rho = \rho_1 + \rho_2$. $t_{CZRPCD}(r_1,r_2) \propto \rho_1\rho_2 \leq \rho^2$. $L(a), L(b), L(c) \leq \rho$, so $t_{GPHOMC}(A,c)$

$\underline{\propto} \mu^2\rho^2 + \mu^2\rho L(d)$. $d_1 = |A_1|_\infty \le |c|^m d$ so $L(d_1) \underline{\propto} \mu\rho + L(d)$. Hence $t_{GPDTR}(A_1,a,b) \underline{\propto} \mu^3 L(ab)^2 + \mu^2 L(ab)L(d_1)$

$\underline{\propto} \mu^3\rho^2 + \mu^2\rho(\mu\rho + L(d)) \sim \mu^3\rho^2 + \mu^2\rho L(d)$. $d_2 = |A_2|_\infty$

$\le (|a| + 1)^m (|b| + 1)^m d_1 \le (|a| + 1)^m (|b| + 1)^m |c|^m d$

and $L(d_2) \underline{\propto} \mu\rho + L(d)$. Hence $t_{GPHOME}(A_2,c) \underline{\propto} \mu^2 L(c)^2$

$+ \mu^2 L(c)L(d_2) \underline{\propto} \mu^2\rho^2 + \mu^2\rho(\mu\rho + L(d)) \sim \mu^3\rho^2 +$

$\mu^2\rho L(d)$. So $t_{GPDRTR}(A,r_1,r_2) \underline{\propto} \mu^3\rho^2 + \mu^2\rho L(d)$. ∎

Note that $|B|_\infty \le |c|^m d_2 \le (|a| + 1)^m (|b| + 1)^m |c|^{2m} d$

$\le (r + 1)^{8m} d$, where $r = \max(|r_{11}|, |r_{12}|, |r_{21}|, |r_{22}|)$.

The following algorithm, which maps the upper half-plane to the unit circle, requires some documentation.

Let $\Phi(z) = (z - i)/(z + i)$ for $z \neq -i$. If $w = (z - i)/(z + i)$ then $wz + wi = z - i$ so $(w - 1)z = -i(w + 1)$, $w \neq 1$, and $z = -i(w + 1)/(w - 1)$. Conversely, if $w \neq 1$ and $z = -i(w + 1)/(w - 1)$ then $wz - z = -wi - i$ so $(z + i)w = z - i$, $z \neq -i$, and $w = (z - i)/(z + i)$. This shows that

Φ is one-one from $\{z: z \neq -i\}$ onto $\{w: w \neq 1\}$.

Let $z = a + bi$, where a and b are real, and $z \neq -i$.

Then $\Phi(z) = \{a + (b - 1)i\}/\{a + (b + 1)i\} = \{(a^2 + b^2 - 1) - 2ai\}/\{a^2 + (b + 1)^2\}$. Hence $|\Phi(z)|^2 = \{a^4 + 2a^2b^2 + b^4 + 2a^2 + 1 - 2b^2\}/\{a^4 + 2a^2b^2 + b^4 + 2a^2 + 1 + 6b^2 + 4a^2b + 4b^3 + 4b\} = u/v$. Since $v - u = 4b\{a^2 + (b + 1)^2\}$, $|\Phi(z)| < 1$ if $b > 0$, $|\Phi(z)| = 1$ if $b = 0$, and $|\Phi(z)| > 1$ if $b < 0$.

Hence Φ maps the upper half-plane $\{a + bi: b > 0\}$ onto the interior of the unit circle $\{w: |w| < 1\}$, maps the real axis $\{a + bi: b = 0\}$ onto the cut unit circle $\{w: |w| = 1$ and $w \neq 1\}$, and maps the punctured lower half-plane $\{a + bi: b < 0$ and $a + bi \neq -i\}$ onto the exterior of the unit circle.

Let $C(z) = A(\Phi(z)) = A((z - i)/(z + i))$ and $D(z) = (z + i)^n$ where $n = \deg(A)$. Note that $D(z) \neq 0$ for z in the upper half-plane.

Then $B(z) = D(z)C(z) = (z + i)^n A((z - i)/(z + i))$ is a Gaussian polynomial which has the same number of zeros in the upper half-plane as A(z) has in the unit circle and the same number on the real axis as A(z) has on the unit circle excluding point $(1 + 0i)$.

The following algorithm computes the polynomial $B(z) = (z + i)^n A((z - i)/(z + i))$.

ALGORITHM GPHPTC:

B=GPHPTC(A)

Gaussian Polynomial, Half

Plane To Circle Mapping

A is a univariate Gaussian polynomial. B is the univariate Gaussian polynomial $B(z) = (z + i)^n A((z - i)/(z + i))$, where $n = \deg(A)$.

Method:

Let $A(z) = \sum_{j=0}^{n} a_j z^j$ and $B_0(z) = a_n$, $D_0(z) = 1$, $D_{k+1}(z) = (z + i)D_k(z)$, $B_{k+1}(z) = (z - i)B_k(z) + a_{n-k-1} D_{k+1}(z)$; then $B(z) = B_n(z)$.

Description:

(1) [Initialize.] n ← GPDEG(A); if n = 0, (B ← BORROW(A); return); a ← GPLDCF(A); FIRST2(a_1,a_2,a); B ← GPCONS(A,a_1,0,a_2,0); erase a; A' ← GPRED(A); e ← PFA(1,0); f ← PFA(-1,0); D ← GPCONS(A,e,0,0,0); E ← GPCONS(A,e,1,e,0); F ← GPCONS(A,e,1,f,0); erase e, f; k ← 0.

(2) [Apply recurrence.] D' ← GPPROD(D,E); erase D; D ← D'; B' ← GPPROD(B,F); erase B; B ← B'; n' ← GPDEG(A'); if n' ← n - k - 1, go to (3); a ← GPLDCF(A'); A'' ← GPRED(A'); erase A'; A' ← A''; D' ← GPSPRD(D,a,0);

erase a; $B' \leftarrow GPSUM(B,D')$; erase B, D'; $B \leftarrow B'$.

(3) [Terminate.] $k \leftarrow k + 1$; if $k < n$, go to (2);

erase D, E, F; return.

Computing Time: $\propto \nu^3 + \nu^2 L(d)$, where $n = \deg(A)$, $\nu = n + 1$, and $d = |A|_\infty$.

Proof: $\deg(E), \deg(F) = 1$ and $|E|_\infty, |F|_\infty = 1$.

Consider the j-th execution of Step (2). $D = (z + i)^{j-1}$ so $\deg(D) = j - 1$ and $|D|_1 = 2^{j-1}$. Hence

$t_{GPPROD}(D,E) \propto (j - 1)L(2^{j-1}) \propto j^2 L(2) \sim j^2$. Then

$\deg(D') = j$ and $|D'|_1 = 2^j$. $B = \sum_{k=0}^{j-1} a_{n-k} (z - i)^{j-1-k} (z + i)^k$ so $\deg(B) = j - 1$ and $|B|_1 \leq jd2^{j-1}$.

Hence $t_{GPPROD}(B,F) \propto (j - 1)L(jd2^{j-1}) \propto jL(d) + j^2$.

Then $|B'|_1 \leq jd2^j$ and $\deg(B') = j$. $t_{GPSPRD}(D,a) \propto jL(d)L(2^j) \propto j^2 L(d)$. $|D'|_\infty \leq d2^j$ and $\deg(D') \leq j$.

So $t_{GPSUM}(B,D') \leq jL(jd2^j) + jL(d2^j) \propto j^2 + jL(d)$.

Hence the time for all executions is $\sum_{j=1}^{n} \{j^2 +$

$$jL(d)\} \underline{\alpha} \nu^3 + \nu^2 L(d). \quad \square$$

Note that $|B|_1 \leq nd2^n$. This fact will be used in later computing time derivations.

4.4 Applications

This section applies root squaring and mappings to compute the number of roots of a polynomial in various areas of the plane.

The first algorithm computes the number of roots in an arbitrary vertical half-plane, that is, right of and on the vertical line $z = r$ for a rational number r. The mappings required are a rotation of 90^{o} and a translation to r, thus taking the upper half-plane to the specified right half-plane. Homothetic operations are used, of course, if r is a non-zero rational number $r = r_1/r_2$. The successive functions applied are $\Phi_1(z) = z/r_2$, $\Phi_2(z) = z + r_1$, $\Phi_3(z) = r_2 z$, and $\Phi_4(z) = -iz$, giving a composition $\Phi_1(\Phi_2(\Phi_3(\Phi_4(z))))$ $= \Phi_1(\Phi_2(\Phi_3(-iz))) = \Phi_1(\Phi_2(-ir_2 z)) = \Phi_1(-ir_2 z + r_1) = (-ir_2 z + r_1)/r_2 = -iz + r_1/r_2$.

ALGORITHM CZNZVH:

CZNZVH(A,r,ℓ,m)

Complex Zero System, Number of Zeros in a Vertical Half-Plane

A and r are inputs; ℓ and m are outputs. A is a non-zero square-free univariate Gaussian polynomial. r is a rational number. ℓ is the number of zeros of

A on the line z = r, m is the number right of that line. ℓ and m are Fortran integers.

Description:

(1) [Initialize.] If r = 0, ($A_3 \leftarrow$ BORROW(A); go to (2)); FIRST2(h,k,r); $A_1 \leftarrow$ GPHOMC(A,k); $A_2 \leftarrow$ GPTRAN(A_1,h); erase A_1; $A_3 \leftarrow$ GPHOME(A_2,k); erase A_2.

(2) [Finish.] $A_4 \leftarrow$ GPROT(A_3,-1); erase A_3; CZNZHP(A_4, ℓ,m); erase A_4; return.

Computing Time: $\preceq \mu^{4+s}\{\mu L(\mu hk) + L(d)\}^2$, where r = h/k, m = deg(A), μ = m + 1, and $d = |A|_\infty$; s = 0 if modular methods are used or the p.r.s.'s are normal, s = 1 otherwise.

Proof: Let $d_i = |A_i|_\infty$. $t_{GPHOMC}(A,k) \preceq \mu^2 L(k)L(kd)$. $d_1 \leq k^m d$ so $t_{GPTRAN}(A_1,h) \preceq \mu^2 L(h)L(d_1) + \mu^3 L(h)^2 \preceq \mu^3 L(h)L(k) + \mu^2 L(h)L(d) + \mu^3 L(h)^2$. $d_2 \leq (|h| + 1)^m k^m d$ so $t_{GPHOME}(A_2,k) \preceq \mu^2 L(k)L(kd_2) \preceq \mu^3 L(k)L(hk) + \mu^2 L(k)L(d)$. Hence $t_1 \preceq \mu^3 L(k)L(hk) + \mu^3 L(h)^2 +$

$$\mu^2 L(k)L(kd) + \mu^2 L(h)L(d) \sim \mu^3 L(hk)^2 + \mu^2 L(hk)L(d).$$

$d_3 \leq (|h| + 1)^m k^{2m} d$ so $t_{GPROT}(A_3) \underline{\propto} \mu^2 L(hk) +$

$\mu L(d)$. $d_4 = d_3 \leq (|h| + 1)^m k^{2m} d$ so $L(d_4) \underline{\propto} \mu L(hk) +$

$L(d)$. Hence $t_{CZNZHP}(A_4) \underline{\propto} \mu^{6+s} L(\mu)^2 + \mu^{5+s} L(\mu)L(d_4) +$

$$\mu^{4+s} L(d_4)^2 \underline{\propto} \mu^{6+s} L(\mu)^2 + \mu^{6+s} L(\mu)L(hk) + \mu^{5+s} L(\mu)L(d)$$

$$+ \mu^{4+s} L(d_4)^2 \underline{\propto} \mu^{6+s} L(\mu)^2 + \mu^{6+s} L(\mu)L(hk) + \mu^{5+s} L(\mu)L(d)$$

$$+ \mu^{6+s} L(hk)^2 + \mu^{5+s} L(hk)L(d) + \mu^{4+s} L(d)^2 \sim \mu^{6+s} L(\mu hk)^2$$

$$+ \mu^{5+s} L(\mu hk)L(d) + \mu^{4+s} L(d)^2 \sim \mu^{4+s} \{\mu L(\mu hk) + L(d)\}^2 .$$ ∎

The next algorithm is for arbitrary horizontal half-planes. It may seem that this should have been first, since no rotation is required--$\Phi(z) = z + ir$. However, translation by an imaginary value is difficult when implemented directly. Hence, considerable simplification and little loss of efficiency is achieved by rotating first and then applying the preceding algorithm for vertical half-planes. Then $\Phi_1(z) = iz$ and $\Phi_2(z) = -iz + r$, giving a composition $i(-iz + r) = z + ir$, as required.

ALGORITHM CZNZHH:

CZNZHH(A,r,ℓ,m)

Complex Zero System, Number of Zeros

in a Horizontal Half-Plane

A and r are inputs; ℓ and m are outputs. A is a non-zero square-free univariate Gaussian polynomial. r is a rational number. ℓ and m are Fortran integers. ℓ is the number of zeros of A on the line z = ri and m is the number above that line.

Description:

(1) [Rotate and call CZNZVH.] $A_1 \leftarrow$ GPROT(A,+1);

CZNZVH(A_1,r,ℓ,m); erase A_1; return.

Computing Time: $\propto \mu^{4+s}\{\mu L(\mu hk) + L(d)\}^2$, where m = deg(A), $\mu = m + 1$, $d = |A|_\infty$, and $r = h/k$; s = 0 if modular methods are used or the p.r.s.'s are normal, s = 1 otherwise.

Proof: Let $d_1 = |A_1|_\infty$. $t_{GPROT}(A,+1) \propto \mu L(d)$ and $d_1 = d$. Hence the time for the algorithm is the same as for CZNZVH. ∎

The following algorithm uses an algorithm of Heindel [HEL70] to find the number of real roots of an integral polynomial in a horizontal line interval.

ALGORITHM PNRHL:

n=PNRHL(A,r)

Polynomial, Number of Roots in

a Horizontal Line Interval

A is a non-zero square-free univariate integral polynomial. r is a rational number, $r < 0$. n is the number of roots of A in the interval (r,0].

Description:

(1) [Check degree and initialize.] If PDEG(A) = 0, (n ← 0; return); A' ← PABS(A); I ← PFL(BORROW(r),PFL(0,0)).

(2) [Compute value.] n ← IRTS(A',I); erase A', I; return.

Computing Time: $\propto \mu^3 L(r)^2 + \mu^4 L(r)L(\mu d) + \mu^{4+k} L(\mu d)^2$, where m =deg(A), $\mu = m + 1$, and $d = |A|_\infty$; k = 0 if the p.r.s. is normal, and k = 1 otherwise.

Proof: $t_{PABS}(A) \propto \mu L(d)$, $t_{IRTS}(A',I) \propto \mu^3 L(r)^2 + \mu^4 L(r)L(\mu d) + \mu^{4+k} L(\mu d)^2$. ∎

Rotation can now be employed to find the number of roots in a vertical line interval. Using $\Phi(z) = iz$ with integral polynomial A gives $A^*(z) = A(iz)$, a Gaussian polynomial. Recall, however, that any real roots of A^* must be real roots of $B = \text{symp}(A^*)$. Since B is an integral polynomial, PNRHL can be applied to it.

ALGORITHM GPNRVL:

n=GPNRVL(A,r)

Gaussian Polynomial, Number of Roots

in a Vertical Line Interval

A is a non-zero square-free univariate Gaussian polynomial. r is a rational number, $r < 0$. n is the number of roots of A in the interval (ir,0].

Description:

(1) [Obtain values.] A* ← GPROT(A,+1); n ← GPNRHL(A*,r); erase A*; return.

Computing Time: $\propto \mu^3 L(r)^2 + \mu^4 L(r)L(d) + \mu^5 L(r)L(\mu) + \mu^{4+k}\{\mu L(\mu) + L(d)\}^2$, where $m = \deg(A)$, $\mu = m + 1$, and $d = |A|_\infty$; $k = 0$ if the p.r.s. is normal, $k = 1$ otherwise.

Proof: Follows immediately from the time for GPNRHL by noting that $|A^*|_\infty = |A|_\infty$. ∎

Given the four preceding algorithms in this section, it is an easy matter to compute the number of roots of a Gaussian polynomial on the upper and right hand edges of an arbitrary rectangle. These quantities are required in major algorithm PRIR of Chapter 5.

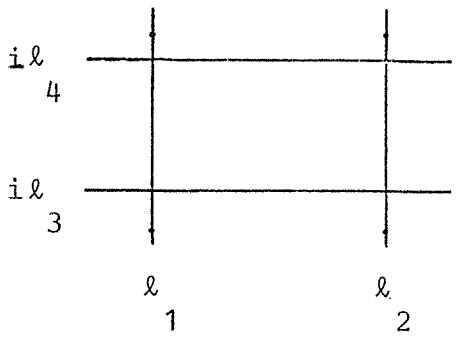
ALGORITHM GPNRER:

GPNRER(A, ℓ_1, ℓ_2, ℓ_3, ℓ_4, n_1, n_2, n_3)

Gaussian Polynomial, Number of

Roots on Edges of a Rectangle

A, ℓ_1, ℓ_2, ℓ_3, ℓ_4 are the inputs. n_1, n_2, n_3 are outputs. A is a non-zero square-free univariate Gaussian polynomial. ℓ_1, ℓ_2, ℓ_3, ℓ_4 are rational numbers representing the following rectangle in the complex plane:



Then n_1 is the number of zeros of A on the top edge (not including end points), n_2 is the number at the point $\ell_2 + i\ell_4$, and n_3 is the number on the right hand edge (not including end points).

Description:

(1) [Compute values.] A* ← GPDRTR(A,ℓ_2,ℓ_4); GPSYMP(A*, B,C); erase A*; n_2 ← PORD(B); B* ← PWRAZR(B); erase B;

$\overline{\ell}_1 \leftarrow$ RDIF(ℓ_1,ℓ_2); $\overline{\ell}_3 \leftarrow$ RDIF(ℓ_3,ℓ_4); $n_1 \leftarrow$ PNRHL(B*,$\overline{\ell}_1$);

$n_3 \leftarrow$ GPNRVL(C,$\overline{\ell}_3$) + PNRVL(B*,$\overline{\ell}_3$); erase B*, C, $\overline{\ell}_1$, $\overline{\ell}_3$;

return.

Computing Time: $\leq \mu^{4+k}\{\mu L(\mu r) + L(d)\}^2$, where $m = \deg(A)$, $\mu = m + 1$, $d = |A|_\infty$, and for $\ell_i = \ell_{i1}/\ell_{i2}$, $r = \max_{1 \leq i \leq 4}\{|\ell_{i1}|, \ell_{i2}\}$; $k = 0$ if the p.r.s.'s are normal, $k = 1$ otherwise.

Proof: $t_{GPDTR}(A,\ell_2,\ell_4) \leq \mu^3 L(r)^2 + \mu^2 L(r)L(d)$ and $a = |A^*|_\infty \leq (r+1)^{8m} d$. Therefore $t_{GPSYMP}(A^*) \leq \mu^3 L(\mu(r+1)^{8m} d)^2 \leq \mu\{\mu L(r) + L(\mu d)\}^3$ and $|B|_\infty, |C|_\infty \leq \mu^{2m+1} a \leq \mu^{2m+1}(r+1)^{8m} d$. $t_{PORD}(B)$, $t_{PWRAZR}(B) \leq \mu$.

$t_{GPDIF}(\ell_1,\ell_2)$, $t_{GPDIF}(\ell_3,\ell_4) \leq L(r)^2$ and $L(\overline{\ell}_1)$, $L(\overline{\ell}_2)$

$\leq L(r)$. Thus, $t_{PNRHL}(B^*,\overline{\ell}_1)$, $t_{PNRVL}(B^*,\overline{\ell}_3)$,

$t_{GPNRVL}(C,\overline{\ell}_3) \leq \mu^3 L(r)^2 + \mu^4 L(r)L(\mu^{2m+1}(r+1)^{8m} d) + \mu^5 L(r)L(\mu)^{4+k} + \mu^{2m+1}(r+1)^{8m} d)\}^2 \leq$

$\mu^3 L(r)^2 + \mu^4 L(r)L(d) + \mu^5 L(r)L(\mu r) + \mu^5 L(r)L(\mu) +$

$$\mu^{4+k}\{\mu L(\mu) + \mu L(\mu r) + L(d)\}^2 \propto \mu^{4+k}\{\mu L(\mu r) + L(d)\}^2 . \square$$

Root squaring and mappings are next used in an important algorithm which computes the number of roots of a Gaussian polynomial in seven areas of the complex plane: [0,0], $(-\infty,0)$, $(0,+\infty)$, $(-i\infty,0)$, $(0,+i\infty)$, quadrants I and III, and quadrants II and IV.

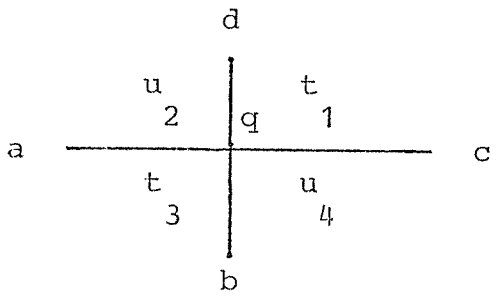
ALGORITHM CZRBA:

CZRBA(A,t,u,a,b,c,d,q)

Complex Zero System, Roots in

Bi-Quadrants and on the Axes

A is the input. t, u, a, b, c, d, and q are outputs. A is a non-zero square-free univariate Gaussian polynomial. t, u, a, b, c, d, and q are Fortran integers, the number of zeros of A in seven disjoint portions of the complex plane as indicated in the following diagram:

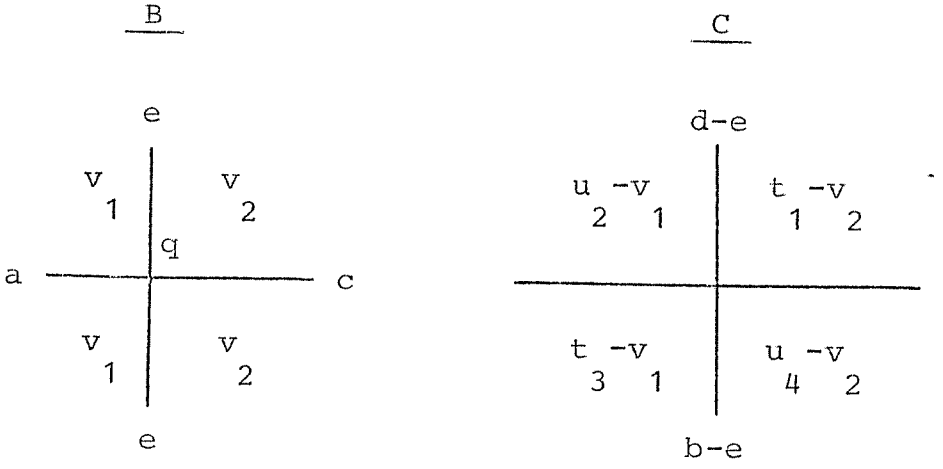


More precisely, $t = t_1 + t_3$ is the number of zeros in the first and third quadrants, $u = u_2 + u_4$ is the number in the second and fourth quadrants. a and c are

the numbers of negative and positive real zeros. b and d are the numbers of imaginary zeros below and above the real axis. q = 1 if A(0) = 0 and q = 0 otherwise.

Method:

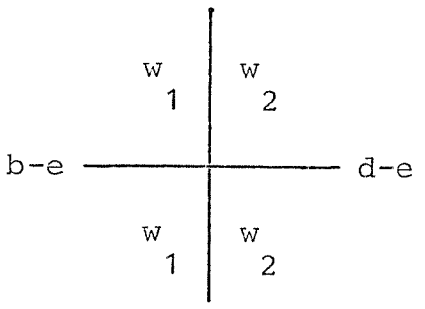
Let A = BC where B is the symmetric part of A (that is, the g.c.d. of its real and imaginary parts). Let the zeros of B and C be distributed as follows:



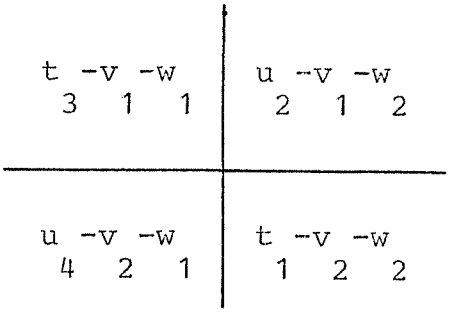
Application of PORD and PWRAZR to B yield q and $\overline{B}$. Application of CZNNPA to $\overline{B}$ yields a, c, and $v_1 + v_2 + e$. Let $D(z) = \overline{B}(iz)$. Then application of CZNNPA to the symmetric part of D yields e, from which $v_1 + v_2 = v$ can be computed.

Let E(z) = C(iz) and E = FG, where F is the symmetric part of E. The zeros of F and G are distributed as follows:

F



G



Application of CZNNPA to F yields b' = b - e, d' = d - e, and $w = w_1 + w_2$ from which b and d are computed.

Let H be the rootsquare of G. Application of CZABRA to H yields u - v - w and t - v - w, from which u and t can be computed since v and w are already known.

Description:

(1) [Compute q, a, c, e, and v.] GPSYMP(A,B,C); q ← PORD(B); $\overline{B}$ ← PWRAZR(B); erase B; CZNNPA($\overline{B}$,a,c,v'); $\overline{B}$ ← PFL($\overline{B}$,PFL(0,0)); D ← GPROT($\overline{B}$,1); erase $\overline{B}$; GPSYMP(D,D_1,D_2); erase D, D_2; CZNNPA(D_1,e,e',v''); erase D_1; v ← v' - e.

(2) [Compute w, b, and d.] E ← GPROT(C,1); erase C; GPSYMP(E,F,G); erase E; CZNNPA(F,b',d',w); erase F; b ← b' + e; d ← d' + e.

(3) [Compute t and u.] H ← GPRSQR(G); erase G; CZABRA(H,u',t'); erase H; t ← t' + v + w; u ← u' + v + w; return.

Computing Time: $\preceq \mu^{4+k}\{\mu L(\mu) + L(d)\}^2$, where $m = \deg(A)$, $\mu = m + 1$, $d = |A|_\infty$; $k = 0$ if modular methods are used or the p.r.s.'s are normal, $k = 1$ otherwise.

Proof: $t_{GPSYMP}(A) \preceq \mu^3 L(\mu d)^2 \sim \mu^3\{L(\mu) + L(d)\}^2$. $B|A$ so $L(|B|_\infty) \preceq \mu L(\mu) + L(d)$. $t_{PORD}(B), t_{PWRAZR}(B) \preceq \mu$. $|\overline{B}|_\infty = |B|_\infty$ so $t_{CZNNPA}(\overline{B}) \preceq \mu^{4+k}\{\mu L(\mu) + L(d)\}^2$ and $t_{GPROT}(\overline{B}, 1) \preceq \mu\{\mu L(\mu) + L(d)\}$. $|D|_\infty = |\overline{B}|_\infty$, so $t_{GPSYMP}(D) \preceq \mu^3\{\mu L(\mu) + L(d)\}^2$. $D_1(-iz)|D(-iz)$, $D(-iz) = \overline{B}(z)$, $\overline{B}(z)|B(z)$, and $B(z)|A(z)$, so $|D_1|_\infty = |D_1(-iz)|_\infty$ and $L(|D_1|_\infty) \preceq \mu L(\mu) + L(d)$. Hence $t_{CZNNPA}(D_1) \preceq \mu^{4+k}\{\mu L(\mu) + L(d)\}^2$. $C|A$ so $L(|C|_\infty) \preceq \mu L(\mu) + L(d)$ and $t_{GPROT}(C,1) \preceq \mu\{\mu L(\mu) + L(d)\}$. $|E|_\infty = |C|_\infty$ so $t_{GPSYMP}(E) \preceq \mu^3\{\mu L(\mu) + L(d)\}^2$. $F(z)|A(iz)$ so $F|A$, $L(|F|_\infty) \preceq \mu L(\mu) + L(d)$ and $t_{CZNNPA}(F) \preceq \mu^{4+k}\{\mu L(\mu) + L(d)\}$. $G|A$ so $L(|G|_\infty) \preceq \mu L(\mu) + L(d)$ and $t_{GPRSQR}(G) \preceq$

$\mu^2 L(\mu|G|_\infty) \leq \mu^2\{\mu L(\mu) + L(d)\}^2$. $L(|H|_\infty) \leq L(\mu) +$

$L(|G|_\infty) \leq \mu L(\mu) + L(d)$ and $t_{CZABRA}(H) \leq \mu^{4+k}\{\mu L(\mu) +$

$L(d)\}^2$. ∎

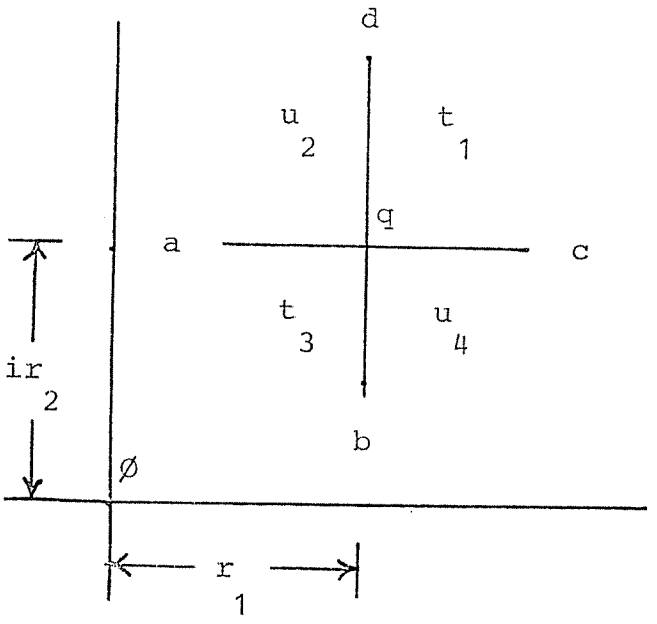
The preceding algorithm is made more versatile by allowing the coordinate system being considered to have its origin at any rational point in the complex plane. This is accomplished by the following algorithm, which applies the double translation mapping.

ALGORITHM CZRABA:

CZRABA(A,r_1,r_2,t,u,a,b,c,d,q)

Complex Zero System, Roots in Arbitrary Bi-Quadrants and on Axes

A, r_1, r_2 are inputs. t, u, a, b, c, d, and q are outputs. A is a non-zero square-free univariate Gaussian polynomial. r_1 and r_2 are rational numbers. t, u, a, b, c, d, and q are Fortran integers, the number of zeros of A in seven disjoint portions of the complex plane, as indicated in the following diagram:



More precisely, $t = t_1 + t_3$ is the number of zeros in the first and third quadrants of the translated coordinate system, $u = u_2 + u_4$ is the number in the second and fourth quadrants. a and c are the numbers of negative and positive real zeros. b and d are the numbers of imaginary zeros below and above the real axis. $q = 1$ if $A(r_1 + ir_2) = A^*(0) = 0$ and $q = 0$ otherwise, where $A^*(z) = A(z + r_1 + ir_2)$.

Description:

(1) [Translate.] A' ← GPDRTR(A,r_1,r_2).

(2) [Compute values.] CZRBA(A',t,u,a,b,c,d,q); erase A'; return.

Computing Time: $\leq \mu^{4+k}\{\mu L(\mu) + \mu\rho + L(d)\}^2$, where $m = \deg(A)$, $\mu = m + 1$, $\rho = L(r_1) + L(r_2)$, $d = |A|_\infty$, and $k = 0$ if modular methods are used or the p.r.s.'s are normal, $k = 1$ otherwise.

Proof: $t_{GPRDTR}(A, r_1, r_2) \leq \mu^3\rho^2 + \mu^2\rho L(d) \leq \mu\{\mu\rho + L(d)\}^2$. $L(|A'|_\infty) \leq \mu\rho + L(d)$ so $t_{CZRBA}(A') \leq \mu^{4+k}\{\mu L(\mu) + \mu\rho + L(d)\}^2$. ∎

If it is necessary to compute the number of roots in individual quadrants rather than bi-quadrants, the following algorithm can be used.

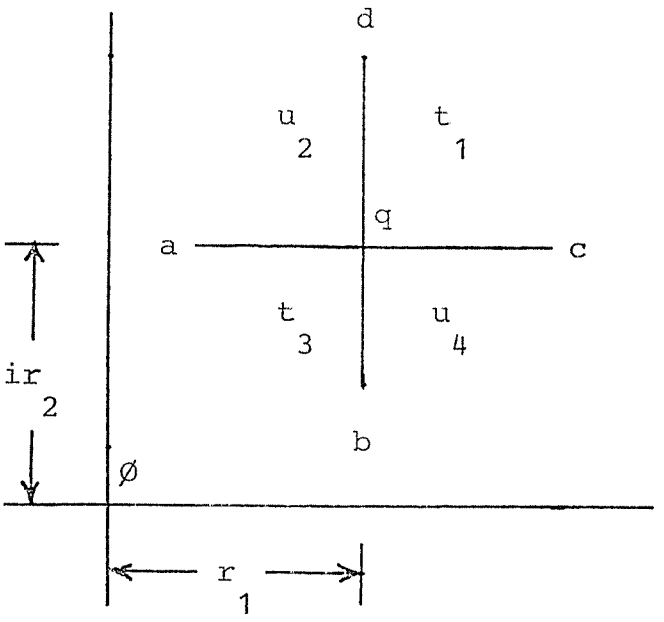
ALGORITHM GPNRQA:

GPNRQA($A, r_1, r_2, t_1, u_1, t_2, u_2, a, b, c, d, q$)

Gaussian Polynomial, Number of Roots in Quadrants and on Axes

A, r_1, and r_2 are inputs. t_1, u_1, t_2, u_2, a, b, c, d, and q are outputs. A is a non-zero square-free univariate Gaussian polynomial. r_1 and r_2 are rational numbers. t_1, u_1, t_2, u_2, a, b, c, d, q are Fortran integers, the number of zeros of A in nine disjoint portions of the complex plane, as indicated

in the following diagram:



More precisely, t_1, u_2, t_3, and u_4 are the number of zeros in the first, second, third, and fourth quadrants of the translated coordinate system. a and c are the number of negative and positive real zeros. b and d are the number of imaginary zeros below and above the real axis. $q = 1$ if $A(r_1 + ir_2) = A^*(0) = 0$ and $q = 0$ otherwise, where $A^*(z) = A(z + r_1 + ir_2)$.

Method:

Apply CZRBA to A* to obtain $t = t_1 + t_3$, $u = u_2 + u_4$, a, b, c, d, and . Apply CZNZHP to A* to obtain $k = t_1 + d + u_2$ and $h = a + q + c$. Apply CZNZHP to $\overline{A}(z) =$

A*(-iz) to obtain $m = t_1 + c + u_4$ and $\ell = d + q + b$.

Let $k' = k - d$ and $m' = m - c$. Then $t_1 = (k' + m' - u)/2$, $t_3 = t_1 - t_2$, $u_1 = k' - t_2$, $u_4 = m' - t_1$.

Description:

(1) $A^* \leftarrow$ GPDRTR(A, r_1, r_2); CZRBA$(A^*, t, u, a, b, c, d, q)$; CZNZHP$(A^*, h, k)$; $\overline{A} \leftarrow$ GPROT$(A^*, -1)$; erase A^*; CZNZHP$(\overline{A}, \ell, m)$; erase $\overline{A}$; $k' \leftarrow k - d$; $m' \leftarrow m - c$; $t_1 \leftarrow (k' + m' - u)/2$; $t_3 \leftarrow t_1 - t_2$; $u_1 \leftarrow k' - t_2$; $u_4 \leftarrow m' - t_1$; return.

Computing Time: $\preceq \mu^{4+k}\{\mu L(\mu) + \mu\rho + L(d)\}^2$, where $m = \deg(A)$, $\mu = m + 1$, $\rho = L(r_1) + L(r_2)$, $d = |A|_\infty$, and $k = 0$ if modular methods are used or the p.r.s.'s are normal, $k = 1$ otherwise.

Proof: $t_{\text{GPDRTR}}(A, r_1, r_2) \preceq \mu^3\rho^2 + \mu^2\rho L(d) \preceq \mu\{\mu\rho + L(d)\}^2$. $L(|A^*|_\infty) \preceq \mu\rho + L(d)$ so $t_{\text{CZRBA}}(A^*) \preceq \mu^{4+k}\{\mu L(\mu) + \mu\rho + L(d)\}^2$, $t_{\text{CZNZHP}}(A^*) \preceq \mu^{4+k}\{\mu L(\mu) + \mu\rho + L(d)\}^2$, and $t_{\text{GPROT}}(A^*) \preceq \mu^2\rho + \mu L(d)$. $|\overline{A}|_\infty = |A^*|_\infty$ so $t_{\text{CZNZHP}}(\overline{A}) \preceq \mu^{4+k}\{\mu L(\mu) + \mu\rho + L(d)\}^2$. ∎

On the other hand, considerable simplification can be achieved in certain instances by grouping axes and quadrants together into four areas, rather than the seven or nine areas which the previous algorithms work with. To aid in exposition, several definitions are now introduced.

An interval I is any set such that if a, b ε I and a < x < b, then x ε I.

The various kinds of intervals are (a,b), (a,b], [a,b), and [a,b], where a < b; {a}, a single point; (a,∞), [a,∞), ($-\infty$,a), ($-\infty$,a], and ($-\infty$,∞) = R.

A standard interval is an interval of one of the forms (a,b] where a < b, {a}, (a,∞), or ($-\infty$,a].

A rectangle S in the complex plane is a Cartesian product I $\times$ J, where I and J are non-empty intervals in the field R of real numbers.

A standard rectangle is a rectangle I $\times$ J, where I and J are standard intervals.

The standard quadrants are the standard rectangles (0,∞) $\times$ (0,∞), ($-\infty$,0] $\times$ (0,∞), ($-\infty$,0] $\times$ ($-\infty$,0], and (0,∞) $\times$ ($-\infty$,0].

The following algorithm, then, computes the number of roots in standard quadrants I and III, and standard quadrants II and IV.

ALGORITHM CZRSB:

CZRSB(A, $\bar{t}_{13}$, $\bar{t}_{24}$)

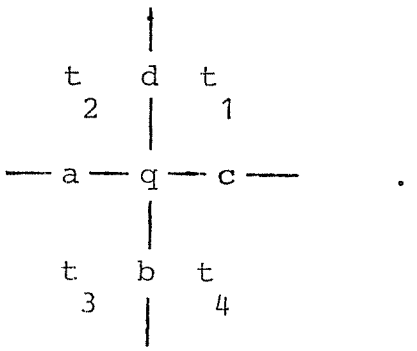
Complex Zero System, Roots in

Standard Bi-Quadrants

A is the input. $\bar{t}_{13}$ and $\bar{t}_{24}$ are outputs. A is a non-zero square-free univariate Gaussian polynomial. $\bar{t}_{13}$ and $\bar{t}_{24}$ are Fortran integers. $\bar{t}_{13}$ is the number of zeros of A in the first and third standard quadrants, $\bar{t}_{24}$ is the number of zeros of A in the second and fourth standard quadrants.

Method:

Let the number of zeros of A in the nine disjoint regions of the complex plane be as follows:



.

CZRBA returns $t_1 + t_3$, $t_2 + t_4$, a, b, c, d, q. Then $\bar{t}_{13} = (t_1 + t_3) + a + b + q$ and $\bar{t}_{24} = (t_2 + t_4) + c + d$.

Description:

(1) CZRBA(A,t,u,a,b,c,d,q); $\bar{t}_{13} \leftarrow t + a + q + b$;

$\bar{t}_{24} \leftarrow u + d + c$; return.

Computing Time: $\propto \mu^{4+k}\{\mu L(\mu) + L(d)\}^2$, where $m = \deg(A)$, $\mu = m + 1$, $d = |A|_\infty$, and $k = 0$ if modular methods are used or the p.r.s.'s are normal, $k = 1$ otherwise.

As with CZRBA, the previous algorithm can be extended to arbitrary coordinate systems using double translation.

ALGORITHM CZRASB:

CZRASB(A,r_1,r_2,$\bar{t}_{13}$,$\bar{t}_{24}$)

Complex Zero System, Roots in Arbitrary Standard Bi-Quadrants

A, r_1, and r_2 are the inputs. $\bar{t}_{13}$ and $\bar{t}_{24}$ are outputs. A is a non-zero, square-free univariate Gaussian polynomial. $\bar{t}_{13}$ and $\bar{t}_{24}$ are Fortran integers. Let $\bar{t}_i$ be the number of roots of A in the standard quadrants Q_i, $Q_1 = (r_1,\infty) \times (r_2,\infty)$, $Q_2 = (-\infty,r_1] \times (r_2,\infty)$, $Q_3 = (-\infty,r_1] \times (-\infty,r_2]$, $Q_4 = (r_1,\infty) \times (-\infty,r_2]$. Then $\bar{t}_{13} = \bar{t}_1 + \bar{t}_3$ and $\bar{t}_{24} = \bar{t}_2 + \bar{t}_4$.

Description:

(1) [Translate.] $A' \leftarrow GPDRTR(A, r_1, r_2)$.

(2) [Obtain values.] $CZRSB(A', \bar{t}_{13}, \bar{t}_{24})$; erase A';

return.

Computing Time: $\preceq \mu^{4+k}\{\mu L(\mu) + \mu\rho + L(d)\}^2$, where $m = \deg(A)$, $\mu = m + 1$, $\rho = L(r_1) + L(r_2)$, $d = |A|_\infty$, and $k = 0$ if modular methods are used or the p.r.s.'s are normal, $k = 1$ otherwise.

Proof: $t_{GPDRTR}(A, r_1, r_2) \preceq \mu^3\rho^2 + \mu^2\rho L(d) \preceq \mu\{\mu\rho + L(d)\}^2$. $L(|A'|_\infty) \preceq \mu\rho + L(d)$ so $t_{CZRSB}(A') \preceq \mu^{4+k}\{\mu L(\mu) + \mu\rho + L(d)\}^2$. ∎

The simplification of using standard quadrants is illustrated in the next algorithm, which computes the number of roots of a Gaussian polynomial in a standard rectangle.

ALGORITHM GPNRAR:

m=GPNRAR(A,B)

Gaussian Polynomial, Number of

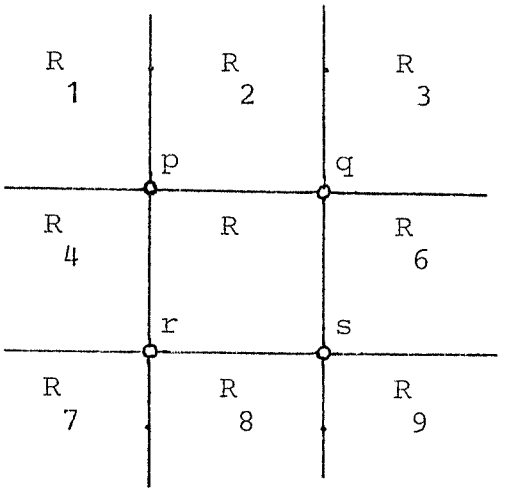
Roots in Arbitrary Rectangle

A is a non-zero square-free univariate Gaussian

polynomial. B is the list $((b_{11},b_{12}),(b_{21},b_{22}))$, representing a standard rectangle $R = (b_{11},b_{12}] \times (b_{21},b_{22}]$, where the b_{ij} are rational numbers. m is the number of roots of A in standard rectangle R.

Method:

Consider R and the surrounding area as nine standard rectangles:



Let t_i be the number of roots of A in the standard rectangle R_i (m roots in R). Consider applying CZRASB at points p, q, r, s:

$$\bar{p}_{13} = t_2 + t_3 + t_4 + t_7 ;$$

$$\bar{q}_{24} = t_1 + t_2 + t_6 + t_9 ;$$

$$\bar{r}_{13} = t_2 + t_3 + m + t_6 + t_7 ;$$

$$\bar{s}_{24} = t_1 + t_2 + t_4 + m + t_9 .$$

Then $m = (\bar{s}_{24} + \bar{r}_{13} - \bar{q}_{24} - \bar{p}_{13})/2$.

Description:

(1) [Obtain endlines of rectangle.] FIRST2(b_1,b_2,B); FIRST2(b_{11},b_{12},b_1); FIRST2(b_{21},b_{22},b_2).

(2) [Apply CZRASB at the vertices.] CZRASB(A,b_{11},b_{22},$\bar{p}_{13}$,$\bar{p}_{24}$); CZRASB(A,b_{12},b_{22},$\bar{q}_{13}$,$\bar{q}_{24}$); CZRASB(A,b_{11},b_{21},$\bar{r}_{13}$,$\bar{r}_{24}$); CZRASB(A,b_{12},b_{21},$\bar{s}_{13}$,$\bar{s}_{24}$).

(3) [Apply formula.] $m = (\bar{s}_{24} + \bar{r}_{13} - \bar{q}_{24} - \bar{p}_{13})/2$;

return.

Computing Time: $\propto \mu^{4+k}\{\mu L(\mu) + \mu\lambda + L(d)\}^2$, where $B = ((b_{11},b_{12}),(b_{21},b_{22}))$, $\lambda = L(b_{11}) + L(b_{12}) + L(b_{21}) + L(b_{22})$, $m = \deg(A)$, $\mu = m + 1$, $d = |A|_\infty$, and $k = 0$ if modular methods are used or the p.r.s.'s are normal, $k = 1$ otherwise.

Proof: The time to apply CZRASB is $\propto \sum_{i=1}^{2}\sum_{j=1}^{2}\{\mu^{4+k}\cdot [\mu L(\mu) + \mu\{L(b_{1,i}) + L(b_{2,j})\} + L(d)]^2\} \leq$

$$\mu^{4+k}\{\mu L(\mu) + \mu\sum_{i=1}^{2}\sum_{j=1}^{2}[L(b_{1,i}) + L(b_{2,j})] + L(d)\}^2$$

$$\sim \mu^{4+k}\{\mu L(\mu) + \mu\lambda + L(d)\}^2 . \blacksquare$$

The final algorithm in this section computes the number of roots of a Gaussian polynomial in a circle. It is not quite as straightforward as might be expected because, as discussed in Section 4.3, the half-plane to unit circle mapping excludes the point (1 + 0i).

This point can be handled, however, by noting the following about the mapping.

If $A(z) = \sum_{j=0}^{n} a_j z^j$ then $B(z) = (z + i)^n A((z - i)/(z + i)) = \sum_{j=0}^{n} a_j (z - i)^j (z + i)^{n-j} = \{\sum_{j=0}^{n} a_j\} z^n +$

$\{\sum_{j=0}^{n}\{(n - j)i - ji\}a_j\} z^{n-1} + \ldots = A(1)z^n + \{nA(1) - 2A'(1)\}iz^{n-1} + \ldots$

Hence deg(B) < deg(A) just in case A(1) = 0. More precisely, since A is square-free, A and A' have no common zeros; therefore if A(1) = 0 then $A'(1) \neq 0$ and deg(B) = deg(A) - 1 . So A has a zero at (1 + 0i) just in case deg(B) = deg(A) - 1.

These observations plus the mapping of Section 4.3 are used in the following algorithm for roots in a circle.

ALGORITHM GPNRIC:

GPNRIC(A,c,r,j,k)

Gaussian Polynomial, Number

of Roots in a Circle

A, c, and r are inputs; j and k are outputs. A is a non-zero square-free univariate Gaussian polynomial. c is a Gaussian rational number. r is a rational number, $r > 0$. j is the number of zeros of A on the circle with center at c and radius r, k is the number in the circle.

Method:

Let $A_1(z) = A(z + c)$, $A_2(z) = r^n A_1(rz)$ for $r = r_1/r_2$,

and $A_3(z) = (z + i)^n A((z - i)/(z + i))$ for $n = \deg(A)$.

Then apply CZNZHP(A_3,j,k). If $\deg(A_3) < \deg(A)$, then there is a root at $c + r$, so $j \leftarrow j + 1$.

Description:

(1) [Translate.] If $c = 0$, ($A_1 \leftarrow$ BORROW(A); go to (2));

FIRST2(c_1,c_2,c); $A_1 \leftarrow$ GPDRTR(A,c_1,c_2).

(2) [Rational homothetic operation.] $A_2 \leftarrow$ GPRHOM(A_1,r);

erase A_1.

(3) [Half-plane to circle mapping.] $A_3 \leftarrow$ GPHPTC(A_2);

erase A_2.

(4) [Compute number of zeros.] CZNZHP(A_3,j,k);

if GPDEG(A_3) < GPDEG(A), $j \leftarrow j + 1$; erase A_3; return.

Computing Time: $\preceq \mu^{4+k}\{\mu L(\mu\bar{r}\bar{c}) + L(a)\}^2$, where $a = |A|_\infty$, $m = \deg(A)$, $\mu = m + 1$, $c = c_{11}/c_{12} + ic_{21}/c_{22}$, $\bar{c} = \max_{1 \leq j,k \leq 2}\{|c_{j,k}|\}$, $r = r_1/r_2$, $\bar{r} = \max\{|r_1|, |r_2|\}$, $k = 0$ if modular methods are used or the p.r.s.'s are normal and $k = 1$ otherwise.

Proof: $t_{GPDRTR}(A,c_1,c_2) \preceq \mu^3 L(\bar{c})^2 + \mu^2 L(\bar{c})L(a)$.

$|A_1|_\infty \leq |\bar{c} + 1|^{8m} a$, so $t_{GPRHOM}(A_1,r) \preceq \mu^3 L(\bar{r})^2 + \mu^2 L(\bar{r})L(|\bar{c} + 1|^{8m} a) \preceq \mu^3 L(\bar{r})^2 + \mu^3 L(\bar{r})L(\bar{c}) + \mu^2 L(\bar{r})L(a)$. $|A_2|_\infty \leq |r + 1|^m |A_1|_\infty \leq (|\bar{r} + 1| \cdot |\bar{c} + 1|^{8m}) a$. Hence $t_{GPHPTC}(A_2) \preceq \mu^3 + \mu^2 L(\{|\bar{r} + 1| \cdot |\bar{c} + 1|^{8m}\} a) \preceq \mu^3 + \mu^3 L(\bar{r}) + \mu^3 L(\bar{c}) + \mu^3 L(a)$. $|A_3|_\infty \leq m2^m |A_2|_\infty \leq m(2|\bar{r} + 1| \cdot |\bar{c} + 1|^{8m}) a$. So $t_{CZNZHP}(A_3) \preceq$

$$\mu^{4+k}\{\mu L(\mu) + L(m\{2|\bar{r}+1|\cdot|\bar{c}+1|^{8m}\}a)\}^{2} \underline{\propto} \mu^{4+k}\{\mu L(\mu) + L(\mu) + \mu L(\bar{r}\bar{c}) + L(a)\}^{2} \underline{\propto} \mu^{4+k}\{\mu L(\mu\bar{r}\bar{c}) + L(a)\}^{2}. \blacksquare$$

CHAPTER 5: ROOT ISOLATION AND REFINEMENT

This chapter presents the two major algorithms of the thesis--Gaussian polynomial and integral polynomial root isolation and refinement.

The method used in these algorithms requires a bound on the moduli of all of the roots of a Gaussian polynomial. Hence the first algorithm computes such a bound.

ALGORITHM GPRBND:

B=GPRBND(A)

Gaussian Polynomial Root Bound

A is a non-zero univariate Gaussian polynomial. $B = b_1/b_2$ is a rational number, where b_1 is a power of two and $b_2 = 1$, such that if b is a root of A then $|b| < B$.

Method:

Using [KND69] let $B' = \max_{1 \le j \le n} \{2|a_{n-j}/a_n|^{1/j}\}$.

If $|a_i| \le |a_n|$ for $0 \le i < n$, then let $B' = 2$. Otherwise, compute B' as follows:

$$\log_2 B' = \max_{1 \le j \le n} \{\log_2 (2|a_{n-j}/a_n|^{1/j})\}$$

$$\le 1 + \max_{\substack{1 \le j \le n \\ |a_{n-j}| > |a_n|}} \{(1/j)\log_2 (|a_{n-j}/a_n|)\}$$

$$= 1 + \max_{\substack{1 \le j \le n \\ |a_{n-j}| > |a_n|}} \{(1/[2j])\log_2(|a_{n-j}|^2/|a_n|^2)\}$$

$$= 1 + \max_{\substack{0 \le k < n \\ |a_k| > |a_n|}} \{(1/[2(n-k)])\log_2(|a_k|^2/|a_n|^2)\}$$

$$\le 1 + \max_{\substack{0 \le k < n \\ |a_k| > |a_n|}} \{(1/[2(n-k)]) \cdot (\lceil \log_2 |a_k|^2 \rceil - \lfloor \log_2 |a_n|^2 \rfloor)\}$$

$$\le 1 + \max_{\substack{0 \le k < n \\ |a_k| > |a_n|}} \{\lceil ((\lceil \log_2 |a_k|^2 \rceil - \lfloor \log_2 |a_n|^2 \rfloor)/[2(n-k)]) \rceil\}$$

$$= 1 + h .$$

Then $b_1 = 2^{h+1}$

Description:

(1)[Initialize.] a' ← GPLDCF(A); b ← GPMSQR(a');

erase a'; A_1 ← GPRED(A); h ← 0; n ← GPDEG(A);

if $A_1 = 0$, go to (5); ELPOF2(b,ℓ,t).

(2) [Compute next term.] $a' \leftarrow \text{GPLDCF}(A_1)$; $A_2 \leftarrow \text{GPRED}(A_1)$; $b' \leftarrow \text{GPMSQR}(a')$; if $\text{ICOMP}(b',b) \leq 0$, go to (4); $\text{ELPOF2}(b',t,\ell')$; $t \leftarrow 2(n - \text{GPDEG}(A_1))$; $h' \leftarrow (\ell' - \ell + t - 1)/t$.

(3) [Compare.] If $h' > h$, $h \leftarrow h'$.

(4) [Increment and check.] Erase b', a', A_1; $A_1 \leftarrow A_2$; if $A_1 \neq 0$, go to (2).

(5) [Finish.] $t \leftarrow \text{PFA}(2,0)$; $a \leftarrow \text{PPOWER}(t,h+1)$; $B \leftarrow \text{RPOLY}(a)$; erase t, b, a; return.

Computing Time: $\propto \mu L(d)^2$, where $m = \deg(A)$, $\mu = m + 1$, and $d = |A|_\infty$.

Proof:

Step	n_i	t_i
1	1	$\propto L(d)^2$
2	$\leq \mu$	$\propto L(d)^2$
3	$\leq \mu$	~ 1
4	$\leq \mu$	~ 1
5	1	$\propto L(d)^2$

$L(a') \leq L(d)$, so $t_{\text{GPMSQR}}(a') \propto L(d)^2$. $L(b)$, $L(b') \leq$

$2L(d)$, so $t_{ELPOF2}(b), t_{ELPOF2}(b') \underline{\propto} L(d)^2$. $t_{PPOWER}(t, h+1) \underline{\propto} L(t)^2 (h+1)^2 \underline{\propto} L(2)[L(d)+1]^2 \underline{\propto} L(d)^2$. ∎

The next algorithm, GPRIR, isolates all of the roots of a Gaussian polynomial into disjoint squares and, if so directed, refines these squares to a specified width.

Given a Gaussian polynomial A, one can use GPRBND to compute a bound on the moduli of the roots of A, and thus specify a square in the complex plane which contains all of the roots of A.

Since an algorithm for computing the number of roots of a polynomial in a rectangle was previously explained, the following straightforward approach to isolation and refinement is suggested.

Initialize list L_1 to the square containing all roots of A and list L_2 to zero. Now enter a loop. Take the next square S off list L_1. If S contains no roots, discard it. If S contains one root and its width is less than the specified parameter ε, or $\varepsilon = 0$, prefix S to list L_2. Otherwise, split S into four subsquares and prefix these subsquares to list L_1. The loop is repeated until list L_1 is empty. List L_2 is then the desired output.

The problem with this approach is the time required to compute the number of roots in a rectangle, plus the large amount of information which is repeatedly re-computed. Although many improvements can be made, a different approach proved more successful.

Recalling the discussion of intervals in Section 4.4, define a standard horizontal strip in the complex plane as $(-\infty,\infty) \times (a,b]$ and a standard vertical strip as $(a,b] \times (-\infty,\infty)$ for a,b real numbers. (Note that, if H and V are standard horizontal and vertical strips, then $H \cap V$ is a standard rectangle.)

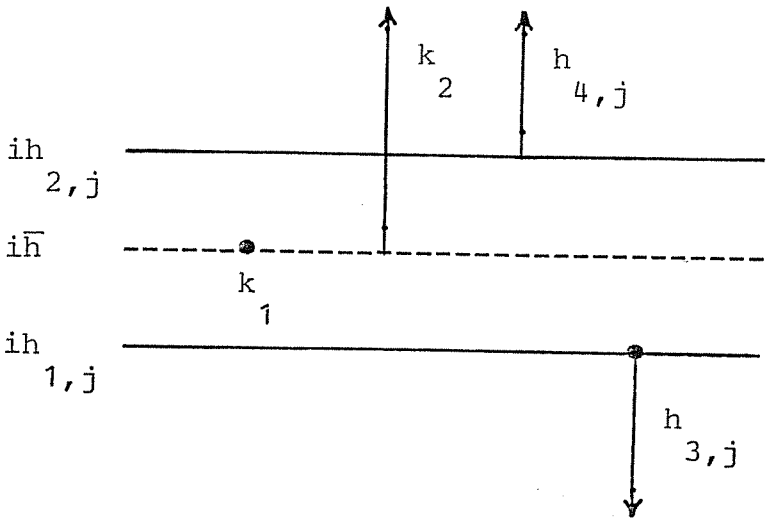
Some of the data structures in the algorithm are then as follows. $H_1 = (h_{1,1}, \ldots, h_{1,r})$, $H_2 = (h_{2,1}, \ldots, h_{2,r})$, $V_1 = (v_{1,1}, \ldots, v_{1,t})$, and $V_2 = (v_{2,1}, \ldots, v_{2,t})$ are lists in which $(h_{1,j}, h_{2,j})$ represents a standard horizontal strip $H_j = (-\infty,\infty) \times (h_{1,j}, h_{2,j}]$ and $(v_{1,k}, v_{2,k})$ represents a standard vertical strip $V_k = (v_{1,k}, v_{2,k}) \times (-\infty,\infty)$. $M = ((m_{1,1}, \ldots, m_{1,t}), \ldots, (m_{r,1}, \ldots, m_{r,t}))$ is then a list in which $m_{j,k}$ is the number of roots of A in standard rectangle $H_j \cap V_k$.

Let b be a bound on the moduli of the roots of A.

Initially, then, one has $H_1 = (-b,b)$ and $V_1 = (-b,b)$, with $M = ((\deg(A)))$. Now, rather than directly splitting squares into four subsquares, one proceeds by splitting the horizontal and vertical strips into two substrips each. (Which, of course, accomplishes the same thing.)

In order to perform this as efficiently as possible, the algorithm saves some additional information. $H_3 = (h_{3,1}, \ldots, h_{3,r})$ and $H_4 = (h_{4,1}, \ldots, h_{4,r})$ are lists in which $h_{3,j}$ is the number of roots of A below strip H_j and $h_{4,j}$ is the number of roots of A above strip H_j. Similarly, V_3 and V_4 store the number of roots of A left of and right of the corresponding vertical strips.

Hence, suppose one wants to split horizontal strip H_j, $(h_{1,j}, h_{2,j})$. (An exactly analogous process is used for vertical strips.) Let $\bar{h} = (h_{1,j} + h_{2,j})/2$ and apply the algorithm for arbitrary half-planes at $\bar{h}$. Suppose the result is k_1 roots on line $z = i\bar{h}$ and k_2 roots above the line. Then the following information is available.



(1)

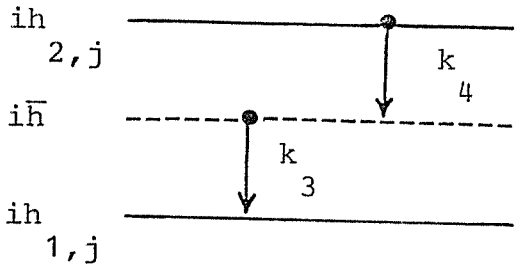
k_1 roots on $z = i\bar{h}$;

k_2 roots above $z = i\bar{h}$;

$h_{3,j}$ roots below and on $z = ih_{1,j}$;

$h_{4,j}$ roots above $z = ih_{2,j}$.

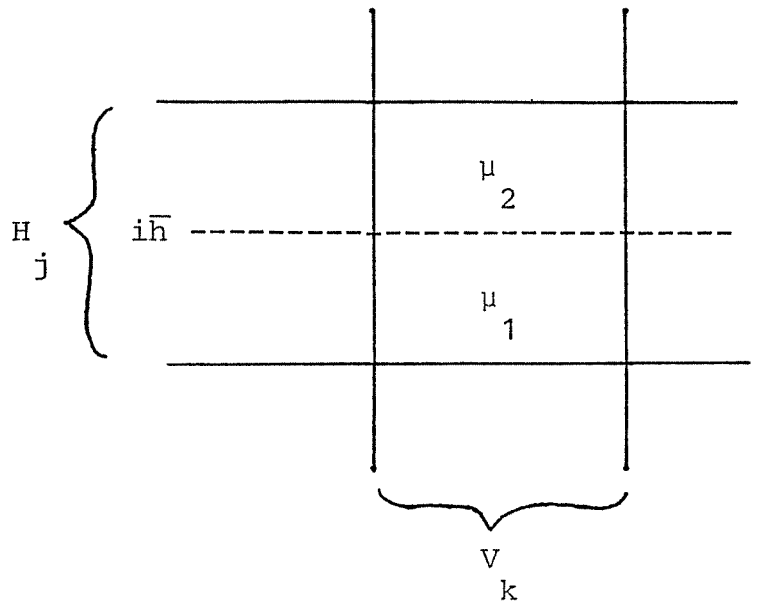
What is desired is the number of roots in strip $(h_{1,j},\bar{h})$ and $(\bar{h},h_{2,j})$, say k_3 and k_4 .



(2)

Reference to the two diagrams shows that $k_3 = \deg(A) - k_2 - h_{3,j}$ and $k_4 = k_2 - h_{4,j}$.

Suppose $k_3 = 0$. Then the only operation required is to replace $h_{1,j}$ by $\bar{h}$. Since $k_3 = 0$, $h_{3,j}$ remains the same, as does $h_{4,j}$. Consider the square $H_j \cap V_k$ and let $\mu = \mu_1 + \mu_2$ be the number of roots of A in this square.



(3)

Since there are no roots in the lower sub-strip, $\mu_1 = 0$ for all V_k. Hence $\mu_2 = \mu$ and matrix M remains the same.

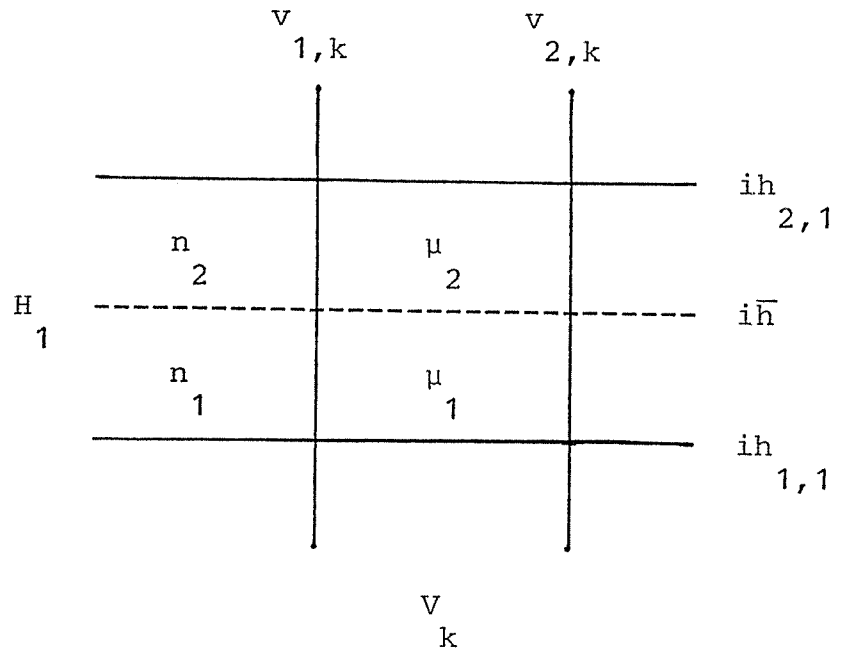
An analogous situation holds for $k_4 = 0$; in this case $h_{2,j}$ is altered to $\bar{h}$.

Now suppose $k_3, k_4 \neq 0$. Two strips must be set up, say $(h'_{1,j}, h'_{2,j})$ and $(h'_{1,j+1}, h'_{2,j+1})$ where $h'_{1,j} = h_{1,j}$, $h'_{1,j} = \bar{h}$,

$h'_{1,j+1} = \bar{h}$, and $h'_{2,j+1} = h_{2,j}$. Computing the other h entries is easy; reference to diagrams (1) and (2) shows that $h'_{3,j} = h_{3,j}$, $h'_{4,j} = h_{4,j} + k_4$, $h'_{3,j+1} = h_{3,j} + k_3$, and $h'_{4,j+1} = h_{4,j}$.

There is also an easy way of obtaining the new M entries--apply the algorithm for rectangles. However, this re-introduces the time problem created by that algorithm. A much faster alternative is doing a systematic scan and using the algorithm for standard bi-quadrants.

Suppose the first strip split is the lowest one. Then there are no roots below this strip. Consider the first non-zero entry $\mu = \mu_1 + \mu_2$ in the first row of M.

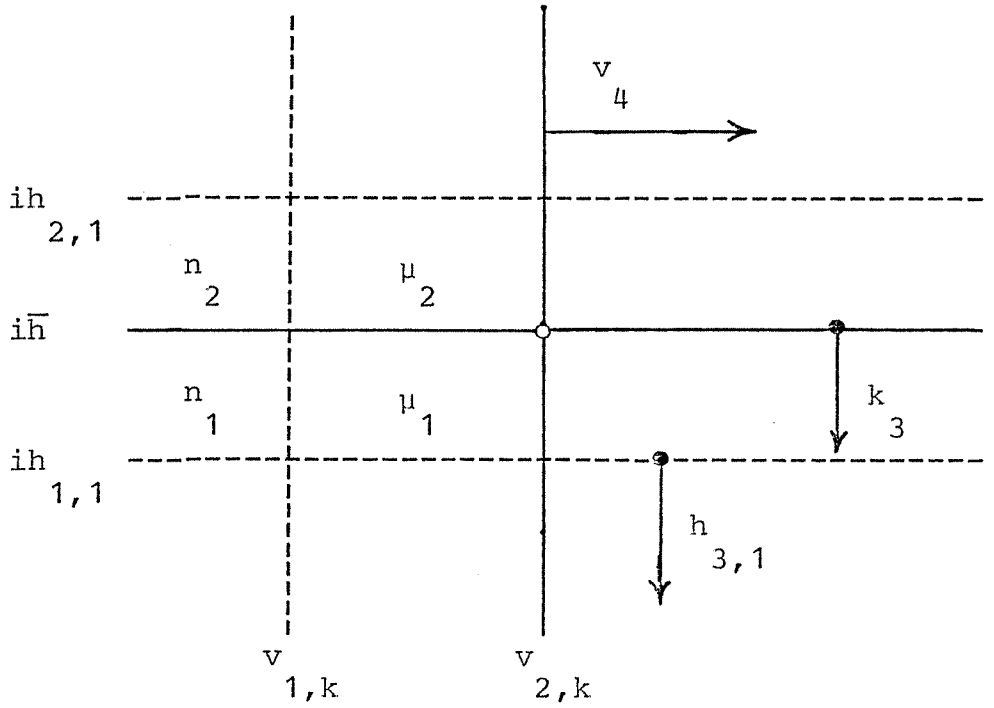


(4)

n_1 and n_2 are the number of roots of A in $(-\infty, v_{1,k}] \times$

$(h_{1,1}, \bar{h}]$ and $(-\infty, v_{1,k}] \times (\bar{h}, h_{2,1}]$, respectively. $n_1 =$ $n_2 = 0$ here because μ was the first non-zero entry.

Applying the algorithm for standard bi-quadrants at point $v_{2,k} + i\bar{h}$, one obtains q_{13} and q_{24}, the number of roots of A in standard bi-quadrants I + III and II + IV, respectively (for the coordinate axes centered at $v_{2,k} + i\bar{h}$). Combining this with other known values, one has the following situation:



(5)

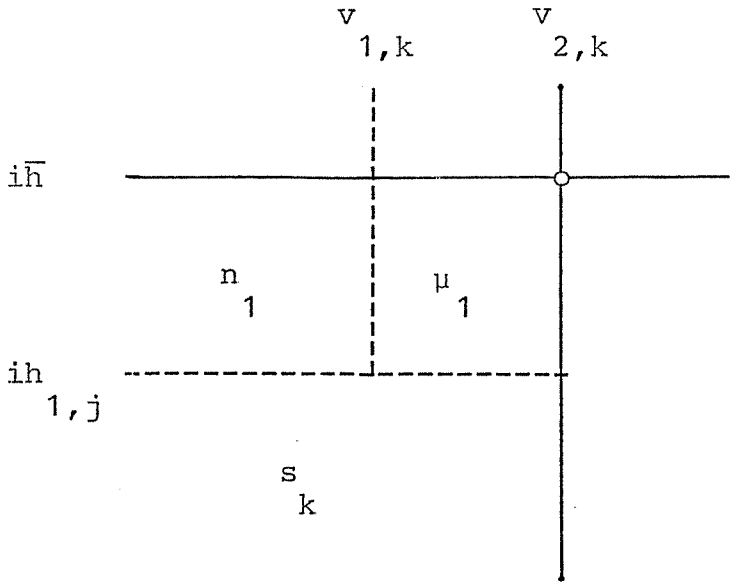
Then q_3, the number of roots in standard quadrant III, is $(\deg(A) - v_4 + h_{3,1} + k_3 - q_{24})/2$. Since there are no

roots below $h_{1,1}$ and $n_1 = 0$, then $\mu_1 = q_3$ and $\mu_2 = \mu - \mu_1$.

Now add μ_1 to n_1 and μ_2 to n_2. At the next rectangle in this horizontal strip, then, the same process can be used except $\mu_1 = q_3 - n_1$.

This gives a method for the entire lowest strip. On subsequent strips, however, one must account for roots in quadrant III below the strip. This can be done by adding one additional step to the previous discussion. Before beginning the split of horizontal strips, set up a list $S = (s_1, \ldots, s_t)$, with each entry initialized to zero. When the j-th horizontal strip is being processed, s_k will be the number of roots of A in $(-\infty, v_{2,k}] \times (-\infty, h_{1,j}]$.

Initially, all entries in S are zero because there are no roots below $h_{1,1}$. As each rectangle in a horizontal strip is processed, s_k is updated by $s_k \leftarrow s_k + n_1 + n_2 + \mu$. [See diagram (5).] When the processing of the next horizontal strip again reaches vertical strip V_k, the situation will be as follows.



(6)

Hence $\mu_1 = q_3 - n_1 - s_k$.

The process just described for splitting horizontal strips is alternately applied, with obvious modifications, to the splitting of vertical strips. This method can be repeated until the roots are isolated, that is, each square contains no roots or one root. If a refinement parameter ε is specified, it can further be continued until each square has width $< \varepsilon$.

The following algorithm employs the method just described for Gaussian polynomial root isolation and refinement.

ALGORITHM GPRIR:

L=GPRIR(A,ε)

Gaussian Polynomial Root

Isolation and Refinement

A is a square-free univariate Gaussian polynomial of positive degree m. ε is a non-negative rational number. L is a list of m disjoint standard squares each containing exactly one zero of A. If ε > 0, then each square has width less than ε.

Description:

(1) [Initialize.] b ← GPRBND(A); w ← RSUM(b,b); b' ← RNEG(b); V_1 ← PFL(b',0); V_2 ← PFL(b,0); H_1 ← PFL(BORROW(b'),0); H_2 ← PFL(BORROW(b),0); V_3 ← PFA(0,0); V_4 ← PFA(0,0); H_3 ← PFA(0,0); H_4 ← PFA(0,0); h ← RNUM(1,2); n ← GPDEG(A); M ← PFL(PFA(n,0),0); A* ← GPROT(A,+1); I ← 0; go to (25).

(2) [Initialize for rescan.] S ← 0; for j ← 1, . . . , LENGTH(V_1), do: S ← PFA(0,S); H'_1 ← 0; H'_2 ← 0; H'_3 ← 0; H'_4 ← 0; M' ← 0.

(3) [Split strip.] DECAP(m,M); S' ← S; DECAP(h_1,H_1); DECAP(h_2,H_2); DECAP(h_3,H_3); DECAP(h_4,H_4); $\overline{h}$ ← RNAVER(h_1,h_2); CZNZVH(A*,$\overline{h}$,k_1,k_2); k_3 ← n − k_2 − h;

$k_4 \leftarrow k_2 - h_4$; if $k_3 = 0$, go to (4); if $k_4 = 0$, go to (5); go to (6).

(4) [No roots in lower strip.] $M' \leftarrow PFL(m,M')$; erase h_1; $H' \leftarrow PFL(\bar{h}_1,H')$; $H' \leftarrow PFL(h_1,H')$; $H' \leftarrow PFA(h_2,H')$; $H' \leftarrow PFA(h_3,H')$; $\ell \leftarrow 0$; go to (10).

(5) [No roots in upper strip.] $M' \leftarrow PFL(m,M')$; erase h_2; $H' \leftarrow PFL(h_1,H')$; $H' \leftarrow PFL(\bar{h}_2,H')$; $H' \leftarrow PFA(h_3,H')$; $H' \leftarrow PFA(h_4,H')$; $\ell \leftarrow 0$; go to (10).

(6) [Zeros in both strips.] $H' \leftarrow PFL(\bar{h}_1,PFL(h_1,H'))$; $H' \leftarrow PFL(h_2,PFL(BORROW(\bar{h}_2),H'))$; $H' \leftarrow PFA(h_3 + k_3,PFA(h_3,H'))$; $H' \leftarrow PFA(h_4,PFA(h_4 + k_4,H'))$; $m' \leftarrow 0$; $m' \leftarrow 0$; $V' \leftarrow V_1$; $V' \leftarrow V_1$; $V' \leftarrow V_2$; $V' \leftarrow V_2$; $V' \leftarrow V_3$; $V' \leftarrow V_3$; $V' \leftarrow V_4$; $V' \leftarrow V_4$; $n_1 \leftarrow 0$; $n_2 \leftarrow 0$.

(7) [Determine number in lower section of new strip.]

DECAP(μ,m); ADV(v_1,V'); ADV(v_1,V'); ADV(v_2,V'); ADV(v_2,V'); ADV(v_3,V'); ADV(v_3,V'); ADV(v_4,V'); ADV(v_4,V'); $s \leftarrow FIRST(S')$; if $\mu_4 = 0$, ($\mu_1 \leftarrow 0$; go to (8)); if $n_1 = k_3$, ($\mu_1 \leftarrow 0$; go to (8)); if $n_2 = k_4$, ($\mu_1 \leftarrow \mu_1$; go to (8)); CZRASB(A,v_2,$\bar{h}$,q_{13},q_{24}); $\ell \leftarrow n_4 - v_3$; $y \leftarrow h_3 + k_3$; $q_{24} \leftarrow (\ell + y - q_1)/2$; $\mu_3 \leftarrow q_{24} - n_1 - s$.

(8) [Determine number in upper section.] $\mu_2 \leftarrow \mu - \mu_1$;

$n_1 \leftarrow n_1 + \mu_1$; $n_2 \leftarrow n_2 + \mu_2$; ALTER($s + n_1 + n_2$,S');

S' $\leftarrow$ TAIL(S'); $m'_1 \leftarrow$ PFA(μ_1,m'); $m'_2 \leftarrow$ PFA(μ_2,m');

if $m \neq 0$, go to (7).

(9) [Prefix pair and check for end.] M' $\leftarrow$ PFL(INV(m'_2),

PFL(INV(m'_1),M')); if $M \neq 0$, go to (3); go to (11).

(10) [Update S and check for end.] ADV(m_1,m); $\ell \leftarrow \ell +$

m_1; ALTER(FIRST(S') + ℓ,S'); S' $\leftarrow$ TAIL(S'); if S' $\neq 0$,

go to (10); if $M \neq 0$, go to (3).

(11) [Invert lists.] M' $\leftarrow$ INV(M'); $H_1 \leftarrow$ INV(H'_1); $H_2 \leftarrow$

INV(H'_2); $H_3 \leftarrow$ INV(H'_3); $H_4 \leftarrow$ INV(H'_4).

(12) [Prepare matrices for rescan.] M $\leftarrow$ CZTRAN(M');

erase M', S.

(13) [Initialize for rescan.] S $\leftarrow$ 0; for j $\leftarrow$ 1, . . . ,

LENGTH(H_1), do: S $\leftarrow$ PFA(0,S); $V'_1 \leftarrow 0$; $V'_2 \leftarrow 0$; $V'_3 \leftarrow 0$;

$V'_4 \leftarrow 0$; M' $\leftarrow$ 0.

(14) [Split strip.] DECAP(m,M); S' $\leftarrow$ S; DECAP(v_1,V_1);

DECAP(v_2,V_2); DECAP(v_3,V_3); DECAP(v_4,V_4); $\bar{v} \leftarrow$

RNAVER(v_1,v_2); CZNZVH(A,$\bar{v}$,k_1,k_2); $k_3 \leftarrow n - k_1 - v_2$;

$k_4 \leftarrow k_2 - v_4$; if $k_3 = 0$, go to (15); if $k_4 = 0$, go to (16); go to (17).

(15) [No roots in left strip.] $M' \leftarrow PFL(m,M')$; erase v_1; $V' \leftarrow PFL(\bar{v},V')$; $V' \leftarrow PFL(v_1,V')$; $V' \leftarrow PFA(v_2,V')$; $V' \leftarrow PFA(v_2,V')$; $V' \leftarrow PFA(v_3,V')$; $V' \leftarrow PFA(v_3,V')$; $V' \leftarrow PFA(v_4,V')$; $V' \leftarrow PFA(v_4,V')$; $\ell \leftarrow 0$; go to (21).

(16) [No roots in right strip.] $M' \leftarrow PFL(m,M')$; erase v_2; $V' \leftarrow PFL(v_1,V')$; $V' \leftarrow PFL(\bar{v},V')$; $V' \leftarrow PFL(\bar{v},V')$; $V' \leftarrow PFA(v_2,V')$; $V' \leftarrow PFA(v_3,V')$; $V' \leftarrow PFA(v_3,V')$; $V' \leftarrow PFA(v_4,V')$; $V' \leftarrow PFA(v_4,V')$; $\ell \leftarrow 0$; go to (21).

(17) [Zeros in both strips.] $V' \leftarrow PFL(\bar{v},PFL(v_1,V'))$; $V' \leftarrow PFL(v_1,PFL(BORROW(\bar{v}),V'))$; $V' \leftarrow PFA(v_2 + k_3,V')$; $V' \leftarrow PFA(v_2 + k_3,V')$; $V' \leftarrow PFA(v_3,PFA(v_3,V'))$; $V' \leftarrow PFA(v_4,PFA(v_4,V'))$; $m' \leftarrow 0$; $m' \leftarrow 0$; $H' \leftarrow H_1$; $H' \leftarrow H_1$; $H' \leftarrow H_2$; $H' \leftarrow H_2$; $H' \leftarrow H_3$; $H' \leftarrow H_3$; $H' \leftarrow H_4$; $H' \leftarrow H_4$; $n_1 \leftarrow 0$; $n_2 \leftarrow 0$.

(18) [Determine number in left section of new strip.] DECAP(μ,m); ADV(h_1,H'); ADV(h_1,H'); ADV(h_2,H'); ADV(h_2,H'); ADV(h_3,H'); ADV(h_3,H'); ADV(h_4,H'); $s \leftarrow FIRST(S')$; if $\mu = 0$, ($\mu_1 \leftarrow 0$; go to (19)); if $n_1 = k_3$, ($\mu_1 \leftarrow 0$; go to (19)); if $n_2 = k_4$, ($\mu_1 \leftarrow \mu$; go to (19)); CZRASB($A,\bar{v},h_2,q_{13},q_{24}$); $\ell \leftarrow n_4 - h_4$;

$y \leftarrow v_3 + k_3$; $q_3 \leftarrow (\ell + y - q_{24})/2$; $\mu_1 \leftarrow q_3 - n_3 - s_1$.

(19) [Determine number in right section.] $\mu_2 \leftarrow \mu - \mu_1$; $n_1 \leftarrow n_1 + \mu_1$; $n_2 \leftarrow n_2 + \mu_2$; ALTER($s + n_1 + n_2$,S'); S' $\leftarrow$ TAIL(S'); $m'_1 \leftarrow$ PFA(μ_1,m'); $m'_2 \leftarrow$ PFA(μ_2,m'); if $m \neq 0$, go to (18).

(20) [Prefix pair and check for end.] M' $\leftarrow$ PFL(INV(m')$_2$, PFL(INV(m')$_1$,M')); if $M \neq 0$, go to (14); go to (22).

(21) [Update S and check for end.] ADV(m_1,m); $\ell \leftarrow \ell + m_1$; ALTER(FIRST(S') + ℓ,S'); S' $\leftarrow$ TAIL(S'); if S' $\neq 0$, go to (21); if $M \neq 0$, go to (14).

(22) [Invert lists.] M' $\leftarrow$ INV(M'); $V_1 \leftarrow$ INV(V'_1); $V_2 \leftarrow$ INV(V'_2); $V_3 \leftarrow$ INV(V'_3); $V_4 \leftarrow$ INV(V'_4).

(23) [Prepare matrices for rescan.] M $\leftarrow$ CZTRAN(M'); erase M', S.

(24) [Compute width.] T $\leftarrow$ RPROD(w,h); erase w; w $\leftarrow$ T.

(25) [Test for isolation.] If I = 1, go to (28); M' $\leftarrow$ M.

(26) [Obtain next row.] ADV(m,M').

(27) [Check for 0 and 1.] ADV(m_1,m); if $m_1 > 1$, go to (2); if $m \neq 0$, go to (27); if M' $\neq 0$, go to (26); I $\leftarrow$ 1.

(28) [Check refinement.] If $\varepsilon = 0$, go to (29); if RCOMP$(w,\varepsilon) \geq 0$, go to (2).

(29) [Erase intermediates and initialize.] $L \leftarrow 0$; erase w, h, A^*, H_3, H_4, V_3, V_4.

(30) [Construct output.] $V' \leftarrow V_1$; $V' \leftarrow V_2$; DECAP(m,M); DECAP(h_1,H_1); DECAP(h_2,H_2).

(31) [Scan row.] DECAP(m_1,m); ADV(v_1,V'); ADV(v_2,V'); if $m_1 = 0$, go to (33).

(32) [Construct pair.] $V \leftarrow$ PFL(BORROW(v_1),PFL(BORROW(v_2),0)); $H \leftarrow$ PFL(BORROW(h_1),PFL(BORROW(h_2),0)); $\ell \leftarrow$ PFL$(V$,PFL$(H,0))$; $L \leftarrow$ PFL(ℓ,L).

(33) [Check end of row.] If $m \neq 0$, go to (31); erase h_1, h_2.

(34) [Check end of matrix.] If $M \neq 0$, go to (30); erase V_1, V_2; return.

Computing Time: $\propto \mu^{5+k} L(bh)\{\mu L(\mu bh) + L(d)\}^2 + L(\varepsilon)L(bh)^2$, where $m = \deg(A)$, $\mu = m + 1$, $d = |A|_\infty$, b is the positive integer bound on the moduli of the roots of A computed by GPRBND, $\lambda = 1/b$ if $m = 1$ and $\lambda = \text{sep}(A)$ if $m > 1$, $\delta = \lambda$ if $\varepsilon = 0$ and $\delta = \min\{\lambda,\varepsilon\}$ if ε

> 0, and $h = \lceil 1/\delta \rceil$; $k = 0$ if modular methods are used or the p.r.s.'s are normal, $k = 1$ otherwise.

Proof: Let n be the number of executions of Step (2). Let w_i be the width of each square during the i-th execution of Step (2). Then $w_i = 2b/2^{i-1}$. If $n > 0$, then either $\varepsilon = 0$ and $\sqrt{2}w_n > \lambda$ or $\varepsilon > 0$ and $w_n \geq \varepsilon$. In either case, $\sqrt{2}w_n > \delta$, that is, $2\sqrt{2}b/2^{n-1} > \delta$. Hence $2^n < 4\sqrt{2}b/\delta < 6bh$ and $n \propto L(bh)$. If $r = r_1/r_2$ is the endpoint of any interval processed during the i-th execution of the main loop then $|r_1|, |r_2| \leq b2^i$, so $L(r) \propto L(bh)$. This gives the following chart:

Step	n_i	t_i
1	1	$\propto \mu L(d)^2$
2	n	$\propto \mu$
3	$\leq \mu n$	$\propto \mu^{4+k}\{\mu L(\mu bh) + L(d)\}^2$
4	$\leq \mu n$	~ 1
5	$\leq \mu n$	~ 1
6	$\leq \mu n$	~ 1

7		$\lesssim \mu^{4+k} \{\mu L(\mu bh) + L(d)\}^2$
8	$\leq \mu^2 n$	~ 1
9	$\leq \mu n$	$\lesssim \mu$
10	$\leq \mu^2 n$	~ 1
11	n	$\lesssim \mu$
12	n	$\lesssim \mu^2$
13	n	$\lesssim \mu$
14	$\leq \mu n$	$\lesssim \mu^{4+k} \{\mu L(\mu bh) + L(d)\}^2$
15	$\leq \mu n$	~ 1
16	$\leq \mu n$	~ 1
17	$\leq \mu n$	~ 1
18		$\lesssim \mu^{4+k} \{\mu L(\mu bh) + L(d)\}^2$
19	$\leq \mu^2 n$	~ 1
20	$\leq \mu n$	$\lesssim \mu$
21	$\leq \mu^2 n$	~ 1
22	n	$\lesssim \mu$
23	n	$\lesssim \mu^2$
24	n	$\lesssim L(bh)$
25	$n + 1$	~ 1
26	$\leq \mu (n + 1)$	~ 1

27	$\leq \mu^2 (n + 1)$	~ 1
28	$\leq n + 1$	$\propto L(\varepsilon)L(bh)$
29	1	~ 1
30	$\leq \mu$	~ 1
31	$\leq \mu^2$	~ 1
32	μ	~ 1
33	$\leq \mu^2$	~ 1
34	$\leq \mu$	~ 1

$t_{GPRBND}(A) \propto \mu L(d)^2$ and $b \leq 2d$ so $t_{RSUM}(b,b)$,

$t_{RNEG}(b) \propto L(d)$. $t_{GPROT}(A) \propto \mu L(d)$, so $t_1 \propto \mu L(d)^2$.

Note that $|A^*|_\infty = d$.

length$(V_1) \leq m$ so $t_2 \propto \mu$.

length$(M) \leq m$ so $n_3 \leq \mu n$. $L(h_1), L(h_2) \propto L(bh)$ so

$t_{RNAVER}(h_1, h_2) \propto L(bh)^2$. Similarly, for $\bar{h} = \bar{h}_1 / \bar{h}_2$,

$L(\bar{h}_1), L(\bar{h}_2) \propto L(bh)$ so $\mu L(\mu \bar{h}_1 \bar{h}_2) \propto \mu L(\mu bh)$ and

$$t_{CZNZVH}(A^*, \bar{h}) \propto \mu^{4+k} \{\mu L(\mu bh) + L(d)\}^2.$$

Steps (7) and (8) form an inner loop which can be executed $m^2 n$ times, since there can be m rows and maximum row length is m. However, CZRASB can be

applied at most mn times, since there are at most m non-zero entries in M. For each execution, $t_{CZRASB}(A, v_2, \overline{h}) \underline{\propto} \mu^{4+k}\{\mu L(\mu bh) + L(d)\}^2$ so T_7, the time for all executions of Step (7), is $\underline{\propto} \mu^{5+k} L(bh)\{\mu L(\mu bh) + L(d)\}^2$.

Comments for Steps (2) through (12) apply for (13) through (23) also.

In Step (24), h = 1/2, so $t_{RPROD}(w,h) \underline{\propto} L(bh)$.

In Step (28), $t_{RCOMP}(w,\varepsilon) \underline{\propto} L(w)L(\varepsilon) \underline{\propto} L(\varepsilon)L(bh)$. ∎

To give another idea of the computing time of GPRIR, the following corollary is presented.

Corollary 5.1: $t_{GPRIR}(A,\varepsilon) \underline{\propto} \mu^{7+k}\{\mu L(\mu d) + L(\varepsilon)\}^3$, where m = deg(A), $\mu = m + 1$, and $d = |A|_\infty$.

Proof: Using the notation of the algorithm, Collins [COG73b] has shown that $L(\lceil 1/\lambda \rceil) \underline{\propto} \mu L(\mu d)$ and hence $L(h) \underline{\propto} \mu L(\mu d) + L(\varepsilon)$. Noting that $L(b) \underline{\propto} L(d)$ gives the corollary. ∎

The algorithm for integral polynomial root isolation and refinement is essentially the same. However, it achieves a considerable gain in speed by using the fact that any roots of an integral polynomial which are not real must

occur as complex conjugate pairs. Therefore, only the real roots and roots below the real axis are isolated and refined. The resulting regions in the lower half-plane containing a root are then used to generate the corresponding regions in the upper half-plane containing the conjugate of that root.

Several other comments should be made about differences in the algorithms. Heindel's real zero system [HEL70a] is faster than this algorithm for real roots. Hence Step (1) checks if all roots of the polynomial are real; if so, it calls Heindel's routine ANALR.

The bounds used to form the beginning area are (-b,0) and (-b,b). Hence the region is not square. To be consistent, an initial split of vertical strips is made. Subsequent operations thus work with square areas.

Recall the n_1 and n_2 discussed with the preceding algorithm. They designate the number of roots in the upper and lower sub-strips which have already been located. The total number of roots in each of these substrips, k_3 and k_4, are also known. If at some point $n_1 = k_3$, then there can be no other roots in this substrip so all subsequent μ_1 must be zero. A corresponding situation holds for n_2, k_4, and μ_2. When vertical strips are being processed in the following

algorithm, an extension of this can be used. Each time roots are located in the lower half-plane, the locations of their conjugates are also specified. Hence, if there are n_1 roots in $(a,b] \times (-\infty,-c]$ there are also n_1 roots in $(a,b] \times [c,\infty)$. The number of roots yet to be located in $(a,b] \times (-c,c)$ is thus $k_3 - 2n_1$, so processing can be stopped when $2n_1 = k_3$. Again, a corresponding situation holds for n_2 and k_4. This fact is used in Step (18) of the algorithm.

ALGORITHM PRIR:

L=PRIR(A,ε)

Polynomial Root Isolation and Refinement

A is a square-free univariate integral polynomial of positive degree m. ε is a non-negative rational number. L is a list of m disjoint regions each containing exactly one zero of A. Each region R is represented by a list ((a,b),(c,d)), where a, b, c, and d are rational numbers, as follows:

R = (a,b) × (c,d) if a < b and c < d;

R = (a,b) × {c} if a < b and c = d;

R = {a} × (c,d) if a = b and c < d;

R = {a} × {c} if a = b and c = d.

If ε > 0, then each region has width w < ε.

Description:

(1) [Check for all real zeros and initialize.]

CZNRNA(A,k,ℓ); if $\ell = 0$, go to (43); $A_1 \leftarrow$

PFL(BORROW(A),PFL(0,0)); $b \leftarrow$ GPRBND(A_1); $w \leftarrow$ RSUM(b,b);

$b' \leftarrow$ RNEG(b); $V_1 \leftarrow$ PFL(b',0); $V_2 \leftarrow$ PFL(b,0); $H_1 \leftarrow$

PFL(BORROW(b'),0); $H_2 \leftarrow$ PFL(0,0); $V_3 \leftarrow$ PFA(0,0); $V_4 \leftarrow$

PFA(0,0); $H_3 \leftarrow$ PFA(0,0); $H_4 \leftarrow$ PFA(ℓ,0); $h \leftarrow$ RNUM(1,2);

$n \leftarrow$ GPDEG(A_1); $M \leftarrow$ PFL(PFA($n - \ell$,0),0); $A_2 \leftarrow$

GPROT(A_1,+1); $I \leftarrow 0$; go to (13).

(2) [Initialize for rescan.] $S \leftarrow 0$; for j $\leftarrow$ 1, . . . ,

LENGTH(V_1), do: $S \leftarrow$ PFA(0,S); $H'_1 \leftarrow 0$; $H'_2 \leftarrow 0$; $H'_3 \leftarrow 0$;

$H'_4 \leftarrow 0$; $M' \leftarrow 0$.

(3) [Split strip.] DECAP(m,M); $S' \leftarrow$ S; DECAP(h_1,H_1);

DECAP(h_2, H_2); DECAP(h_3,H_3); DECAP(h_4,H_4); $\bar{h} \leftarrow$

RNAVER(h_1,h_2); CZNZVH(A_2,$\bar{h}$,k_1,k_2); $k_3 \leftarrow n - k_2 - h_3$;

$k_4 \leftarrow k_2 - h_4$; if $k_3 = 0$, go to (4); if $k_4 = 0$,

go to (5); go to (6).

(4) [No roots in lower strip.] $M' \leftarrow$ PFL(m,M');

erase h_1; $H'_1 \leftarrow$ PFL($\bar{h}$,H'); $H'_2 \leftarrow$ PFL(h_2,H');

$H_3' \leftarrow PFA(h_3, H_3')$; $H_4' \leftarrow PFA(h_4, H_4')$; $\ell \leftarrow 0$; go to (10).

(5) [No roots in upper strip.] $M_1' \leftarrow PFL(m, M_1')$;

erase h_2; $H_1' \leftarrow PFL(h_1, H_1')$; $H_2' \leftarrow PFL(\bar{h}, H_2')$; $H_3' \leftarrow$

$PFA(h_3, H_3')$; $H_4' \leftarrow PFA(h_4, H_4')$; $\ell \leftarrow 0$; go to (10).

(6) [Zeros in both strips.] $H_1' \leftarrow PFL(\bar{h}, PFL(h_1, H_1'))$;

$H_2' \leftarrow PFL(h_2, PFL(BORROW(\bar{h}), H_2'))$; $H_3' \leftarrow PFA(h_3 + k_3, H_3')$;

$PFA(h_3, H_3'))$; $H_4' \leftarrow PFA(h_4, PFA(h_4 + k_4, H_4'))$; $m_1' \leftarrow 0$; m_2'

$\leftarrow 0$; $V_1' \leftarrow V_1$; $V_2' \leftarrow V_2$; $V_3' \leftarrow V_3$; $V_4' \leftarrow V_4$; $n_1 \leftarrow 0$;

$n_2 \leftarrow 0$.

(7) [Determine number in lower section of new strip.]

DECAP(μ, m); $ADV(v_1, V_1')$; $ADV(v_2, V_2')$; $ADV(v_3, V_3')$;

$ADV(v_4, V_4')$; $s \leftarrow FIRST(S')$; if $\mu = 0$, ($\mu_1 \leftarrow 0$;

go to (8)); if $n_1 = k_3$, ($\mu_1 \leftarrow 0$; go to (8));

if $n_2 = k_4$, ($\mu_1 \leftarrow \mu$; go to (8)); $CZRASB(A_1, v_2, \bar{h}, q_{13}, q_{24})$;

$\ell \leftarrow n_4 - v_3$; $y \leftarrow h_3 + k_3$; $q_{24} \leftarrow (\ell + y - q_{24})/2$;

$\mu_1 \leftarrow q_{13} - n_3 - s$.

(8) [Determine number in upper section.] $\mu_2 \leftarrow \mu_2 - \mu_1$;

$n_1 \leftarrow n_1 + \mu_1$; $n_2 \leftarrow n_2 + \mu_2$; $ALTER(s + n_1 + n_2, S')$;

S' ← TAIL(S'); m'_1 ← PFA(μ_1,m'); m'_2 ← PFA(μ_2,m');

if m ≠ 0, go to (7).

(9) [Prefix pair and check for end.] M' ← PFL(INV(m'), $_2$

PFL(INV(m'),M')); if M ≠ 0, go to (3); go to (11).
$_1$

(10) [Update S and check for end.] ADV(m_1,m); ℓ ← ℓ +

m_1; ALTER(FIRST(S') + ℓ,S'); S' ← TAIL(S'); if S' ≠ 0,

go to (10); if M ≠ 0, go to (3).

(11) [Invert lists.] M' ← INV(M'); H_1 ← INV(H'_1); H_2 ←

INV(H'_2); H_3 ← INV(H'_3); H_4 ← INV(H'_4).

(12) [Prepare matrices for rescan.] M ← CZTRAN(M');

erase M', S.

(13) [Initialize for rescan.] S ← 0; for j ← 1, . . . ,

LENGTH(H_1), do: S ← PFA(0,S); V'_1 ← 0; V'_2 ← 0; V'_3 ← 0;

V'_4 ← 0; M' ← 0.

(14) [Split strip.] DECAP(m,M); S' ← S; DECAP(v_1,V_1);

DECAP(v_2,V_2); DECAP(v_3,V_3); DECAP(v_4,V_4); $\bar{v}$ ←

RNAVER(v_1,v_2); CZNZVH(A_1,$\bar{v}$,k_1,k_2); k_3 ← n − k_2 − v_3;

k_4 ← k_2 − v_4; if k_3 = 0, go to (15); if k_4 = 0,

go to (16); go to (17).

(15) [No roots in left strip.] $M' \leftarrow PFL(m,M')$; erase v_1;

$V' \leftarrow PFL(\overline{v},V')$; $V' \leftarrow PFL(v_1,V')$; $V' \leftarrow PFL(v_2,V')$; $V' \leftarrow PFA(v_2,V')$; $V' \leftarrow PFA(v_3,V')$; $V' \leftarrow PFA(v_4,V')$; $\ell \leftarrow 0$; go to (21).

(16) [No roots in right strip.] $M' \leftarrow PFL(m,M')$;

erase v_2; $V' \leftarrow PFL(v_1,V')$; $V' \leftarrow PFL(\overline{v},V')$; $V' \leftarrow PFA(v_2,V')$; $V' \leftarrow PFA(v_3,V')$; $V' \leftarrow PFA(v_4,V')$; go to (21).

(17) [Zeros in both strips.] $V' \leftarrow PFL(\overline{v},PFL(v_1,V'))$;

$V' \leftarrow PFL(v_2,PFL(BORROW(\overline{v}),V'))$; $V' \leftarrow PFA(v_3 + k_3,PFA(v_3,V'))$; $V' \leftarrow PFA(v_4,PFA(v_4 + k_4,V'))$; $m' \leftarrow 0$; $m' \leftarrow 0$; $H' \leftarrow H_1$; $H' \leftarrow H_1$; $H' \leftarrow H_2$; $H' \leftarrow H_2$; $H' \leftarrow H_3$; $H' \leftarrow H_3$; $H' \leftarrow H_4$; $H' \leftarrow H_4$; $n \leftarrow 0$; $n \leftarrow 0$.

(18) [Determine number in left section of new strip.]

DECAP(μ,m); ADV(h_1,H'); ADV(h_2,H'); ADV(h_3,H'); ADV(h_4,H'); $s \leftarrow FIRST(S')$; if $\mu = 0$, ($\mu \leftarrow 0$; go to (19)); if $2n_1 = k_3$, ($\mu \leftarrow 0$; go to (19)); if $2n_2 = k_4$, ($\mu \leftarrow \mu$; go to (19)); CZRASB(A_1,$\overline{v}$,h_2,q_{13},q_{24}); $\ell \leftarrow n - h_4$; $y \leftarrow v_3 + k_3$; $q_{24} \leftarrow (\ell + y - q_{24})/2$; $\mu \leftarrow q_{24} - n_3 - s_1$.

(19) [Determine number in right section.] $\mu_2 \leftarrow \mu - \mu_1$;

$n_1 \leftarrow n_1 + \mu_1$; $n_2 \leftarrow n_2 + \mu_2$; ALTER($s + n_1 + n_2$,S');

S' $\leftarrow$ TAIL(S'); $m'_1 \leftarrow$ PFA(μ_1,m'); $m'_2 \leftarrow$ PFA(μ_2,m');

if $m \neq 0$, go to (18).

(20) [Prefix pair and check for end.] M' $\leftarrow$ PFL(INV(m'),
$_2$

PFL(INV(m'),M')); if $M \neq 0$, go to (14); go to (22).
$_1$

(21) [Update S and check for end.] ADV(m_1,m); $\ell \leftarrow \ell +$

m_1; ALTER(FIRST(S') + ℓ,S'); S' $\leftarrow$ TAIL(S'); if S' $\neq 0$,

go to (21); if $M \neq 0$, go to (14).

(22) [Invert lists.] M' $\leftarrow$ INV(M'); $V_1 \leftarrow$ INV(V'_1); $V_2 \leftarrow$

INV(V'_2); $V_3 \leftarrow$ INV(V'_3); $V_4 \leftarrow$ INV(V'_4).

(23) [Prepare matrices for rescan.] M $\leftarrow$ CZTRAN(M');

erase M', S.

(24) [Compute width.] T $\leftarrow$ RPROD(w,h); erase w; w $\leftarrow$ T.

(25) [Test for isolation.] If I = 1, go to (28); M' $\leftarrow$ M.

(26) [Obtain next row.] ADV(m,M').

(27) [Check for 0 and 1.] ADV(m_1,m); if $m_1 > 1$,

go to (2); if $m \neq 0$, go to (27); if M' $\neq 0$,

go to (26); I $\leftarrow$ 1.

(28) [Check refinement.] If $\varepsilon = 0$, go to (29);

if RCOMP(w,ε) ≥ 0, go to (2).

(29) [Erase intermediates; initialize.] L ← 0;

erase w, h, A_2, H_3, H_4, V_3, V_4.

(30) [Construct output.] V'_1 ← V_1; V'_2 ← V_2; DECAP(m,M);

DECAP(h_1,H_1); DECAP(h_2,H_2).

(31) [Scan row.] DECAP(m_1,m); ADV(v_1,V'); ADV(v_2,V');

if m_1 = 0, go to (41).

(32) [Obtain location of root.] GPNRER(A_1,v_1,v_2,h_1,h_2,

n_1,n_2,n_3); if n_1 = 1, go to (34); if n_2 = 1, go to (36);

if n_3 = 1, go to (38).

(33) [Root is within rectangle.] R_{11} ← BORROW(v_1); R_{12}

← BORROW(v_2); T_{11} ← BORROW(h_1); T_{12} ← BORROW(h_2); R_{21}

← BORROW(v_1); R_{22} ← BORROW(v_2); T_{21} ← RNEG(h_2); T_{22} ←

RNEG(h_1); go to (39).

(34) [Root is on upper boundary of rectangle.] R_{11} ←

BORROW(v_1); R_{12} ← BORROW(v_2); T_{11} ← BORROW(h_2); T_{12} ←

BORROW(h_2).

(35) [Check for real root.] If h_2 = 0, go to (40);

$R_{21} \leftarrow$ BORROW(v_1); $R_{22} \leftarrow$ BORROW(v_2); $T_{21} \leftarrow$ RNEG(h_2);

$T_{22} \leftarrow$ BORROW(T_{21}); go to (39).

(36) [Root is at upper right vertex.] $R_{11} \leftarrow$ BORROW(v_2);

$R_{12} \leftarrow$ BORROW(v_2); $T_{11} \leftarrow$ BORROW(h_2); $T_{12} \leftarrow$ BORROW(h_2).

(37) [Check for real root.] If $h_2 = 0$, go to (40);

$R_{21} \leftarrow$ BORROW(v_2); $R_{22} \leftarrow$ BORROW(v_2); $T_{21} \leftarrow$ RNEG(h_2);

$T_{22} \leftarrow$ BORROW(T_{21}); go to (39).

(38) [Root is on right boundary of rectangle.] $R_{11} \leftarrow$

BORROW(v_2); $R_{12} \leftarrow$ BORROW(v_2); $T_{11} \leftarrow$ BORROW(h_2); $T_{12} \leftarrow$

BORROW(h_2); $R_{21} \leftarrow$ BORROW(v_2); $R_{22} \leftarrow$ BORROW(v_2); $T_{21} \leftarrow$

RNEG(h_2); $T_{22} \leftarrow$ RNEG(h_1).

(39) [Assemble conjugate interval.] $R_2 \leftarrow$ PFL(R_{21},

PFL(R_{22},0)); $T_2 \leftarrow$ PFL(T_{21},PFL(T_{22},0)); $\ell_2 \leftarrow$ PFL(R_2,

PFL(T_2,0)); $L \leftarrow$ PFL(ℓ_2,L).

(40) [Assemble interval.] $R_1 \leftarrow$ PFL(R_{11},PFL(R_{12},0)); T_1

$\leftarrow$ PFL(T_{11},PFL(T_{12},0)); $\ell_1 \leftarrow$ PFL(R_1,PFL(T_1,0)); $L \leftarrow$

PFL(ℓ_1,L).

(41) [Check for end of row.] If $m \neq 0$, go to (31); erase h_1, h_2.

(42) [Check for end of matrix.] If $M \neq 0$, go to (30); erase V_1, V_2, A_1; return.

(43) [Obtain real zeros.] $\overline{L} \leftarrow$ ANALR(A,ε); $y \leftarrow$ PFL(0,PFL(0,0)); $L \leftarrow 0$.

(44) [Check endpoint.] DECAP($\overline{\ell},\overline{L}$); FIRST2($\overline{\ell}_1,\overline{\ell}_2,\overline{\ell}$); if RCOMP($\overline{\ell}_1,\overline{\ell}_2$) = 0, go to (45); if REVAL($A,\overline{\ell}_2$) = 0, (erase $\overline{\ell}_1$; ALTER(BORROW($\overline{\ell}_2$),$\overline{\ell}$)).

(45) [Convert to PRIR output form.] $\ell \leftarrow$ PFL($\overline{\ell}$, PFL(BORROW(y),0)); $L \leftarrow$ PFL(ℓ,L); if $\overline{L} \neq 0$, go to (44); erase y; return.

Computing Time: $\propto \mu^{5+k} L(bh)\{\mu L(\mu bh) + L(d)\}^2 + L(\varepsilon)L(bh)^2$, where $m = \deg(A)$, $\mu = m + 1$, $d = |A|_\infty$, b is the positive integer bound on the moduli of the roots of A computed by GPRBND, $\lambda = 1/b$ if $m = 1$ and $\lambda = \text{sep}(A)$ if $m > 1$, $\delta = \lambda$ if $\varepsilon = 0$ and $\delta = \min\{\lambda,\varepsilon\}$ if $\varepsilon > 0$, and $h = \lceil 1/\delta \rceil$; $k = 0$ if modular methods are used or the p.r.s.'s are normal, $k = 1$ otherwise.

Proof: Let n be the number of executions of Step (2). Let w_i be the width of each square during the i-th

execution of Step (2). Then $w_i = 2b/2^{i-1}$. If $n > 0$, then either $\varepsilon = 0$ and $\sqrt{2}w_n > \lambda$ or $\varepsilon > 0$ and $w_n \geq \varepsilon$. In either case, $\sqrt{2}w_n > \delta$, that is, $2\sqrt{2}b/2^{n-1} > \delta$. Hence $2^n < 4\sqrt{2}b/\delta < 6bh$ and $n \propto L(bh)$. If $r = r_1/r_2$ is the endpoint of any interval processed during the i-th execution of the main loop then $|r_1|, |r_2| \leq b2^i$, so $L(r) \propto L(bh)$. This gives the following chart:

Step	n_j	t_j
1	1	$\propto \mu L(d)^2$
2	$\leq n$	$\propto \mu$
3	$\leq \mu n$	$\propto \mu^{4+k}\{\mu L(\mu bh) + L(d)\}^2$
4	$\leq \mu n$	~ 1
5	$\leq \mu n$	~ 1
6	$\leq \mu n$	~ 1
7		$\propto \mu^{4+k}\{\mu L(\mu bh) + L(d)\}^2$
8	$\leq \mu^2 n$	~ 1
9	$\leq \mu n$	$\propto \mu$

10	$\leq \mu^2 n$	~ 1
11	n	$\preceq_\alpha \mu$
12	n	$\preceq_\alpha \mu^2$
13	n	$\preceq_\alpha \mu$
14	$\leq \mu n$	$\preceq_\alpha \mu^{4+k}\{\mu L(\mu bh) + L(d)\}^2$
15	$\leq \mu n$	~ 1
16	$\leq \mu n$	~ 1
17	$\leq \mu n$	~ 1
18		$\preceq_\alpha \mu^{4+k}\{\mu L(\mu bh) + L(d)\}^2$
19	$\leq \mu^2 n$	~ 1
20	$\leq \mu n$	$\preceq_\alpha \mu$
21	$\leq \mu^2 n$	~ 1
22	n	$\preceq_\alpha \mu$
23	n	$\preceq_\alpha \mu^2$
24	n	$\preceq_\alpha L(bh)$
25	$n + 1$	~ 1
26	$\leq \mu(n + 1)$	~ 1
27	$\leq \mu^2 (n + 1)$	~ 1
28	$\leq n + 1$	$\preceq_\alpha L(\varepsilon)L(bh)$
29	1	~ 1
30	$\leq \mu$	~ 1

31	$\leq \mu^2$	~ 1
32	$\leq \mu$	$\underset{\sim}{\propto} \mu^{4+k} \{\mu L(\mu bh) + L(d)\}^2$
33	$\leq \mu$	$\underset{\sim}{\propto} L(bh)$
34	$\leq \mu$	~ 1
35	$\leq \mu$	$\underset{\sim}{\propto} L(bh)$
36	$\leq \mu$	~ 1
37	$\leq \mu$	$\underset{\sim}{\propto} L(bh)$
38	$\leq \mu$	$\underset{\sim}{\propto} L(bh)$
39	$\leq \mu$	~ 1
40	$\leq \mu$	~ 1
41	$\leq \mu^2$	~ 1
42	$\leq \mu$	~ 1
43	1	$\underset{\sim}{\propto} \mu^5 L(\mu bh)^2 + \mu^4 L(bh)^3$
44	m	$\underset{\sim}{\propto} \mu^2 L(bh)^2 + \mu^2 L(\mu d) L(bh)$
45	m	~ 1

$t_{GPRBND}(A) \underset{\sim}{\propto} \mu L(d)^2$ and $b \leq 2d$ so $t_{RSUM}(b,b)$, $t_{RNEG}(b) \underset{\sim}{\propto} L(d)$. $t_{GPROT}(A_1) \underset{\sim}{\propto} \mu L(d)$, so $t_1 \underset{\sim}{\propto} \mu L(d)^2$.

Note that $|A_1|_\infty = |A_2|_\infty = d$.

length$(V_1) \leq m$ so $t_2 \underset{\sim}{\propto} \mu$.

length$(M) \underset{\sim}{\leq} m$ so $n_3 \leq \mu n$. $L(h_1), L(h_2) \underset{\sim}{\propto} L(bh)$ so

$t_{RNAVER}(h_1,h_2) \propto L(bh)^2$.

Similarly, for $\bar{h} = \bar{h}_1/\bar{h}_2$, $L(\bar{h}_1), L(\bar{h}_2) \propto L(bh)$ so

$\mu L(\mu\bar{h}_1\bar{h}_2) \propto \mu L(\mu) + \mu L(bh)$ and $t_{CZNZVH}(A^*,\bar{h}) \propto$

$\mu^{4+k}\{\mu L(\mu bh) + L(d)\}^2$.

Steps (7) and (8) form an inner loop which can be executed $m^2 n$ times, since there can be m rows and maximum row length is m. However, CZRASB can be applied at most mn times, since there are at most m non-zero entries in M. For each execution,

$t_{CZRASB}(A_1,v_2,\bar{h}) \propto \mu^{4+k}\{\mu L(\mu bh) + L(d)\}^2$ so T_7, the time for all executions of Step (7), is $\propto \mu^{5+k} L(bh) \cdot \{\mu L(\mu bh) + L(d)\}^2 + L(\varepsilon)L(bh)^2$.

Comments for Steps (2) through (12) apply for (13) through (23) also.

In Step (24), h = 1/2 so $t_{RPROD}(w,h) \propto L(bh)$.

In STEP (28), $t_{RCOMP}(w,\varepsilon) \propto L(w)L(\varepsilon) \propto L(\varepsilon)L(bh)$.

Step (32) is executed once for each non-zero entry in M and hence $t_{32} \leq \mu$.

Collins has re-analyzed the time for ANALR [COG72b] and t_{43} is derived from this.

In Step (44), $t_{RCOMP}(\overline{\ell}_1, \overline{\ell}_2) \underline{\propto} L(\overline{\ell}_1)L(\overline{\ell}_2) \underline{\propto}$

$L(bh)^2 \cdot t_{REVAL}(A, \overline{\ell}_2) \underline{\propto} \mu^2 L(\overline{\ell}_2)^2 + \mu^2 L(\mu d)L(\overline{\ell}_2) \underline{\propto}$

$\mu^2 L(bh)^2 + \mu^2 L(\mu d)L(bh)$. ∎

Corollary 5.1 obviously also holds for PRIR, since the computing time and root separation are the same.

CHAPTER 6: SYSTEM EXTENSIONS

6.1 Refining individual roots

The algorithms of Chapter 5 isolate the roots of a polynomial and then refine all of them simultaneously. In some instances, this is not desired. From knowledge about the polynomial (perhaps from a prior application of an algorithm in Chapter 5), one may have isolating rectangles for certain of the roots and now want to refine only these areas. A common example is the situation in which only the roots in the first quadrant are important. This section presents algorithms for refining such individual roots.

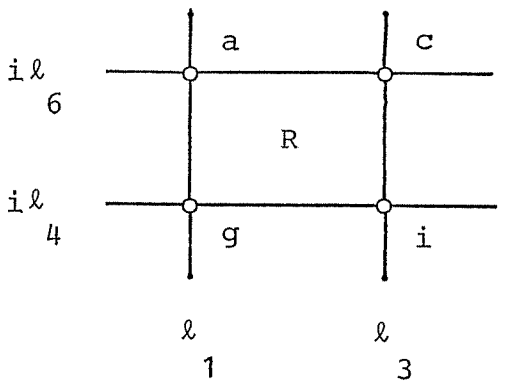
The first algorithm inputs a standard rectangle containing a single root and refines it once, producing a sub-rectangle containing the root with sides 1/2 the length of the input.

ALGORITHM CZREF:

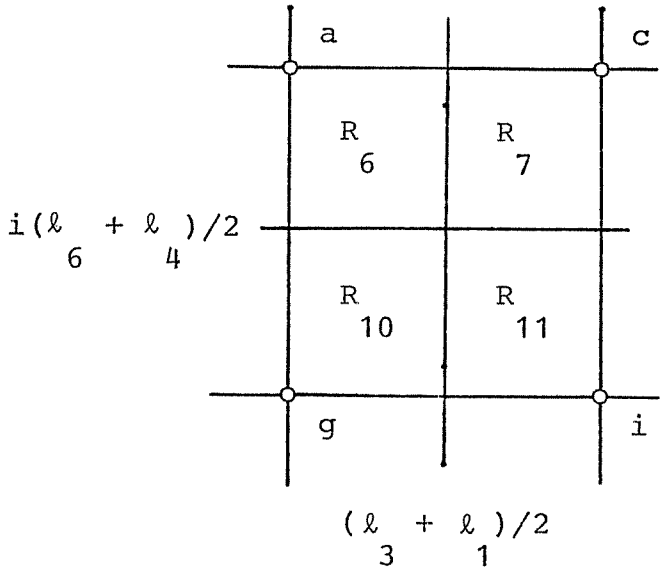
$\overline{L}$=CZREF(A,L)

Complex Zero Refinement

A is a non-zero square-free univariate Gaussian polynomial. L is the interval representation of a rectangle in the complex plane containing (in the standard rectangle sense) one zero of A. That is, L = $((\ell_1,\ell_3),(\ell_4,\ell_6))$, ℓ_i rational numbers, for the following rectangle R:



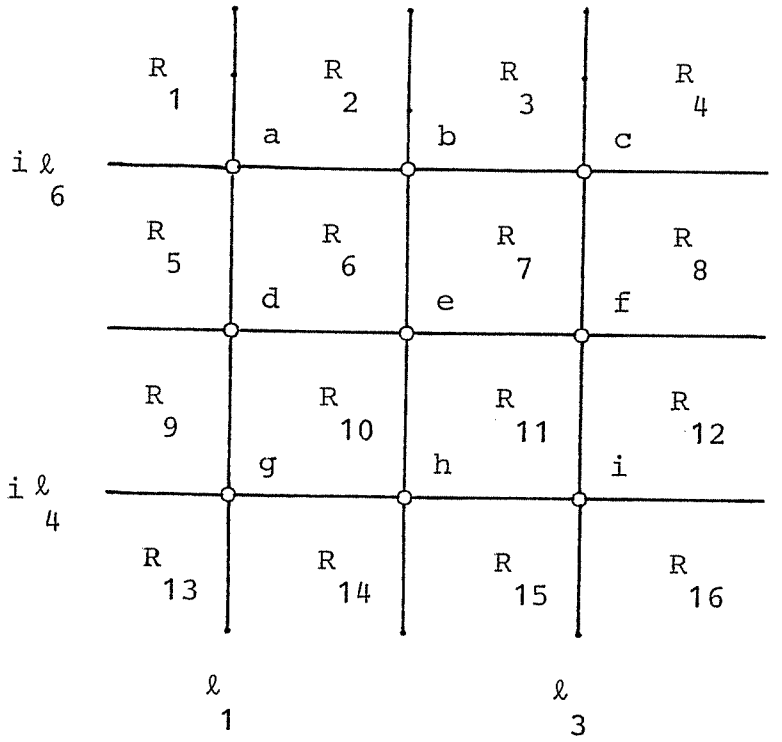
The zero lies in the interior of the rectangle, on line (a,c], or on line (i,c]. Let R be divided into the following four standard sub-rectangles:



Then $\overline{L}$ is the interval representation of the standard subrectangle R_j containing the zero.

Method:

Consider the surrounding standard rectangles as in the following diagram:



Applying CZRASB at points a, b, d, e gives the following, for t_j the number of zeros in R_j:

$$\bar{a} = t_{13} + t_2 + t_3 + t_4 + t_5 + t_9 + t_{13},$$

$$\bar{b} = t_{24} + t_1 + t_2 + t_7 + t_8 + t_{11} + t_{12} + t_{15} + t_{16},$$

$$\bar{d} = t_{13} + t_2 + t_3 + t_4 + t_6 + t_7 + t_8 + t_9 + t_{13},$$

$$\bar{e} = t_{24} + t_1 + t_2 + t_5 + t_6 + t_{11} + t_{12} + t_{15} + t_{16}.$$

Then $t_6 = (\bar{e}_{24} + \bar{d}_{13} - \bar{b}_{24} - \bar{a}_{13})/2$. If $t_6 = 1$, then $\bar{L} = R_6$. Otherwise, from above,

$$\bar{b} = t_{13} + t_3 + t_4 + t_5 + t_6 + t_9 + t_{10} + t_{13} + t_{14},$$

$$\bar{e}_{13} = t_3 + t_4 + t_7 + t_8 + t_9 + t_{10} + t_{13} + t_{14}.$$

Applying CZRASB at points c and f,

$$\bar{c}_{24} = t_1 + t_2 + t_3 + t_8 + t_{12} + t_{16},$$

$$\bar{f}_{24} = t_1 + t_2 + t_3 + t_5 + t_6 + t_7 + t_{12} + t_{16}.$$

Then $t_7 = (\bar{f}_{24} + \bar{e}_{13} - \bar{b}_{13} - \bar{c}_{24})/2$. If $t_7 = 1$, then $\bar{L} = R_7$. Otherwise, from above,

$$\bar{d}_{24} = t_1 + t_5 + t_{10} + t_{11} + t_{12} + t_{14} + t_{15} + t_{16},$$

$$\bar{e}_{13} = t_3 + t_4 + t_7 + t_8 + t_9 + t_{10} + t_{13} + t_{14}.$$

Applying CZRASB at points g and h,

$$\bar{g}_{24} = t_1 + t_5 + t_9 + t_{14} + t_{15} + t_{16},$$

$$\bar{h}_{13} = t_3 + t_4 + t_7 + t_8 + t_{11} + t_{12} + t_{13} + t_{14}.$$

Then $t_{10} = (\bar{d}_{24} + \bar{e}_{13} - \bar{g}_{24} - \bar{h}_{13})/2$. If $t_{10} = 1$, then $\bar{L} = R_{10}$. Otherwise, $\bar{L} = R_{11}$.

Description:

(1) [Obtain endlines.] FIRST2(ℓ_{13}, ℓ_{46}, L); FIRST2(ℓ_1, ℓ_3, ℓ_{13}); FIRST2(ℓ_4, ℓ_6, ℓ_{46}); $\ell_2 \leftarrow$ RNAVER(ℓ_1, ℓ_3); $\ell_5 \leftarrow$ RNAVER(ℓ_4, ℓ_6).

(2) [Check R_6.] CZRASB(A, ℓ_1, ℓ_6, $\bar{a}_{13}$, $\bar{a}_{24}$);

CZRASB(A, ℓ_2, ℓ_6, $\overline{b}_{13}$, $\overline{b}_{24}$); CZRASB(A, ℓ_1, ℓ_5, $\overline{d}_{13}$, $\overline{d}_{24}$);

CZRASB(A, ℓ_2, ℓ_5, $\overline{e}_{13}$, $\overline{e}_{24}$); if $\overline{e}_{24} + \overline{d}_{13} - \overline{a}_{13} - \overline{b}_{24} = 0$,

go to (3); $\overline{\ell}_1 \leftarrow \ell_1$; $\overline{\ell}_3 \leftarrow \ell_2$; $\overline{\ell}_4 \leftarrow \ell_5$; $\overline{\ell}_6 \leftarrow \ell_6$;

go to (6).

(3) [Check R_7.] CZRASB(A, ℓ_3, ℓ_6, $\overline{c}_{13}$, $\overline{c}_{24}$); CZRASB(A, ℓ_3,

ℓ_5, $\overline{f}_{13}$, $\overline{f}_{24}$); if $\overline{f}_{24} + \overline{e}_{13} - \overline{b}_{13} - \overline{c}_{24} = 0$, go to (4);

$\overline{\ell}_1 \leftarrow \ell_2$; $\overline{\ell}_3 \leftarrow \ell_3$; $\overline{\ell}_4 \leftarrow \ell_5$; $\overline{\ell}_6 \leftarrow \ell_6$; go to (6).

(4) [Check R_{10}.] CZRASB(A, $\overline{\ell}_1$, $\overline{\ell}_4$, $\overline{g}_{13}$, $\overline{g}_{24}$); CZRASB(A, ℓ_2,

ℓ_4, $\overline{h}_{13}$, $\overline{h}_{24}$); if $\overline{d}_{24} + \overline{e}_{13} - \overline{g}_{24} - \overline{h}_{13} = 0$, go to (5);

$\overline{\ell}_1 \leftarrow \ell_1$; $\overline{\ell}_3 \leftarrow \ell_2$; $\overline{\ell}_4 \leftarrow \ell_4$; $\overline{\ell}_6 \leftarrow \ell_5$; go to (6).

(5) [Zero is in R_{11}.] $\overline{\ell}_1 \leftarrow \ell_1$; $\overline{\ell}_2 \leftarrow \ell_3$; $\overline{\ell}_3 \leftarrow \ell_4$; $\overline{\ell}_4 \leftarrow \ell_6$;

$\overline{\ell}_5 \leftarrow \ell_5$.

(6) [Finish.] $\overline{\ell}_{13} \leftarrow$ PFL(BORROW($\overline{\ell}_1$), PFL(BORROW($\overline{\ell}_3$), 0));

$\overline{\ell}_{46} \leftarrow$ PFL(BORROW($\overline{\ell}_4$), PFL(BORROW($\overline{\ell}_6$), 0)); $\overline{L} \leftarrow$ PFL($\overline{\ell}_{13}$,

PFL($\overline{\ell}_{46}$, 0)); erase ℓ_2, ℓ_5; return.

Computing Time: $\propto \mu^{4+k}\{\mu L(\mu) + \mu\lambda + L(d)\}^2$, where $L = ((\ell_1, \ell_3), (\ell_4, \ell_6))$, $\lambda = L(\ell_1) + L(\ell_3) + L(\ell_4) + L(\ell_6)$,

$m = \deg(A)$, $\mu = m + 1$, $d = |A|_\infty$, and $k = 0$ if modular

methods are used or the p.r.s.'s are normal, k = 1 otherwise.

Proof: $t_{RNAVER}(\ell_1,\ell_3) \underline{\propto} L(\ell_1)L(\ell_3)$ and $t_{RNAVER}(\ell_4,\ell_6) \underline{\propto} L(\ell_4)L(\ell_6)$. Let $\ell_i = \ell_{i1}/\ell_{i2}$. Then $L(\ell_2) = L((1/2)\cdot(\ell_1 + \ell_3)) \leq L(1/2) + L(\ell_1 + \ell_3) \leq 3 + L(\ell_1 + \ell_3) \underline{\propto} L(\ell_1) + L(\ell_3)$. Similarly $L(\ell_5) \underline{\propto} L(\ell_4) + L(\ell_6)$.

Hence the time for applying CZRASB is $\underline{\propto}$

$$\sum_{i=1}^{3}\sum_{j=4}^{6}\{\mu^{4+k}[\mu L(\mu) + \mu\{L(\ell_i) + L(\ell_j)\} + L(d)]^2\} \underline{\propto}$$

$$\mu^{4+k}\{\mu L(\mu) + \sum_{i=1}^{3}\sum_{j=4}^{6}\mu[L(\ell_i) + L(\ell_j)] + L(d)\}^2 \sim$$

$$\mu^{4+k}\{\mu L(\mu) + \mu\lambda + L(d)\}^2.$$

Hence $t_{CZREF}(A,L) \underline{\propto} \mu^{4+k}\{\mu L(\mu) + \mu\lambda + L(d)\}^2$. ∎

The preceding algorithm is satisfactory for a single refinement. However, in many instances it will be necessary to refine the rectangle repeatedly until its maximum dimension is less than some specified value. The preceding algorithm is not satisfactory for this because it must recompute costly information which could have been saved from the previous refinement.

The following algorithm alleviates this problem by providing additional parameters which allow transfer of this

information. It should be noted here that the user will probably not call this algorithm directly. Rather, it is employed as a sub-algorithm for CZRTSA which simplifies the entire process of individual root refinement from the user point of view.

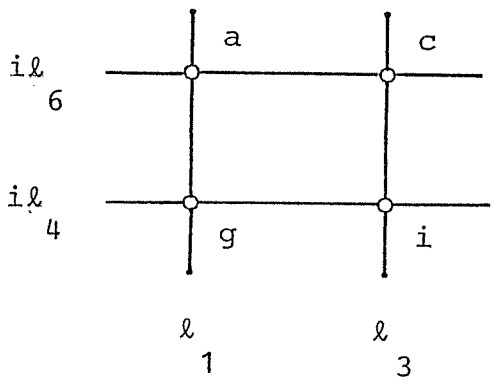
ALGORITHM CZREFA:

CZREFA(A,L,$\overline{L}$,$\overline{a}_{13}$,$\overline{c}_{24}$,$\overline{g}_{24}$)

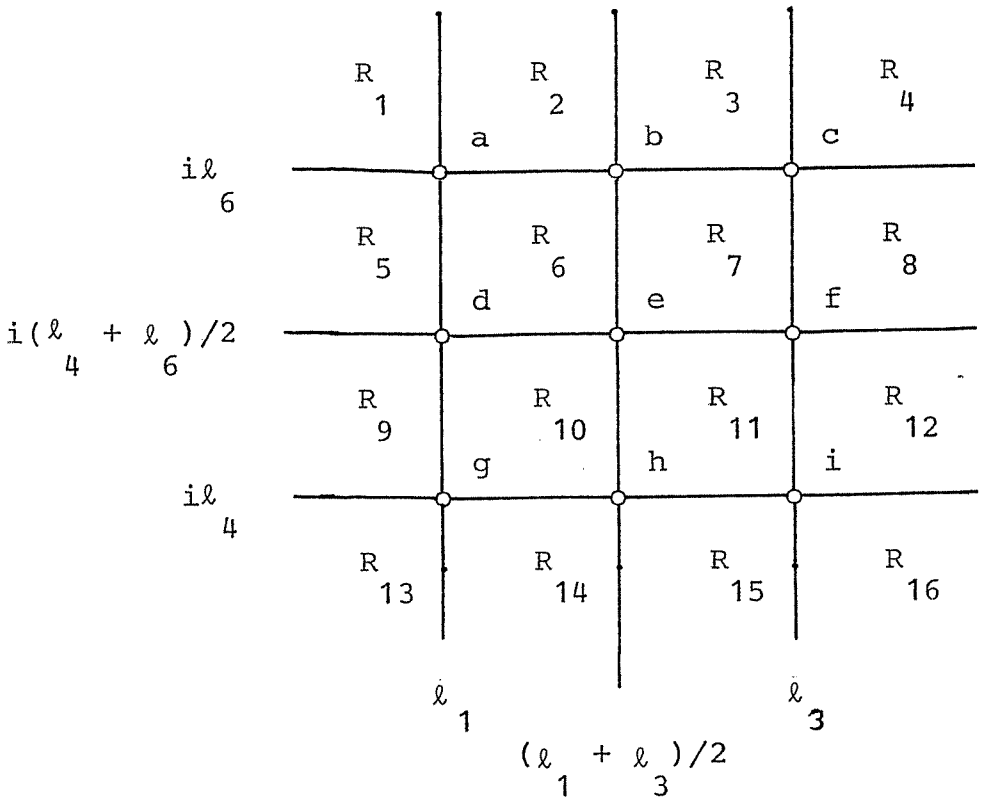
Complex Zero Refinement with

Auxiliary Quantities

A, L, $\overline{a}_{13}$, $\overline{c}_{24}$, $\overline{g}_{24}$ are inputs; $\overline{L}$ is an output; $\overline{a}_{13}$, $\overline{c}_{24}$, $\overline{g}_{24}$ are modified. A is a non-zero square-free univariate Gaussian polynomial. L is the interval representation of a rectangle in the complex plane containing (in the standard rectangle sense) exactly one zero of A. That is, L = $((\ell_1,\ell_3),(\ell_4,\ell_6))$, ℓ_i rational numbers, for the following rectangle R:



and the zero lies in the interior of the rectangle, on line (a,c], or on line (i,c]. Consider four subrectangles of R and surrounding standard rectangles as follows:



Let t_j be the number of zeros of A in R_j. Then $\bar{a}_{13}$, $\bar{c}_{24}$, $\bar{g}_{24}$ are the Fortran integers

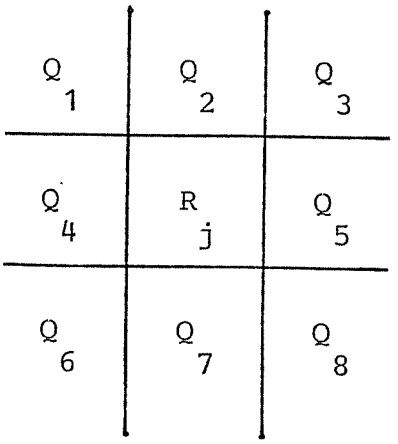
$$\bar{a}_{13} = t_2 + t_3 + t_4 + t_5 + t_9 + t_{13},$$

$$\bar{c}_{24} = t_1 + t_2 + t_3 + t_8 + t_{12} + t_{16},$$

$$\bar{g}_{24} = t_1 + t_5 + t_9 + t_{14} + t_{15} + t_{16}.$$

The output $\bar{L}$ is the interval representation of the

standard subrectangle of R, either R_6, R_7, R_{10}, or R_{11}, containing the zero of A. Let R_j be the one and consider the following standard rectangles:



Then let s_j be the number of zeros in Q_j. Then the new values of $\bar{a}_{13}$, $\bar{c}_{24}$, $\bar{g}_{24}$ are

$$\bar{a}_{13} = s_2 + s_3 + s_4 + s_6 ,$$

$$\bar{c}_{24} = s_1 + s_2 + s_5 + s_8 ,$$

$$\bar{g}_{24} = s_1 + s_4 + s_7 + s_8 .$$

Method:

Applying CZRASB at points b, d, e gives

$$\bar{b}_{24} = t_1 + t_2 + t_7 + t_8 + t_{11} + t_{12} + t_{15} + t_{16} ,$$

$$\bar{d}_{13} = t_2 + t_3 + t_4 + t_6 + t_7 + t_8 + t_9 + t_{13} ,$$

$$\bar{e}_{24} = t_1 + t_2 + t_5 + t_6 + t_{11} + t_{12} + t_{15} + t_{16} .$$

Then $t_6 = (\bar{e}_{24} + \bar{d}_{13} - \bar{b}_{24} - \bar{a}_{13})/2$. If $t_6 = 1$, then

$\bar{L}_6 = R_{13}$, $\bar{a}_{13}$ is not changed, $\bar{c}_{24} \leftarrow \bar{b}_{24}$, and $\bar{g}_{24} \leftarrow \bar{d}_{24}$.

Otherwise, from above,

$$\bar{b}_{13} = t_3 + t_4 + t_5 + t_6 + t_9 + t_{10} + t_{13} + t_{14},$$

$$\bar{e}_{13} = t_3 + t_4 + t_7 + t_8 + t_9 + t_{10} + t_{13} + t_{14}.$$

Applying CZRASB at point f,

$$\bar{f}_{24} = t_1 + t_2 + t_3 + t_5 + t_6 + t_7 + t_{12} + t_{16}.$$

Then $t_7 = (\bar{f}_{24} + \bar{e}_{13} - \bar{b}_{13} - \bar{c}_{24})/2$. If $t_7 = 1$, then

$\bar{L}_7 = R_{13}$, $\bar{a}_{13} \leftarrow \bar{b}_{13}$, $\bar{c}_{24}$ is not changed, $\bar{g}_{24} \leftarrow \bar{e}_{24}$.

Otherwise, from above,

$$\bar{d}_{24} = t_1 + t_5 + t_{10} + t_{11} + t_{12} + t_{14} + t_{15} + t_{16},$$

$$\bar{e}_{13} = t_3 + t_4 + t_7 + t_8 + t_9 + t_{10} + t_{13} + t_{14}.$$

Applying CZRASB at point h,

$$\bar{h}_{13} = t_3 + t_4 + t_7 + t_8 + t_{11} + t_{12} + t_{13} + t_{14}.$$

Then $t_{10} = (\bar{d}_{24} + \bar{e}_{13} - \bar{g}_{24} - \bar{h}_{13})/2$. If $t_{10} = 1$,

then $\bar{L}_{10} = R_{13}$, $\bar{a}_{13} \leftarrow \bar{d}_{13}$, $\bar{c}_{24} \leftarrow \bar{e}_{24}$, and $\bar{g}_{24}$ is not

changed. Otherwise $\bar{L}_{11} = R_{13}$, $\bar{a}_{13} \leftarrow \bar{e}_{13}$, $\bar{c}_{24} \leftarrow \bar{f}_{24}$,

and $\bar{g}_{24} \leftarrow \bar{h}_{24}$.

Description:

(1) [Obtain endlines.] FIRST2(ℓ_{13}, ℓ_{46}, L); FIRST2(ℓ_1, ℓ_3, ℓ_{13}); FIRST2(ℓ_4, ℓ_6, ℓ_{46}); $\ell_2 \leftarrow$ RNAVER(ℓ_1, ℓ_3); $\ell_5 \leftarrow$ RNAVER(ℓ_4, ℓ_6).

(2) [Check R_6.] CZRASB(A, ℓ_2, ℓ_6, $\overline{b}_{13}$, $\overline{b}_{24}$); CZRASB(A, ℓ_1, ℓ_5, $\overline{d}_{13}$, $\overline{d}_{24}$); CZRASB(A, ℓ_2, ℓ_5, $\overline{e}_{13}$, $\overline{e}_{24}$); if $\overline{e}_{24} + \overline{d}_{13} - \overline{a}_{13} - \overline{b}_{24} = 0$, go to (3); $\overline{\ell}_1 \leftarrow \ell_1$; $\overline{\ell}_1 \leftarrow \ell_3$; $\overline{\ell}_2 \leftarrow \ell_4$; $\overline{\ell}_5 \leftarrow \ell_6$; $\overline{\ell}_6 \leftarrow \ell_6$; $\overline{c}_{24} \leftarrow \overline{b}_{24}$; $\overline{g}_{24} \leftarrow \overline{d}_{24}$; go to (6).

(3) [Check R_7.] CZRASB(A, ℓ_3, ℓ_5, $\overline{f}_{13}$, $\overline{f}_{24}$); if $\overline{f}_{24} + \overline{e}_{13} - \overline{b}_{13} - \overline{c}_{24} = 0$, go to (4); $\overline{\ell}_1 \leftarrow \ell_2$; $\overline{\ell}_2 \leftarrow \ell_3$; $\overline{\ell}_3 \leftarrow \ell_4$; $\overline{\ell}_4 \leftarrow \ell_5$; $\overline{\ell}_6 \leftarrow \ell_6$; $\overline{a}_{13} \leftarrow \overline{b}_{13}$; $\overline{g}_{24} \leftarrow \overline{e}_{24}$; go to (6).

(4) [Check R_{10}.] CZRASB(A, ℓ_2, ℓ_4, $\overline{h}_{13}$, $\overline{h}_{24}$); if $\overline{d}_{24} + \overline{e}_{13} - \overline{g}_{24} - \overline{h}_{13} = 0$, go to (5); $\overline{\ell}_1 \leftarrow \ell_1$; $\overline{\ell}_1 \leftarrow \ell_3$; $\overline{\ell}_2 \leftarrow \ell_4$; $\overline{\ell}_4 \leftarrow \ell_4$; $\overline{\ell}_6 \leftarrow \ell_5$; $\overline{a}_{13} \leftarrow \overline{d}_{13}$; $\overline{c}_{24} \leftarrow \overline{e}_{24}$; go to (6).

(5) [Is in R_{11}.] $\overline{\ell}_1 \leftarrow \ell_1$; $\overline{\ell}_2 \leftarrow \ell_3$; $\overline{\ell}_3 \leftarrow \ell_4$; $\overline{\ell}_4 \leftarrow \ell_6$; $\overline{\ell}_5 \leftarrow \ell_5$; $\overline{a}_{13} \leftarrow \overline{e}_{13}$; $\overline{c}_{24} \leftarrow \overline{f}_{24}$; $\overline{g}_{24} \leftarrow \overline{h}_{24}$.

(6) [Finish.] $\overline{\ell}_{13} \leftarrow$ PFL(BORROW($\overline{\ell}_1$), PFL(BORROW($\overline{\ell}_3$), 0));

$\overline{\ell}_{46} \leftarrow$ PFL(BORROW($\overline{\ell}_4$),PFL(BORROW($\overline{\ell}_6$),0)); $\overline{L} \leftarrow$ PFL($\overline{\ell}_{13}$,PFL($\overline{\ell}_{46}$,0)); erase ℓ_2, ℓ_5; return.

Computing Time: $\preceq \mu^{4+k}\{\mu L(\mu) + \mu\lambda + L(d)\}^2$, where $L = ((\ell_1,\ell_3),(\ell_4,\ell_6))$, $\lambda = L(\ell_1) + L(\ell_3) + L(\ell_4) + L(\ell_6)$, $m = \deg(A)$, $\mu = m + 1$, $d = |A|_\infty$, and $k = 0$ if modular methods are used or the p.r.s.'s are normal, $k = 1$ otherwise.

Proof: $t_{RNAVER}(\ell_1,\ell_3) \preceq L(\ell_1)L(\ell_3)$ and $t_{RNAVER}(\ell_4,\ell_6) \preceq L(\ell_4)L(\ell_6)$. Let $\ell_i = \ell_{i1}/\ell_{i2}$. Then $L(\ell_2) = L((1/2)\cdot(\ell_1 + \ell_3)) \leq L(1/2) + L(\ell_1 + \ell_3) \leq 3 + L(\ell_1) + L(\ell_3) \preceq L(\ell_1) + L(\ell_3)$. Similarly, $L(\ell_5) \preceq L(\ell_4) + L(\ell_2)$. Hence the time for applying CZRASB is $\preceq$

$$\sum_{i=1}^{3}\sum_{j=4}^{6}\{\mu^{4+k}[\mu L(\mu) + \mu\{L(\ell_i) + L(\ell_j)\} + L(d)]^2\} \preceq$$

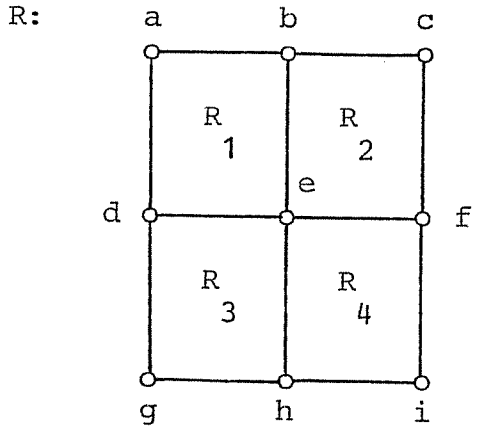
$$\mu^{4+k}\{\mu L(\mu) + \sum_{i=1}^{3}\sum_{j=4}^{6}\mu[L(\ell_i) + L(\ell_j)] + L(d)\}^2 \sim$$

$$\mu^{4+k}\{\mu L(\mu) + \mu\lambda + L(d)\}^2.$$

Hence $t_{CZREFA}(A,L) \preceq \mu^{4+k}\{\mu L(\mu) + \mu\lambda + L(d)\}^2$. ∎

The following algorithm will probably be the one most convenient to use for individual root refinement. It determines how many refinements are necessary to achieve the specified maximum dimension, and based on this applies the method which will give the lower average computing time.

The method to be used is selected according to the probable number of applications of CZRASB, which will comprise the major portion of computing time. Consider the following rectangles and sub-rectangles.



If CZREF is used, CZRASB must be applied at a, b, d, and e to determine if the root is in R_1. If it is not, R_2 requires CZRASB at c and f, and finally R_3 requires it at g and h. Since there is only one root in the rectangle, R_4 needs no additional applications--if it is not in R_1, R_2, or R_3 it must be in R_4. Assuming each area is as likely to contain the root, the average number of

applications of CZRASB is $(4 + 6 + 8 + 8)/4 = 6.5$. Hence, for n refinements CZRASB must be called an average of 6.5n times.

CZREFA requires three initial applications of CZRASB, at a, c, and g. Subsequently, these parameters are updated from information available in the computations. Hence CZRASB must be applied at the following points: R_1, at b, e, and d; R_2, at f; R_3, at h. Therefore the average per refinement is $(3 + 4 + 5 + 5)/4 = 4.25$. This gives an overall average of $3 + 4.25n$ applications of CZRASB for n refinements.

Hence, the following algorithm uses CZREF if only one refinement is required, and CZREFA if more than one is required.

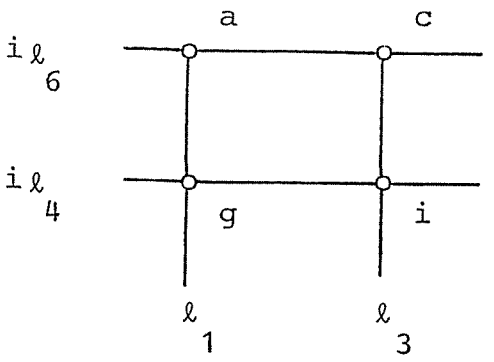
ALGORITHM CZRTSA:

$\bar{L}$=CZRTSA(G,L,e)

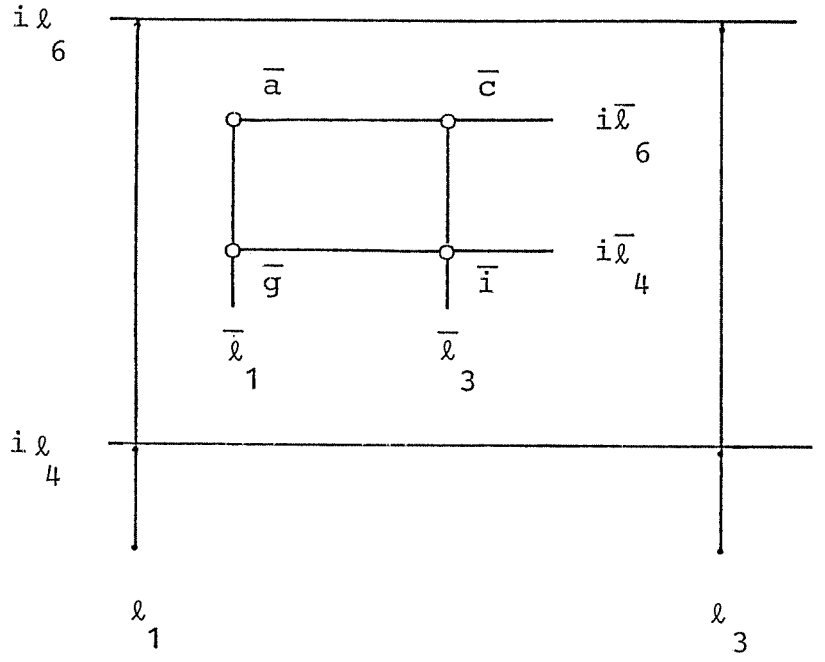
Complex Zero Refinement

to Specified Accuracy

G is a non-zero square-free univariate Gaussian polynomial. L is the interval representation of a standard rectangle in the complex plane containing exactly one zero of A. That is, $L = ((\ell_1, \ell_3), (\ell_4, \ell_6))$, ℓ_i rational numbers, for the following rectangle R:



The zero lies in the interior of the rectangle, on line (a,c], or on line (i,c]. e is a positive rational number. Let $\overline{R}$ be a subrectangle of R as in the following diagram:



where the zero of A in R is in $\overline{R}$, $(\overline{\ell}_3 - \overline{\ell}_1)/(\overline{\ell}_6 - \overline{\ell}_4) = (\ell_3 - \ell_1)/(\ell_6 - \ell_4)$, and $\max\{(\overline{\ell}_3 - \overline{\ell}_1), (\overline{\ell}_6 - \overline{\ell}_4)\} < e$. Then $\overline{L}$ is the interval representation of $\overline{R}$.

Method:

On the average, CZREF requires 6.5 applications of CZRASB while CZREFA requires 3 initial applications plus an average of 4.25 applications per subsequent call. Let $d = \max\{(\ell_3 - \ell_1), (\ell_6 - \ell_4)\}$. If $d < e$, then no refinement is necessary. Otherwise, $k = \lfloor \log_2 d/e \rfloor + 1$ refinements are necessary to obtain the required accuracy. If $k = 1$, then CZREF is used. If $k \geq 2$, then CZREFA is used. Note that d/e is a rational number and hence ELPOF2 cannot be applied to it. However, let $\bar{k} = \lfloor \log_2 d/e \rfloor$, so that $\bar{k} \leq \log_2 d/e < \bar{k} + 1$. Then $2^{\bar{k}} \leq d/e < 2^{\bar{k}+1}$, $2^{\bar{k}} \leq \lfloor d/e \rfloor < 2^{\bar{k}+1}$, $\bar{k} \leq \log_2 \lfloor d/e \rfloor < \bar{k} + 1$, $\bar{k} \leq \lfloor \log_2 \lfloor d/e \rfloor \rfloor < \bar{k} + 1$, and $\bar{k} = \lfloor \log_2 \lfloor d/e \rfloor \rfloor$. Hence, ELPOF2 can be applied to $\lfloor d/e \rfloor$ to compute $\bar{k}$ and then $k = \bar{k} + 1$.

Description:

(1) [Initialize.] FIRST2(ℓ_{13}, ℓ_{46}, L); FIRST2(ℓ_1, ℓ_3, ℓ_{13}); FIRST2(ℓ_4, ℓ_6, ℓ_{46}); $d_1 \leftarrow$ RDIF(ℓ_3, ℓ_1); $d_2 \leftarrow$ RDIF(ℓ_6, ℓ_4); if RCOMP(d_1, d_2) ≤ 0, ($d_2 \leftarrow d_1$; erase d_1; go to (2)); $d_1 \leftarrow d_2$; erase d_2.

(2) [Compute number of refinements.] If RCOMP(d,e) < 0, ($\overline{L} \leftarrow$ BORROW(L); erase d; return); $r \leftarrow$ RQ(d,e); FIRST2(r_1,r_2,r); $q \leftarrow$ IQ(r_1,r_2); ELPOF2(q,$\overline{k}$,t); $k \leftarrow \overline{k} + 1$; erase r, q, d; if $k \geq 2$, go to (4).

(3) [Use single refinement.] $\overline{L} \leftarrow$ CZREF(G,L); return.

(4) [Use multiple refinements.] CZRASB(G,ℓ_1,ℓ_6,$\overline{a}_{13}$, $\overline{a}_{24}$); CZRASB(G,ℓ_3,ℓ_6,$\overline{c}_{13}$,$\overline{c}_{24}$); CZRASB(G,ℓ_1,ℓ_4,$\overline{g}_{13}$, $\overline{g}_{24}$); $T \leftarrow$ BORROW(L).

(5) [Loop.] CZREFA(G,T,$\overline{L}$,$\overline{a}_{13}$,$\overline{c}_{24}$,$\overline{g}_{24}$); erase T; $k \leftarrow k - 1$; if $k \leq 0$, return; $T \leftarrow \overline{L}$; go to (5).

Computing Time: $\propto \{\lambda + L(e)\}^2 + n\mu^{4+k}\{\mu L(\mu) + \mu n + \mu\lambda + L(a)\}^2$, where $m = \deg(A)$, $\mu = m + 1$, $a = |A|_\infty$, $L = ((\ell_1,\ell_3),(\ell_4,\ell_6))$, $\lambda = L(\ell_1) + L(\ell_3) + L(\ell_4) + L(\ell_6)$, $d = \max\{(\ell_3 - \ell_1),(\ell_6 - \ell_4)\}$, $n = \max\{0, \lfloor \log_2 d/e \rfloor + 1\}$, and $k = 0$ if modular methods are used or the p.r.s.'s are normal, $k = 1$ otherwise.

Proof: Heindel has shown [HEL70] that if the interval (p/q, r/s] is bisected j times then the maximum numerator and denominator is at most $2^{j+1}(\max\{|p|,|q|,|r|,|s|\})^2$.

Let $\bar{L}_j = ((\bar{\ell}_{j1}, \bar{\ell}_{j3}), (\bar{\ell}_{j4}, \bar{\ell}_{j6}))$ be the j-th input to CZREF or CZREFA. If $\bar{\ell}_{ji} = \bar{\ell}_{ji1} / \bar{\ell}_{ji2}$ then $M_j =$

$$\max_{i=1,3,4,6} \{|\bar{\ell}_{ji1}|, |\bar{\ell}_{ji2}|\} \leq 2^{j+1} [\max_{i=1,3,4,6} \{|\ell_{i1}|, |\ell_{i2}|\}] \leq 2^{j+1} \sum_{i=1,3,4,6} \sum_{j=1}^{2} \ell_{ij}^2 \text{ so } L(M_j) \propto (j+1)$$

$$+ \sum_{i=1,3,4,6} \sum_{j=1}^{2} 2L(\ell_{ij}) = (j+1) + 2\lambda \sim j + \lambda.$$

Since $\lambda_j = \sum_{k=1,3,4,6} L(\bar{\ell}_{jk}) \propto j + \lambda$ the time for any call of CZREF or CZREFA is $\propto \mu^{4+k} \{\mu L(\mu) + \mu n + \mu\lambda + L(a)\}^2$ and the time for all is $\propto n\mu^{4+k} \{\mu L(\mu) + \mu n + \mu\lambda + L(a)\}^2$.

$t_{RDIF}(\ell_3, \ell_1) \propto \{L(\ell_1) + L(\ell_3)\}^2$ and $t_{RDIF}(\ell_6, \ell_4)$ $\propto \{L(\ell_4) + L(\ell_6)\}^2$. $L(d_1) \propto L(\ell_1) + L(\ell_3)$ and $L(d_2) \propto L(\ell_4) + L(\ell_6)$ so $t_{RCOMP}(d_1, d_2) \propto \lambda^2$ and $t_{RCOMP}(d, e) \propto \{\lambda + L(e)\}^2$.

Hence the total is $\propto \{\lambda + L(e)\}^2 + n\mu^{4+k} \{\mu L(\mu) + \mu n + \mu\lambda + L(a)\}^2$ ∎

The final algorithm in this section refines several areas to a specified width.

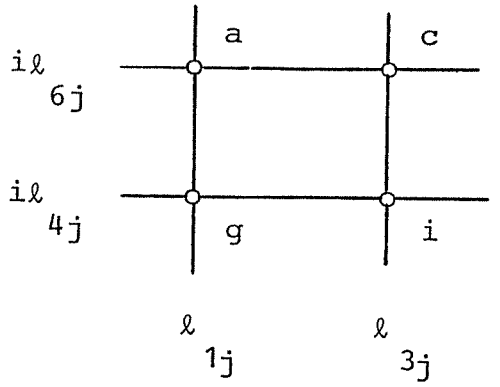
ALGORITHM CZMRSA:

$\overline{L}$=CZMRSA(A,L,e)

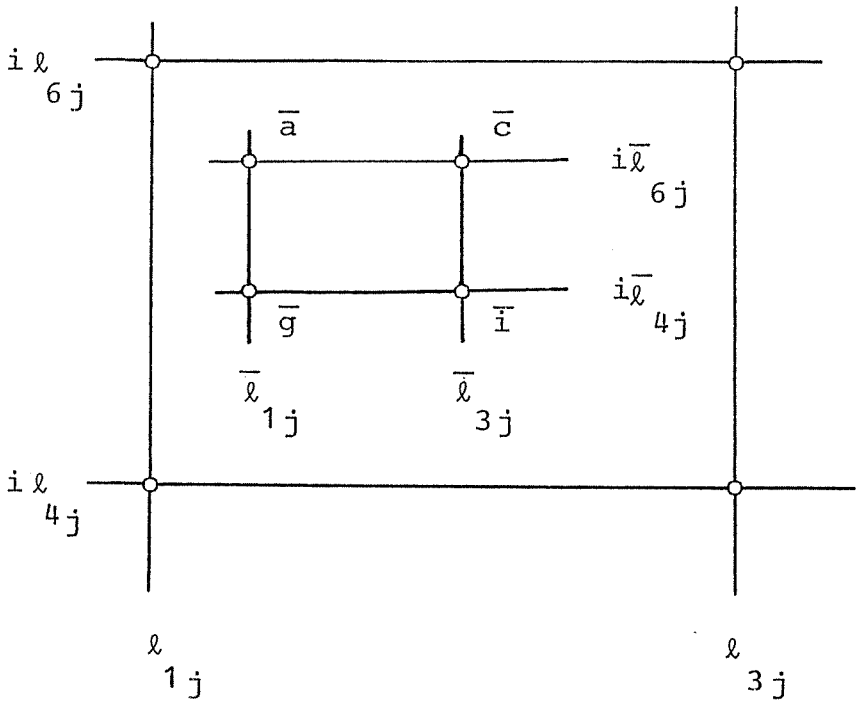
Complex Zero System, Multiple Refinement

to Specified Accuracy

A is a non-zero square-free univariate Gaussian polynomial. L is a list $(L_1, L_2, \ldots, L_n)$, where L_j is the interval representation of a standard rectangle in the complex plane containing exactly one zero of A. That is, $L_j = ((\ell_{1j}, \ell_{3j}), (\ell_{4j}, \ell_{6j}))$, ℓ_{ij} rational numbers, for the following rectangle R_j:



The zero lies in the interior of the rectangle, on line (a,c], or on line (i,c]. Let $\overline{R}_j$ be a sub-rectangle of R_j as in the following diagram:



where the zero of A in R_j is in $\bar{R}_j$, $(\bar{\ell}_{3j} - \bar{\ell}_{1j})/(\bar{\ell}_{6j} - \bar{\ell}_{4j}) = (\ell_{3j} - \ell_{1j})/(\ell_{6j} - \ell_{4j})$, and $\max\{(\bar{\ell}_{3j} - \bar{\ell}_{1j}), (\bar{\ell}_{6j} - \bar{\ell}_{4j})\} < e$. Then $\bar{L}$ is the list $(\bar{L}_1, \bar{L}_2, \ldots, \bar{L}_n)$ where $\bar{L}_j$ is the interval representation of $\bar{R}_j$.

Description:

(1) [Initialize.] $\bar{L} \leftarrow 0$; $T \leftarrow L$.

(2) [Loop.] ADV(L_j, T); $\bar{L}_j \leftarrow$ CZRTSA(A, L_j, e); $\bar{L} \leftarrow$ PFL($\bar{L}_j$, $\bar{L}$); if $T \neq 0$, go to (2); $\bar{L} \leftarrow$ INV($\bar{L}$); return.

Computing Time: $\propto \ell\{\lambda + L(e)\}^2 + \ell n\mu^{4+k}\{\mu L(\mu) + \mu n + \mu\lambda + L(a)\}^2$, where $m = \deg(A)$, $\mu = m + 1$, $a = |A|_\infty$,

$L = (L_1, L_2, \ldots, L_\ell)$, $L_i = ((\ell_{1i}, \ell_{3i}), (\ell_{4i}, \ell_{6i}))$, $\lambda_i = L(\ell_{1i}) + L(\ell_{3i}) + L(\ell_{4i}) + L(\ell_{6i})$, $d_i = \max\{(\ell_{3i} - \ell_{1i}), (\ell_{6i} - \ell_{4i})\}$, $n_i = \max\{0, \lfloor \log_2 d_i / e \rfloor + 1\}$, $n = \max_{1 \leq i \leq \ell} \{n_i\}$, $\lambda = \max_{1 \leq i \leq \ell} \{\lambda_i\}$, and $k = 0$ if modular methods are used or the p.r.s.'s are normal, $k = 1$ otherwise.

Proof: Follows immediately from the time for CZRTSA. ∎

6.2 Square-free factorization

The concept of a square-free factorization of a polynomial A was introduced in Section 2.2, where it was defined as a sequence of polynomials $A_1, A_2, \ldots, A_n$ of positive degrees and a sequence of distinct positive integers $e_1, e_2, \ldots, e_n$ such that $A = \prod_{j=1}^{n} A_j^{e_j}$ and $\prod_{j=1}^{n} A_j$ is square-free.

Such a factorization is important in this system because it provides a means for both isolating and refining the roots of a polynomial and, in addition, specifying the multiplicity of each root. The algorithms of Chapter 5 can be applied to each A_j, since it is square-free; the multiplicity of the roots of A_j is then e_j.

The following two algorithms compute a square-free factorization, using a method presented in Musser [MUD71].

ALGORITHM PSQFF:

L=PSQFF(A)

Polynomial Square-Free Factorization

A is a positive, primitive, univariate integral polynomial of positive degree. L is the list $(i_1, Q_{i_1}, \ldots, i_k, Q_{i_k})$ where $\prod_{i=1}^{k} Q_{i_i}^{i}$ is the square-free

factorization of the primitive part of A, $i_1 < i_2 <$ $i_k = \ell$, and $\{i_1, \ldots, i_k\}$ is the set of all i, $1 \leq i \leq \ell$ such that $Q_i \neq 1$.

Method:

Let $B_j = \prod_{i=j+1}^{\ell} Q_i^{i-j}$ and $C_j = \prod_{i=j}^{\ell} Q_i$ for $1 \leq j \leq \ell$. Then $B_1 = \gcd(A,A')$ and $C_1 = A/B_1$. For $1 \leq j < \ell$ (which is equivalent to $\deg(B_j) > 0$), $C_{j+1} = \gcd(B_j, C_j)$, $B_{j+1} = B_j/C_{j+1}$, and $Q_j = C_j/C_{j+1}$. Finally, $Q_\ell = C_\ell$.

Description:

(1) [Initialize.] L ← 0; v ← PVBL(A); A' ← PDERIV(A,v); erase v; PGCDCF(A,A',B,C,T); erase A',T; j ← 1; go to (3).

(2) [Loop.] PGCDCF(B,C,$\overline{C}$,$\overline{B}$,Q); if PDEG(Q) > 0, L ← PFL(BORROW(Q),PFA(j,L)); erase B, C, Q; B ← $\overline{B}$; C ← $\overline{C}$; j ← j + 1.

(3) [Check for end.] If PDEG(B) > 0, go to (2).

(4) [Finish.] Erase B; if PDEG(C) > 0, L ← PFL(BORROW(C), PFA(j,L)); erase C; L ← INV(L); return.

Computing Time: $\propto \mu^4 L(\mu d)^2$, where m = deg(A), μ = m + 1,

and $d = \max\{|D|_\infty : D|A\}$.

Proof: B, C, and $\overline{C}$ divide A, so the time for each application of PGCDCF is $\preceq \mu^3 L(\mu d)^2$. $\ell \leq \mu$, hence there are at most μ calls of PGCDCF. ∎

ALGORITHM GPSQFF:

L=GPSQFF(A)

Gaussian Polynomial Square

Free Factorization

A is a primitive univariate Gaussian polynomial of positive degree. L is the list $(i_1, Q_{i_1}, \ldots, i_k, Q_{i_k})$ where $\prod_{i=1}^{\ell} Q_i^i$ is the square-free factorization of the primitve part of A, $i_1 < i_2 < \ldots < i_k = \ell$, and $\{i_1, \ldots, i_k\}$ is the set of all i, $1 \leq i \leq \ell$ such that $Q_i \neq 1$.

Method:

Let $B_j = \prod_{i=j+1}^{\ell} Q_i^{i-j}$ and $C_j = \prod_{i=j}^{\ell} Q_i$ for $1 \leq j \leq \ell$. Then $B_1 = \gcd(A,A')$ and $C_1 = A/B_1$. For $1 \leq j < \ell$ (which is equivalent to $\deg(B_j) > 0$), $C_{j+1} \simeq \gcd(B_j, C_j)$, $B_{j+1} \simeq B_j/C_{j+1}$, and $Q_j \simeq C_j/C_{j+1}$. Finally,

$Q_\ell = C_\ell$.

Description:

(1) [Initialize.] L ← 0; v ← GPVBL(A); A' ← GPDERV(A, v); erase v; GPGCDC(A,A',B,C,T); erase A', T; j ← 1; go to (3).

(2) [Loop.] GPGCDC(B,C,$\overline{C}$,$\overline{B}$,Q); if GPDEG(Q) > 0, L ← PFL(GPFQ(Q),PFA(j,L)); erase B, C, Q; B ← $\overline{B}$; C ← $\overline{C}$; j ← j + 1.

(3) [Check for end.] If GPDEG(B) > 0, go to (2).

(4) [Finish.] Erase B; if GPDEG(C) > 0, L ← PFL(GPFQ(C), PFA(j,L)); erase C; L ← INV(L); return.

Computing Time: $\propto \mu^4 L(\mu d)^2$, where m = deg(A), μ = m + 1, and d = max$\{|D|_\infty : D|A\}$.

Proof: B, C, and $\overline{C}$ divide A, so the time for each application of GPGCDC is $\propto \mu^3 L(\mu d)^2$. $\ell \leq \mu$, hence there are at most μ calls of GPGCDC. ∎

6.3 Roots of multiplicity greater than one

As mentioned in Section 2.2, it is sometimes necessary to know both the location and the multiplicity of the roots of a non-square-free polynomial A. The algorithms of that section provide a means for computing a greatest square-free divisor of A, from which the locations of the roots of A can be computed. However, the multiplicities are ignored.

The algorithms in this section, on the other hand, first apply the algorithms of Section 6.2 to obtain a square-free factorization of A, say $\prod_{j=1}^{n} A_j^{e_j}$. The algorithms of Chapter 5 are then applied to each A_j, obtaining a list of areas, say $L_j = (L_{j,1}, \ldots, L_{j,m_j})$ where $m_j = \deg(A_j)$. By pairing with each of these L_j the number e_j, one obtains a list $(e_1, L_1, \ldots, e_n, L_n)$. The concatenation of the L_j, $(L_{1,1}, \ldots, L_{1,m_1}, \ldots, L_{n,1}, \ldots, L_{n,m_n})$, provides a list of the locations of all of the unique roots of A. A similar list would have been produced by applying the algorithms of Chapter 5 to

the greatest square-free divisor of A. However, the multiplicity, e_j, of the root in area $L_{j,k}$ is now also available.

ALGORITHM PMRIR:

R=PMRIR(P,ε)

Polynomial with Arbitrary

Multiplicity Roots, Isolation

and Refinement

P is a univariate integral polynomial of positive degree. ε is a non-negative rational number. R is the list $(e_1, r_1, \ldots, e_j, r_j)$, $e_1 < e_2 < \ldots < e_r$. Each r_k is a list $(r_{k,1}, \ldots, r_{k,\ell_k})$ where $r_{k,p}$ is the interval representation of a rectangle in the complex plane, of the kind produced by PRIR, containing exactly one root $\alpha_{k,p}$ of P. e_k is the multiplicity of each root $\alpha_{k,p}$ of P and $\{\alpha_{k,1}, \ldots, \alpha_{k,m_k}\}$ are all the roots of P of multiplicity e_k. Hence for b = ldcf(P), $P(x) = b\prod_{k=1}^{j}\{\prod_{p=1}^{m_k}(x - \alpha_{k,p})^{e_k}\}$. If $\varepsilon \neq 0$, then each rectangle containing a root has width $w < \varepsilon$.

Method:

Let $b\prod_{j=1}^{n} A_j^{e_j}$ be the square-free factorization of P.

Then PRIR is applied to each A_j.

Description:

(1) [Initialize; obtain factors and multiplicities.] $R \leftarrow 0$; $\overline{P} \leftarrow PPP(P)$; $L \leftarrow PSQFF(\overline{P})$; erase $\overline{P}$.

(2) [Isolate and refine roots for each polynomial factor.] DECAP2(m,Q,L); $I \leftarrow PRIR(Q,\varepsilon)$; erase Q; $R \leftarrow PFL(I,PFA(m,R))$; if $L \neq 0$, go to (2).

(3) [Finish.] $R \leftarrow INV(R)$; return.

Computing Time: $\propto \mu^{7+k} L(dh)L(\mu dh)^2 + \mu L(\varepsilon)L(dh)^2$, where $m = \deg(A)$, $\mu = m + 1$, $d = \max\{|D|_\infty : D|A\}$, $\lambda = \min\{sep(A), 1/d\}$ if A has more than one distinct root and $\lambda = 1/d$ otherwise, $\delta = \lambda$ if $\varepsilon = 0$ and $\delta = \min\{\lambda, \varepsilon\}$ otherwise, $h = \lceil 1/\delta \rceil$, $k = 0$ if modular methods are used or the p.r.s.'s are normal, $k = 1$ otherwise.

Proof: Let b_i be the bound on the roots of polynomial Q_i computed by GPRBND, $m_i = \deg(Q_i)$, $\mu_i = m_i + 1$, $d_i = |Q_i|_\infty$, $\lambda_i = sep(Q_i)$ if $m_i > 1$ and $\lambda_i = 1/b_i$ otherwise, $\delta_i = \lambda_i$ if $\varepsilon = 0$ and $\delta_i = \min\{\lambda_i, \varepsilon\}$ otherwise, and $h_i = \lceil 1/\delta_i \rceil$. Then the time for all executions of PRIR, T,

$$\text{is } T \propto \sum_{i=1}^{n} [\mu_i^{5+k} L(b_i h_i) \cdot \{\mu_i L(\mu_i b_i h_i) + L(d_i)\}^2 +$$

$$L(\varepsilon) L(b_i h_i)^2 .$$

$\text{sep}(A) \leq \text{sep}(Q_i)$ if Q_i has more than one distinct root and $b_i \leq 4d_i \leq 4d$. Hence $\lambda/4 \leq \lambda_i$, $\delta/4 \leq \delta_i$, and $4h \geq h_i$. Thus $T \propto \sum_{i=1}^{n} [\mu_i^{5+k} L(16dh) \cdot \{\mu_i L(16\mu_i dh)$

$$+ L(4d)\}^2 + L(\varepsilon) L(16dh)^2] \sim \sum_{i=1}^{n} [\mu_i^{5+k} L(dh) \cdot \{\mu_i L(\mu_i dh)$$

$$+ L(d)\}^2 + L(\varepsilon) L(dh)^2]. \quad \sum_{i=1}^{n} \mu_i \leq 2\mu, \text{ so } T \propto$$

$$\sum_{i=1}^{n} \mu_i [\mu_i^{4+k} L(dh) \cdot \{\mu L(\mu dh) + L(d)\}^2] + \mu L(\varepsilon) L(dh)^2 \propto$$

$$2\mu^{5+k} L(dh) \cdot \{\mu L(\mu dh) + L(d)\}^2 + \mu L(\varepsilon) L(dh)^2 \propto$$

$$\mu^{5+k} L(dh) \{\mu L(\mu dh)\}^2 + \mu L(\varepsilon) L(dh)^2 \propto \mu^{7+k} L(dh) L(\mu dh)^2$$

$$+ \mu L(\varepsilon) L(dh)^2 . \blacksquare$$

ALGORITHM GPMRIR:

R=GPMRIR(P,ε)

Gaussian Polynomial with Arbitrary

Multiplicity Roots, Isolation

and Refinement

P is a univariate Gaussian polynomial of positive

degree. ε is a non-negative rational number. R is the list $(e_1, r_1, \ldots, e_j, r_j)$, $e_1 < e_2 < \ldots < e_j$. Each r_k is a list $(r_{k,1}, \ldots, r_{k,\ell_k})$ where $r_{k,p}$ is the interval representation of a square in the complex plane containing exactly one root $\alpha_{k,p}$ of P. e_k is the multiplicity of each root $\alpha_{k,p}$ of P and $\{\alpha_{k,1}, \ldots, \alpha_{k,m_k}\}$ are all the roots of P of multiplicity e_k. Hence for $b = \mathrm{ldcf}(P)$, $P(z) = b\prod_{k=1}^{j}\{\prod_{p=1}^{m_k}(z - \alpha_{k,p})\}^{e_k}$. If $\varepsilon \neq 0$, then each square containing a root has width $w < \varepsilon$.

Method:

Let $b\prod_{j=1}^{n} A_j^{e_j}$ be the square-free factorization of P. Then GPRIR is applied to each A_j.

Description:

(1) [Initialize; obtain factors and multiplicities.] $R \leftarrow 0$; $\overline{P} \leftarrow$ GPPP(P); $L \leftarrow$ GPSQFF($\overline{P}$); erase $\overline{P}$.

(2) [Isolate and refine roots for each polynomial factor.] DECAP2(m,Q,L); $I \leftarrow$ GPRIR(Q,ε); erase Q;

$R \leftarrow PFL(I,PFA(m,R))$; if $L \neq 0$, go to (2).

(3) [Finish.] $R \leftarrow INV(R)$; return.

Computing Time: $\propto \mu^{7+k} L(dh)L(\mu dh)^2 + \mu L(\varepsilon)L(dh)^2$, where $m = \deg(A)$, $\mu = m + 1$, $d = \max\{|D|_\infty : D|A\}$, $\lambda = \min\{sep(A), 1/d\}$ if A has more than one distinct root and $\lambda = 1/d$ otherwise, $\delta = \lambda$ if $\varepsilon = 0$ and $\delta = \min\{\lambda, \varepsilon\}$ otherwise, $h = \lceil 1/\delta \rceil$, $k = 0$ if modular methods are used or the p.r.s.'s are normal, $k = 1$ otherwise.

Proof: Analogous to that for PMRIR. ∎

6.4 I/O

This section includes algorithms to simplify the conversion of results of previous algorithms from internal form to some easily comprehensible external form. In addition there is an algorithm for input of a standard rectangle.

The finite standard areas used in this report are represented internally by lists (I_1, I_2), where I_1 and I_2 are intervals. The intervals, in turn, are represented internally by lists (r_{11}, r_{12}) and (r_{21}, r_{22}), where the r_{ij} are rational numbers. The various types of standard areas and their representations are as follows.

area	internal forms
point a + ib	((a*,a*),(b*,b*))
line (a,b] × [c]	((a*,b*),(c*,c*))
line [a] × (b,c]	((a*,a*),(b*,c*))
rectangle (a,b] × (c,d]	((a*,b*),(c*,d*))

Here a, b, c, d are finite rational numbers and a*, b*, c*, d* are their respective internal representations.

Rational numbers are classified according to their denominators for use in the output routines. An arbitrary rational number is 0 or has any positive denominator, while a binary rational number $r = r_1/r_2$ has a denominator $r_2 =$

2^k for $k \geq 0$. $r = 0$ is also considered a binary rational.

The output routines in this section are given in pairs, one for rational output of the form $\pm r_1/r_2$ where r_1 and r_2 are integers, and one for decimal output of the form $\pm d_1 \ldots d_j . d_{j+1} \ldots d_k$. The rational output algorithms may be used for an arbitrary rational number. The decimal algorithms, however, may only be used for binary rationals.

Note that GPRBND computes a bound which is a binary rational, and all splits of intervals in the root isolation and refinment algorithms are by 1/2. Hence the vast majority of rational numbers used will in fact be binary rationals. The only non-binary rationals which occur will be as a result of the input of a rectangle with non-binary rational coordinates (for example, to be used with CZREF).

The first pair of algorithms output a single standard area.

ALGORITHM CZSRRW:

CZSRRW(U,L)

Complex Zero System, Single

Rectangle Rational Write

U and L are inputs. U is an I/O unit and L is the internal representation $((r_1,r_2),(s_1,s_2))$ of a rectangle $(r_1,r_2] \times (s_1,s_2]$, where r_1, r_2, s_1, and s_2

are rational numbers. The rectangle is written on unit U in the following form:

REAL INTERVAL

(r_1 as a rational number)

(r_2 as a rational number)

IMAGINARY INTERVAL

(s_1 as a rational number)

(s_2 as a rational number)

Description:

(1) [Initialize.] FIRST2(R,I,L).

(2) [Real part.] Write "REAL INTERVAL" on unit U; FIRST2(R_1,R_2,R); RWRITE(U,R_1); RWRITE(U,R_2).

(3) [Imaginary part.] Write "IMAGINARY INTERVAL" on unit U; FIRST2(I_1,I_2,I); RWRITE(U,I_1); RWRITE(U,I_2);

return.

ALGORITHM CZSRDW:

CZSRDW(U,L)

Complex Zero System, Single

Rectangle Decimal Write

U and L are inputs. U is an I/O unit and L is the internal representation ((r_1,r_2),(s_1,s_2)) of a

rectangle $(r_1, r_2] \times (s_1, s_2]$, where r_1, r_2, s_1, and s_2 are binary rationals. The rectangle is written on unit U in the following form:

REAL INTERVAL

(r_1 as a decimal number)

(r_2 as a decimal number)

IMAGINARY INTERVAL

(s_1 as a decimal number)

(s_2 as a decimal number)

Description:

(1) [Initialize.] FIRST2(R,I,L).

(2) [Real part.] Write "REAL INTERVAL" on unit U; IPRINT(U,R).

(3) [Imaginary part.] Write "IMAGINARY INTERVAL" on unit U; IPRINT(U,I); return.

The output of algorithms such as PRIR is a list of areas. In order to more easily distinguish these areas, the following pair of algorithms number each area and output this number along with the area.

ALGORITHM CZMRRW:

CZMRRW(U,L)

Complex Zero System, Multiple

Rectangle Rational Write

U and L are inputs. U is an I/O unit and $L = (R_1, R_2, \ldots, R_n)$ is a list of the internal representations of n rectangles in the complex plane. The rectangles are written on unit U in the following form:

RECTANGLE i

(R_i in rational form)

Description:

(1) [Initialize.] $i \leftarrow 0$; $\overline{L} \leftarrow L$; go to (3).

(2) [Loop.] $i \leftarrow i + 1$; write "RECTANGLE" and i on unit U; ADV(R,$\overline{L}$); CZSRRW(U,R).

(3) [Check for end.] If $\overline{L} \neq 0$, go to (2); return.

ALGORITHM CZMRDW:

CZMRDW(U,L)

Complex Zero System, Multiple

Rectangle Decimal Write

U and L are inputs. U is an I/O unit and $L = (R_1, R_2, \ldots, R_n)$ is a list of the internal representations of n rectangles in the complex plane.

The rectangle coordinates are binary rationals. The rectangles are written on unit U in the following form:

RECTANGLE i

(R_i in decimal form) .

Description:

(1) [Initialize.] $i \leftarrow 0$; $\overline{L} \leftarrow L$; go to (3).

(2) [Loop.] $i \leftarrow i + 1$; write "RECTANGLE" and i on unit U; ADV(R,$\overline{L}$); CZSRDW(U,R).

(3) [Check for end.] If $\overline{L} \neq 0$, go to (2); return.

The final pair of output algorithms are used with PMRIR and GPMRIR, which generate lists of the form $(e_1, L_1, \ldots, e_n, L_n)$ where L_j is a list of areas and e_j is a Fortran integer, the multiplicity of the roots in the area specified by L_j. Hence the following algorithms output each multiplicity followed by the corresponding list of areas.

ALGORITHM CZMMRW:

CZMMRW(U,L)

Complex Zero System, Multiple Roots with Arbitrary Multiplicity, Rational Write

U and L are inputs. U is an I/O unit. L is a list $(e_1, L_1, \ldots, e_n, L_n)$ where each e_i is a

multiplicity and each L_i is a list of rectangles, each of which contains exactly one root of this multiplicity. The rectangles have arbitrary rational coordinates. The rectangles are written on U in the form

MULTIPLICITY e

ROOT i

REAL INTERVAL

a_1 / b_1

a_2 / b_2

IMAGINARY INTERVAL

c_1 / d_1

c_2 / d_2

where $(a_1/b_1, a_2/b_2] \times (c_1/d_1, c_2/d_2]$ is the i-th rectangle and each numerator and denominator is expressed as a decimal integer.

Description:

(1) [Initialize.] $\overline{L} \leftarrow L$; go to (3).

(2) [Loop.] ADV2(M,C,$\overline{L}$); write "MULTIPLICITY" and M on unit U; CZMRRW(U,C).

(3) [Check for end.] If $\overline{L} \neq 0$, go to (2); return.

ALGORITHM CZMMDW:

CZMMDW(U,L)

Complex Zero System, Multiple Roots with

Arbitrary Multiplicity, Decimal Write

U and L are inputs. U is an I/O unit. L is a list $(e_1, L_1, \ldots, e_n, L_n)$ where each e_i is a multiplicity and each L_i is a list of rectangles, each of which contains exactly one root of this multiplicity. The rectangles have internal representation with each coordinate a binary rational. The rectangles are written on U in the form

MULTIPLICITY e

ROOT i

REAL INTERVAL

a_1

a_2

IMAGINARY INTERVAL

b_1

b_2

where $(a_1, a_2] \times (b_1, b_2]$ is the i-th rectangle and each coordinate is expressed exactly as a decimal fraction.

Description:

(1) [Initialize.] $\overline{L} \leftarrow L$; go to (3).

(2) [Loop.] ADV2(M,C,$\overline{L}$); write "MULTIPLICITY" and M on unit U; CZMRDW(U,C).

(3) [Check for end.] If $\overline{L} \neq 0$, go to (2); return.

In algorithms such as GPNRAR and CZREF, one is working with rectangles which may be input rather than internally generated by some other section of the program. Hence the following algorithm inputs a rectangle specified externally by rational numbers.

ALGORITHM CZRRD:

L=CZRRD(U)

Complex Zero System, Rectangle_Read

U is an I/O unit on which are positioned the coordinates of a rectangle $(r_1, r_2] \times (s_1, s_2]$ in the order r_1, r_2, s_1, s_2, where each coordinate is a rational number. If end-of-file is encountered, L = -1; if an invalid input is encountered, L = -2; otherwise, the numbers are read and converted to the internal representation of a rectangle.

Description:

(1) [Loop and check for valid inputs.] For j ← 1, 2, . . . , 4, do: (I[j] ← RREAD(U); if I[j] < 0, go to (2)); R ← PFL(I[1],PFL(I[2],0));

C ← PFL(I[3],PFL(I[4],0)); L ← PFL(R,PFL(C,0));
return.

(2) [End of file or invalid input.] k ← j - 1; L ← I[j]; if k = 0, return; for n ← 1, 2, . . . , k, do: erase I[n]; return.

6.5 Rational polynomials

Chapter 1 included some discussion of the fact that too often integer systems are not used because the problems concern polynomials with "real" rather than integer coefficients. The quotes are used because in many cases "real" is not meant in the usual mathematical sense, but rather implies coefficients which are represented as floating-point numbers. As was noted there, with present-day machines these floating-point numbers are actually rationals, not reals, and hence such "real" polynomials can be handled by integer systems.

In order to facilitate the use of this integer system for polynomials with rational coefficients, the present section contains algorithms for input and output of rational polynomials, and for converting from a rational polynomial A to an integer polynomial A* which has the same roots as A.

A non-zero rational polynomial A can be expressed in the usual form $A(v) = \sum_{j=0}^{n} a_j v^j$, where $n = \deg(A)$, $a_n \neq 0$, and the a_j are rational numbers. As with integral polynomials in SAC-1, this form can be compressed by excluding terms with zero coefficients. Then $A(v) = \sum_{j=1}^{m} \bar{a}_j v^{e_j}$, where $e_1 < e_2 < \ldots < e_m$, $e_m = \deg(A)$, and

$\bar{a}_j \neq 0,\ 1 \leq j \leq m.$

For input, the external form of this polynomial will be

$$((\bar{a}_m)v{*}{*}e\ (\bar{a}_{m-1})v'{*}{*}e\ \ldots\ (\bar{a}_1)v'{*}{*}e_1)$$

where the following conditions hold:

(1) e_j is an unsigned non-negative β-digit;

(2) v is a variable name of any length consisting of letters and digits, beginning with a letter; v' is either the same character string as v or is null;

(3) $\bar{a}_j$ is an arbitrary non-zero rational number (that is, it is not necessarily in lowest terms or with a positive denominator) expressed as a_{j1}, where the denominator is assumed to be +1, or a_{j1}/a_{j2}; the a_{jk} are non-zero L-integers;

(4) blanks may occur anywhere in the character string; they are ignored;

(5) the polynomial A(x) = 0 is represented as +0.

Rational polynomials output by the system have the form

$$((\bar{a}_m)v{*}{*}e\ (\bar{a}_{m-1})v{*}{*}e\ \ldots\ (\bar{a}_1)v{*}{*}e_1)$$

where $a_j = a_{j1}/a_{j2}$, $a_{j2} > 0$, and $\gcd(a_{j1}, a_{j2}) = 1$. As above, the polynomial A(x) = 0 is represented as +0.

A rational Gaussian polynomial is represented on input and output in the form

$$A_1$$

$$A_2$$

where A_1 and A_2 are rational polynomials and $A = A_1 + iA_2$.

Internally, the non-zero rational polynomial A is represented as the list

$$(v^*, \bar{a}_m^*, e_m^*, \ldots, \bar{a}_1^*, e_1^*)$$

where v^* is the representation of variable v, $\bar{a}_j^*$ is the internal canonical form of rational number $\bar{a}_j$, and e_j^* is the Fortran representation of integer e_j. A = 0 is represented as ().

A rational Gaussian polynomial $A = A_1 + iA_2$ is represented internally as (A_1^*, A_2^*), where A_1^* and A_2^* are the internal representations of rational polynomials A_1 and A_2, respectively. A = 0 is represented as ().

It would have been possible to write the rational polynomial input and output routines each as one large algorithm. Instead, a number of simple routines were written using a modular approach. A heirarchy is set up in

which each module uses the previous ones to input or output some basic part of the polynomial. Essentially, the heirarchy is character, integer, rational number, and finally the polynomial.

The first input algorithm, then, locates and returns the next non-blank character in the input stream. In addition, it divides the characters into classes to facilitate later processing. These classes are as follows.

Character	Class
0,...,9	1
A,...,Z	2
+ -	3
*	4
/	5
(	6
)	7
, . = $	8

ALGORITHM CZNCRD:

CZNCRD(U,I,c,C)

Complex Zero System,

Next Character Read

U and I are inputs. I is modified. c and C are outputs. U is an I/O unit. I is a Fortran integer, $I \geq 0$. The next non-blank character beyond RECORD[I] is located. If end-of-file is encountered, C = -1.

Otherwise, c is the character read, C is its class, and I is its location in RECORD.

Description:

(1) [Locate character.] I ← I + 1; if I > 72, (I ← 1; READ(U,RECORD); if RECORD[1] = -1, (C ← -1; return)); if RECORD[I] = 44, go to (1); c ← RECORD[I].

(2) [Determine class.] If c ≤ 9, (C ← 1; return); if c ≤ 35, (C ← 2; return); if c ≤ 37, (C ← 3; return); if c ≥ 42, (C ← 8; return); C ← c - 34; return.

The integer input algorithm needs a method of incorporating the next digit d in the input stream into the value of the integer obtained so far, say D. The obvious approach is to compute 10·D + d. The following algorithm does this, with the added benefits of performing initialization and saving time by altering the input rather than creating new lists which must later be erased.

ALGORITHM CZIMAD:

CZIMAD(a,b,c)

Complex Zero System, L-Integer

Multiplied by a β-Integer,

Result Added to a β-Integer

a, b, and c are inputs. a is modified. a is an L-integer. b and c are β-integers with abc ≥ 0. a is modified to represent ab + c, an L-integer.

Description:

(1) [Check for zeros.] If a $\neq$ 0 and b $\neq$ 0, go to (2); erase a; a $\leftarrow$ 0; if c $\neq$ 0, a $\leftarrow$ PFA(c,0); return.

(2) [Multiply and add.] t $\leftarrow$ a; a $\leftarrow$ IMADD(t,b,c); return.

Computing Time: $\propto$ L(a).

The following algorithm inputs the next integer in the input stream. It is used by the rational number algorithm for obtaining numerator and denominator, and directly by the polynomial routine for obtaining exponents.

ALGORITHM CZNRD:

CZNRD(U,I,c,C,n)

Complex Zero System, Number Read

U, I, and c are inputs. I and c are modified. C and n are outputs. U is an I/O unit. I is a Fortran integer, 1 $\leq$ I $\leq$ 72. c = RECORD[I] is the first character of an L-integer terminated by a ")", "(", or "/". If end-of-file is encountered, C = -1. If an invalid terminal character is encountered, C = -2. Otherwise, n is the internal representation of the L-integer. c is the terminal character, C is its class, and I is its location in RECORD.

Description:

(1) [Initialize and check signs.] s $\leftarrow$ 1; n $\leftarrow$ 0; if c < 10, go to (2); if c = 37, s $\leftarrow$ -1; CZNCRD(U,I,c,C).

(2) [Incorporate next digit into n.] If C $\neq$ 1, go to (3); CZIMAD(n,10,c); CZNCRD(U,I,c,C); go to (2).

(3) [Check for termination validity.] If C < 8 and C > 4, go to (4); erase n; if C = -1, return; C $\leftarrow$ -2; return.

(4) [Check sign and return.] If s > 0, return; t $\leftarrow$ INEG(n); erase n; n $\leftarrow$ t; return.

The next rational number in the input stream is obtained by the following algorithm. It accepts rational numbers with or without a denominator, using algorithms of Section 2.1 to convert them to SAC-1 internal canonical form.

ALGORITHM CZRNRD:

CZRNRD(U,I,c,C,r)

Complex Zero System, Rational Number Read

U, I, and c are inputs. I and c are modified. C and r are outputs. U is an I/O unit. I is a Fortran integer, 1 $\leq$ I $\leq$ 72. c = RECORD[I] = "(" is the first character of a rational number terminated by a ")". If end-of-file is encountered, C = -1. If an invalid terminal character is encountered, C = -2. Otherwise, r is the internal representation of the number. c is the terminal character, C is its class, and I is its location in RECORD.

Description:

(1) [Obtain and check second character.] CZNCRD(U,I, c,C); if C ≠ 1 and C ≠ 3, go to (5).

(2) [No denominator.] CZNRD(U,I,c,C,r_1); if C = 7, ($r \leftarrow$ RNINT1(r_1); erase r_1; return); if C = 5, go to (3); if C ≠ 6, go to (5); erase r_1; go to (5).

(3) [Check first character of denominator.] CZNCRD(U, I,c,C); if C = 1 or C = 3, go to (4); erase r_1; go to (5).

(4) [Obtain denominator.] CZNRD(U,I,c,C,r_2); if C ≠ 7, (erase r_1; if C ≠ 5 and C ≠ 6, go to (5); erase r_2; go to (5)); $r \leftarrow$ RNINT2(r_1,r_2); erase r_1, r_2; return.

(5) [Error return.] If C = -1, return; C ← -2; return.

Using the previous four algorithms, the next algorithm performs the actual input of a rational polynomial.

ALGORITHM CZRPRD:

P=CZRPRD(U)

Complex Zero System, Rational

Polynomial Read

U is an I/O unit on which is positioned the external representation of a rational polynomial. If end-of-file is encountered, P = -1. If the polynomial

is invalid, P = -2. Otherwise, the polynomial is read and converted to internal form.

Description:

(1) [Initialize.] I ← 72; P ← -1; CZNCRD(U,I,c,C); if C = -1, return; CZNCRD(U,I,c',C'); if C' = -1, return; P ← -2; if c = 36, (if c' ≠ 0, return; P ← 0; return); if C ≠ 6 or C' ≠ 6, return.

(2) [Obtain first coefficient.] CZRNRD(U,I,c,C,r); if C ≠ 7, (P ← C; return); v ← 0.

(3) [Obtain variable.] CZNCRD(U,I,c,C); if C = 4, go to (4); if C < 0, (P ← C: erase v, r; return); v ← PFA(c,v); go to (3).

(4) [Check for valid termination.] CZNCRD(U,I,c,C); if C = 4, go to (5); if C ≠ -1, C ← -2; P ← C; erase v, r; return.

(5) [Process variable name.] v ← INV(v); P ← PFL(r, PFL(PROSYM(v),0)); erase v; go to (8).

(6) [Obtain next coefficient.] CZRNRD(U,I,c,C,r); if C ≠ 7, (erase P; P ← C; return); P ← PFL(r,P).

(7) [Go over variable.] CZNCRD(U,I,c,C); if C < 0, (erase P; P ← C; return); if C ≠ 4, go to (7); CZNCRD(U,I,c,C); if C = 4, go to (8); if C ≠ -1, C ← -2; erase P; P ← C; return.

(8) [Check next exponent.] CZNCRD(U,I,c,C); if C = 1 or C = 3, go to (9); if C ≠ -1, C ← -2; erase P; P ← C; return.

(9) [Obtain exponent.] CZNRD(U,I,c,C,n); if C < 0, (erase P; P ← C; return); if C = 7, (if n = 0, n ← PFA(0,0); if TAIL(n) ≠ 0, (erase P; P ← -2; return); SSUCC(P,n); P ← INV(n); return); if C = 6, (if TAIL(n) ≠ 0, (erase P; P ← -2; return); SSUCC(P, n); P ← n; go to (6)); if C = 5, erase n; if C ≠ -1, C ← -2; erase P; P ← C; return.

The following rational Gaussian polynomial input algorithm simply applies the rational polynomial routine twice.

ALGORITHM CZRGPR:

G=CZRGPR(U)

Complex Zero System, Rational

Gaussian Polynomial Read

U is an I/O unit on which are positioned G_1 and G_2, two univariate rational polynomials representing the real and imaginary parts of a Gaussian polynomial. If an end-of-file is encountered, G = -1. If the polynomials are not valid SAC-1 representations, G = -2. Otherwise, G is the internal representation of the Gaussian polynomial $G = G_1 + iG_2$.

Description:

(1) [Read first polynomial.] G ← 0; G_1 ← CZRPRD(U);

if $G_1 \geq 0$, go to (2); $G \leftarrow G_1$; return.

(2) [Read second polynomial.] $G_2 \leftarrow$ CZRPRD(U);

if $G_2 \geq 0$, go to (3); erase G_1; $G \leftarrow G_2$; return.

(3) [Assemble parts.] If $G_1 \neq 0$ or $G_2 \neq 0$, $G \leftarrow$

PFL(G_1,PFL(G_2,0)); return.

The first algorithm of the output series is used for an arbitrary list of characters.

ALGORITHM CZLSTW:

CZLSTW(U,I,L)

Complex Zero System, List Write

U, I, and L are inputs. I is modified. U is an I/O unit. I is a Fortran integer, $1 \leq I \leq 72$. L is a first-order list, $(\ell_1, \ell_2, \ldots, \ell_n)$, $n \geq 1$. For $1 \leq j \leq n$, RECORD[I + j - 1] is set to ℓ_j. If at any time all 72 elements of RECORD have been filled, then it is written on unit U and the remaining elements of L inserted beginning with RECORD[1]. $I \leftarrow 1 + (I + n - 1 \pmod{72})$, the next available entry in RECORD.

Description:

(1) [Initialize.] $L' \leftarrow L$.

(2) [Loop.] ADV(ℓ,L'); RECORD[I] $\leftarrow \ell$; if I = 72, (I $\leftarrow$ 0; WRITE(U,RECORD)); I $\leftarrow$ I + 1; if $L' \neq 0$, go to (2);

return.

The next algorithm outputs an integer by converting it to a list of characters and then calling the preceding list output algorithm.

ALGORITHM CZNWR:

CZNWR(U,I,n,f)

Complex Zero System, Number Write

U, I, n, and f are inputs. I is modified. U is an I/O unit. I is a Fortran integer, $1 \leq I \leq 72$. n is an L-integer. The character code for n is entered into RECORD beginning with the I-th element. f is a Fortran integer. If f = 1, the sign of n is suppressed. Otherwise, the sign is the first character entered into RECORD. If during the routine all 72 elements of RECORD are filled, then it is written on unit U and the remaining characters of n are entered beginning with RECORD[1]. The value of I returned is the next empty element of RECORD.

Description:

(1)n' ← IBTOD(n); if f = 1, DECAP(s,n'); CZLSTW(U,I, n'); erase n'; return.

The following algorithm is used to output a rational number.

ALGORITHM CZRNWR:

CZRNWR(U,I,r)

Complex Zero System,

Rational Number Write

U, I, and r are inputs. I is modified. U is an I/O unit. I is a Fortran integer, $1 \leq I \leq 72$. r is a rational number, $r \neq 0$. Let $r = r_1/r_2$ and r^*_i be the character code for r_i. Then the characters "r^*_1 / r^*_2" are entered into RECORD beginning with the I-th element. If during the routine all 72 elements of RECORD are filled, then it is written on unit U and the remaining characters are entered beginning with RECORD[1]. The value of I returned is the next empty element of RECORD.

Description:

(1) FIRST2(r_1,r_2,r); CZNWR(U,I,r_1,0); L ← PFA(44, PFA(39,PFA(44,0))); CZLSTW(U,I,L); erase L; CZNWR(U,I,r_2,0); return.

The next algorithm uses the previous three to output a rational polynomial.

ALGORITHM CZRPWR:

CZRPWR(U,P)

Complex Zero System, Rational

Polynomial Write

U and P are inputs. U is an I/O unit. P is a rational polynomial. P is written on unit U.

Description:

(1) [Check for zero.] If P = 0, (PWRITE(U,0); return); v ← FIRST(P); P' ← TAIL(P); I ← 1; p ← PFA(40,0); E ← PFA(0,0); V ← CONC(ITOS(v),PFA(38,PFA(38,0))); V ← PFA(41,V); CZLSTW(U,I,p).

(2) [Output next term.] ADV2(r,e,P'); CZLSTW(U,I,p); CZRNWR(U,I,r); CZLSTW(U,I,V); if e = 0, (CZNWR(U,I, 0,1); go to (3)); ALTER(e,E); CZNWR(U,I,E,1); if P' ≠ 0, go to (2).

(3) [Finish output of characters.] Erase E; ALTER(41,p); CZLSTW(U,I,p); erase p; if I = 1, go to (5); j ← 73 - i; L ← 0.

(4) [Fill RECORD to output last line.] L ← PFA(44,L); j ← j - 1; if j > 0, go to (4); CZLSTW(U,I,L); erase L.

(5) [Finish.] Erase V; return.

As with the input algorithm, the rational Gaussian polynomial output algorithm simply calls the rational polynomial output algorithm twice.

ALGORITHM CZRGPW:

CZRGPW(U,G)

Complex Zero System, Rational

Gaussian Polynomial Write

U and G are inputs. U is an I/O unit. G is a univariate rational Gaussian polynomial. G is written on U in the form

REAL PART

G_1

IMAGINARY PART

G_2

Description:

(1) [Initialize.] $G_1 \leftarrow 0$; $G_2 \leftarrow 0$; if $G \neq 0$,

FIRST2(G_1,G_2,G).

(2) [Write G_1.] Write "REAL PART" on unit U;

CZRPWR(U,G_1).

(3) [Write G_2.] Write "IMAGINARY PART" on unit U;

CZRPWR(U,G_2); return.

The previous algorithms in this section provide for rational polynomial input and output. The other facility required is conversion of the rational polynomials to

integral polynomials which have the same roots. Multiplying a polynomial by a non-zero constant does not change the roots, so the following algorithm can be used as the basis of the conversion.

If the input list of rationals is $(r_{11}/r_{12}, \ldots, r_{n1}/r_{n2})$, this algorithm obtains a corresponding list of integers by multiplying each rational by $\text{lcm}(r_{12}, \ldots, r_{n2})/\text{gcd}(r_{11}, \ldots, r_{n1})$.

ALGORITHM CZRTIC:

B=CZRTIC(R)

Complex Zero System, Rational

to Integer Coefficients

R is a list $(r_1, r_2, \ldots, r_n)$ of non-zero rational numbers. Let $r_i = r_{i1}/r_{i2}$, $\bar{r}_1 = \text{gcd}(r_{11}, r_{21}, \ldots, r_{n1})$, and $\bar{r}_2 = \text{lcm}(r_{12}, r_{22}, \ldots, r_{n2})$. If R is null then B = 0. Otherwise, B is the list $(b_1, b_2, \ldots, b_n)$, where $b_i = (r_{i1}/\bar{r}_1)(\bar{r}_2/r_{i2})$, an L-integer.

Description:

(1) [Initialize.] B ← 0; if R = 0, return; R' ← R; ADV(r,R'); ADV2(r_1,r_2,r); $\bar{r}_1$ ← BORROW(r_1);

$\overline{r}_2 \leftarrow$ BORROW(r_2); go to (5).

(2) [Obtain next number.] ADV(r,R'); ADV2(r_1,r_2,r).

(3) [Compute next gcd.] If PONE($\overline{r}_1$) = 1, go to (4);

$t \leftarrow$ IGCD(r_1,$\overline{r}_1$); erase $\overline{r}_1$; $\overline{r}_1 \leftarrow t$.

(4) [Compute next lcm.] $t \leftarrow$ IGCD($\overline{r}_2$,r_2); if PONE(t) $\neq$ 1,

($t_2 \leftarrow$ IQ(r_2,t); erase t; $\overline{t}_2 \leftarrow \overline{r}_2$; $\overline{r}_2 \leftarrow$ IPROD($\overline{t}_2$,t_2);

erase t_2, $\overline{t}_2$; go to (5)); erase t; $t \leftarrow$ IPROD($\overline{r}_2$,r_2);

erase $\overline{r}_2$; $\overline{r}_2 \leftarrow t$.

(5) [Check for end.] If R' $\neq$ 0, go to (2).

(6) [Obtain absolute values.] $t \leftarrow$ IABSL($\overline{r}_1$); erase $\overline{r}_1$;

$\overline{r}_1 \leftarrow t$; $t \leftarrow$ IABSL($\overline{r}_2$); erase $\overline{r}_2$; $\overline{r}_2 \leftarrow t$; R' $\leftarrow$ R.

(7) [Multiply each rational.] ADV(r,R'); ADV2(r_1,r_2,r);

$t_2 \leftarrow$ IQ($\overline{r}_2$,r_2); $t_1 \leftarrow$ IQ(r_1,$\overline{r}_1$); $b \leftarrow$ IPROD(t_2,t_1);

erase t_1, t_2; B $\leftarrow$ PFL(b,B); if R' $\neq$ 0, go to (7).

(8) [Finish.] B $\leftarrow$ INV(B); erase $\overline{r}_1$, $\overline{r}_2$; return.

Computing Time: $\propto n\, L(r)^2$, where $R = (r_1, \ldots, r_n)$,

$r_i = r_{i1}/r_{i2}$, $r = \max_{\substack{1 \le i \le n \\ 1 \le j \le 2}} \{|r_{ij}|\}$ if $n > 0$; ~ 1 if

$n = 0$.

Proof:

Step	n_i	t_i
1	1	~ 1
2	$n - 1$	~ 1
3	$n - 1$	$\underline{\propto} L(r)^2$
4	$n - 1$	$\underline{\propto} nL(r)^2$
5	n	~ 1
6	1	$\underline{\propto} nL(r)$
7	n	$\underline{\propto} nL(r)^2$
8	1	$\sim n$

The times follow immediately from noting that $L(\overline{r}_1) \leq L(r)$ and $L(\overline{r}_2) \leq nL(r)$. ∎

The following theorem proves that if the rational numbers in the input to CZRTIC were in lowest terms then the integers computed are relatively prime.

Theorem 6.5.1: Let B and R be as described in algorithm CZRTIC. If $\gcd(r_{i1}, r_{i2}) = 1$ for $1 \leq i \leq n$ then $\gcd(b_1, \ldots, b_n) = 1$.

Proof: Consider the prime factorization of r_{i1} and

r_{i2} for primes $p_j > 1$:

$$r_{i1} = \prod_{j \geq 1} p_j^{e_{ij}};$$

$$r_{i2} = \prod_{j \geq 1} p_j^{f_{ij}}.$$

Let $\overline{e}_j = \min_{1 \leq i \leq n} \{e_{ij}\}$ and $\overline{f}_j = \max_{1 \leq i \leq n} \{f_{ij}\}$. Then

$\overline{r}_1 = \prod_{j \geq 1} p_j^{\overline{e}_j}$ and $\overline{r}_2 = \prod_{j \geq 1} p_j^{\overline{f}_j}$. Hence $b_i = \prod_{j \geq 1} p_j^{g_{ij}}$

where $g_{ij} = e_{ij} - \overline{e}_j + \overline{f}_j - f_{ij}$. If $\overline{e}_j \neq 0$, then $e_{ij} > 0$

for all i. But $\min\{e_{ij}, f_{ij}\} = 0$ for all i since $\gcd\{r_{i1},$

$r_{i2}\} = 1$. So then $f_{ij} = 0$ for all i, and hence $\overline{f}_j = 0$.

Thus either $\overline{e}_j = e_{ij} = 0$ for all i or $\overline{f}_j = f_{ij} = 0$ for all

i. In either case, $\min_{1 \leq i \leq n} \{g_{ij}\} = 0$. But if $b = \gcd(b_1, \ldots, b_n) = \prod_{j \geq 1} p_j^{g_j}$ then $g_j = \min_{1 \leq i \leq n} \{g_{ij}\}$ $= 0$ and $b = 1$. ∎

The following algorithm uses CZRTIC to compute an integral polynomial with the same roots as the input rational polynomial. Note that CZRPRD reduces all rational coefficients to lowest terms, and hence the result of

applying this algorithm to a polynomial obtained from CZRPRD will be a primitive polynomial.

ALGORITHM CZRIP:

B=CZRIP(A)

Complex Zero System, Rational

to Integral Polynomial

A is a univariate rational polynomial. B is the unique primitive integral polynomial which is similar to A and has sign(B) = +1.

Description:

(1) [Initialize.] $B \leftarrow 0$; if $A = 0$, return; $A_1 \leftarrow$ TAIL(A); $A_2 \leftarrow A_1$; $L \leftarrow 0$.

(2) [Obtain coefficients.] ADV2(C,E,A_1); $L \leftarrow$ PFL(BORROW(C),L); if $A_1 \neq 0$, go to (2).

(3) [Compute list of integral coefficients.] $M \leftarrow$ CZRTIC(L); erase L; $M \leftarrow$ INV(M); $B \leftarrow$ PFL(PVBL(A),0).

(4) [Construct B.] ADV2(C,E,A_2); DECAP($\overline{C}$,M); $B \leftarrow$ PFA(E,PFL($\overline{C}$,B)); if $A_2 \neq 0$, go to (4).

(5) [Finish.] $\overline{B} \leftarrow$ INV(B); $B \leftarrow$ PABS($\overline{B}$); erase $\overline{B}$; return.

Computing Time: $\propto \mu^2 L(d)^2$, where $m = \deg(A)$, $\mu = m + 1$, and $d = |A|_\infty$.

Proof: Steps (2) and (4) are executed at most μ times

and $t_2, t_4 \sim 1$. $t_{CZRTIC}(L) \leq \mu^2 L(d)^2$. $t_{INV}(M) \leq \mu$.

$t_{INV}(B) \leq \mu$. $t_{PABS}(\overline{B}) \leq \mu L(d)$. ∎

The last algorithm of this section uses CZRTIC to convert a rational Gaussian polynomial to a Gaussian polynomial with the same roots.

There is an interesting aspect to note about this algorithm. Suppose $A(z) = \sum_{j=0}^{n} (a_j + ib_j) z^j$ is a Gaussian polynomial. Define the integer content of A to be $\gcd(a_0, b_0, a_1, b_1, \ldots, a_n, b_n)$. Application of CZRTIC does yield a Gaussian polynomial whose integer content is one. However, this does not imply that its content is one. A simple example is $(1 + 3i)z + (3 + 4i)$, whose integer content is one but whose content is $2 + i$. This is why the following algorithm must apply GPPP in the last step.

ALGORITHM CZGRIP:

B=CZGRIP(A)

Complex Zero System, Gaussian Rational

to Integral Polynomial

A is a univariate Gaussian rational polynomial. B is the unique primitive Gaussian polynomial which is similar to A and has ldcf(B) in the first quadrant or

on the non-negative real axis.

Description:

(1) [Initialize.] B ← 0; if A = 0, return; FIRST2(A , A ,A); if A ≠ 0, A ← TAIL(A); if A ≠ 0, A ← TAIL(A); L ← 0; A* ← A ; A* ← A .

(2) [Obtain coefficients from A .] If A* = 0, go to (3); ADV2(C,E,A*); L ← PFL(BORROW(C),L); go to (2).

(3) [Obtain coefficients from A .] If A* = 0, go to (4); ADV2(C,E,A*); L ← PFL(BORROW(C),L); go to (3).

(4) [Compute list of integral coefficients.] M ← CZRTIC(L); erase L; M ← INV(M); B ← 0; B ← 0.

(5) [Compute B .] If A = 0, go to (6); ADV2(C,E,A); DECAP($\overline{C}$,M); B ← PFA(E,PFL($\overline{C}$,B)); go to (5).

(6) [Compute B .] If A = 0, go to (7); ADV2(C,E,A); DECAP($\overline{C}$,M); B ← PFA(E,PFL($\overline{C}$,B)); go to (6).

(7) [Construct B.] If B ≠ 0, B ← PFL(GPVBL(A), INV(B)); if B ≠ 0, B ← PFL(GPVBL(A),INV(B));

$B \leftarrow PFL(B_1, PFL(B_2, 0))$; $\overline{B} \leftarrow GPPP(B)$; erase B; $B \leftarrow \overline{B}$;

return.

Computing Time: $\underline{\propto} \mu^2 L(d)^2$, where $m = \deg(A)$, $\mu = m + 1$, and $d = |A|_\infty$.

Proof: Steps (2), (3), (5), and (6) are executed at most μ times and $t_2, t_3, t_5, t_6 \sim 1$. $t_{CZRTIC}(L) \underline{\propto} 2\mu^2 L(d)^2 \sim \mu^2 L(d)^2$. $t_{INV}(M) \underline{\propto} 2\mu \sim \mu$. $t_{INV}(B_1)$, $t_{INV}(B_2) \sim \mu$. $t_{GPPP}(B) \underline{\propto} \mu L(d)^2$. ∎

CHAPTER 7: CONCLUSION

7.1 Deck structure and sample run

As an aid in system application, this section presents a complete sample run of the polynomial root isolation and refinement routine.

The sample program is shown on the following two pages. It should serve as a guideline for the general deck structure of a program using routines from this system. Note, however, that the actual parameter values such as BETA will depend on the particular installation at which the program is run.

AVAIL is the location of the available space list and STAK the location of the pushdown stack. These values are initialized by subroutine BEGIN. RECORD is used by the I/O routines, and need not be initialized. SYMLST is used for polynomial variable names, and is initialized to zero. BETA is the modulus of the integer arithmetic representations. PRIME is the list of primes, each of which must be greater than 2^{PEXP}.

TR6 is the block of particular interest to users of this system. METHOD is set to zero for computing the p.r.s.'s with integer methods, and it is set to one for computing them with modular methods. PRINV is the list of primes and inverses used by the modular methods.

```
      INTEGER A,ABAR,ACC,ARRAY,B,BBAR,C,OUT,OUT1,OUT2,SPACE
      INTEGER AVAIL,BETA,PEXP,PRIME,PRINV,RECORD,STAK,SYMLST,ZETA
      INTEGER CZRIP,CZRPRD,GENPR,GLPINV,PPROD,PRIR,RNUM
      COMMON /TR1/ AVAIL,STAK,RECORD(72)
      COMMON /TR2/ SYMLST
      COMMON /TR3/ BETA
      COMMON /TR4/ PRIME,PEXP
      COMMON /TR6/ METHOD,PRINV
      COMMON /TR8/ ZETA,KAPPA
      DIMENSION SPACE(10000)                                    -- line 10
      DIMENSION ARRAY(1275)
      CALL BEGIN(SPACE,10000)
      SYMLST=0
      BETA=2**33
      ZETA=33
      KAPPA=7
      PRIME=GENPR(ARRAY,400,2**31+1)
      PRINV=GLPINV(PRIME)
      PEXP=31
      METHOD=0                                                  -- line 20
      IN=2HIN
      CALL IFILE(20,IN)
      OUT=3HOUT
      CALL OFILE(21,OUT)
      IN2=20
      OUT1=5
      OUT2=21
      LEN=LENGTH(AVAIL)
      WRITE(OUT1,100) LEN
      WRITE(OUT2,100) LEN                                       -- line 30
100   FORMAT(I10)
      WRITE (OUT1,200)
      WRITE (OUT2,200)
200   FORMAT (21H RATIONAL POLYNOMIALS)
      A=CZRPRD(IN2)
      CALL CZRPWR(OUT1,A)
```

```
      CALL CZRPWR(OUT2,A)
      ABAR=CZRIP(A)
      CALL ERASE(A)
      DO 300 I=1,3                                          -- line 40
      B=CZRPRD(IN2)
      CALL CZRPWR(OUT1,B)
      CALL CZRPWR(OUT2,B)
      BBAR=CZRIP(B)
      CALL ERASE(B)
      C=PPROD(ABAR,BBAR)
      CALL PERASE(ABAR)
      CALL PERASE(BBAR)
300   ABAR=C
      WRITE (OUT1,400)                                      --.line 50
      WRITE (OUT2,400)
400   FORMAT (36H INTEGRAL POLYNOMIAL WITH SAME ROOTS)
      CALL PWRITE(OUT1,ABAR)
      CALL PWRITE(OUT2,ABAR)
      ACC=RNUM(1,10)
      WRITE (OUT1,500)
      WRITE (OUT2,500)
500   FORMAT (8H EPSILON)
      CALL RWRITE(OUT1,ACC)
      CALL RWRITE(OUT2,ACC)                                 -- line 60
      L=PRIR(ABAR,ACC)
      CALL RERASE(ACC)
      CALL CZMRDW(OUT1,L)
      CALL CZMRDW(OUT2,L)
      CALL ERASE(L)
      CALL PERASE(ABAR)
      CALL ERASE(SYMLST)
      LEN=LENGTH(AVAIL)
      WRITE (OUT1,100) LEN
      WRITE (OUT2,100) LEN                                  -- line 70
      STOP
      END
```

ZETA and KAPPA are used for Gaussian integer division and for the new version of PGCDCF presented in this thesis.

Lines 21 through 27 of the program set up the I/O. This program was run on a PDP-10 from a remote terminal with video screen display. Hence there are always two outputs, one to the terminal for monitoring the computation and one to the disk for later printing or other hard copy production.

The next three lines compute and write the length of available space. The same thing is done at the end of the program, and comparison of the values will indicate if any cells were lost.

The input to the program consists of four simple rational polynomials with obvious roots. In lines 35 through 49 these polynomials are read, converted to corresponding integer polynomials, and multiplied together. The result is an integral polynomial which is large enough in degree and coefficients to provide an interesting example, and at the same time one which has known roots for checking the system output.

The accuracy to which the roots are refined is 1/10. The internal representation for this rational number is created using RNUM in line 55. For more flexibility, an arbitrary value could have been read with RREAD.

Recall that the root isolation and refinement routines

produce lists whose components are binary rationals. Hence either the rational or decimal output routines can be used. In this example, decimal is selected. CZMRRW could be substituted in lines 63 and 64 to obtain rational output.

The actual program output for one run is shown on the following two pages.

```
      3320
 RATIONAL POLYNOMIALS
((+1 / +1)X**1(-1 / +2)X**Ø)
((+1 / +1)X**1(-1 / +6)X**Ø)
((+1 / +1)X**2(+2 / +1)X**1(+1Ø / +9)X**Ø)
((+1 / +5)X**1(-1 / +1)X**Ø)
 INTEGRAL POLYNOMIAL WITH SAME ROOTS
(+1Ø8X**5-396X**4-735X**3+13X**2+32ØX**1-5ØX**Ø)
 EPSILON
+1   /   +1Ø
      RECTANGLE    1
   REAL INTERVAL
+5
+5
   IMAGINARY INTERVAL
+Ø
+Ø
      RECTANGLE    2
   REAL INTERVAL
+Ø,5
+Ø,5
   IMAGINARY INTERVAL
+Ø
+Ø
      RECTANGLE    3
   REAL INTERVAL
+Ø,125
+Ø,1875
   IMAGINARY INTERVAL
+Ø
+Ø
      RECTANGLE    4
   REAL INTERVAL
```

```
-1 .
-1
   IMAGINARY INTERVAL
-Ø,375
-Ø,3125
      RECTANGLE    5
   REAL INTERVAL
-1
-1
   IMAGINARY INTERVAL
+Ø,3125
+Ø,375
      3320
```

7.2 Empirical data

This section presents empirical data to augment previous theoretical discussions and assist in application of the system. The data includes selected subprogram computing times and core requirements, and a study of polynomial coefficients in a sample run of the Gaussian polynomial root isolation and refinement routine.

Computing times will of course vary greatly depending on machine and installation. The times given here were obtained on a PDP-10 at an installation primarily concerned with interactive programming and not large, long-running Fortran batch jobs. Hence there was considerable swap time overhead. In addition, the SAC-1 implementation had only the primitives written in assembly language. Comparison to a Univac 1108 doing mainly batch scientific computing with many of the SAC-1 list processing routines in assembly language indicated a virtually constant time reduction factor of 13 to 1.

Hence the times are useful relative indicators. However, it should be realized that they are not in any sense minimum, and that in other environments much larger cases could be run in reasonable time.

Core requirements listed include the routine itself, all of the other routines it causes to be loaded, and approximately 5K of system overhead accompanying every

Fortran run. It does not include data or code for the main program, the major portion of which will probably be allocation of available space. As a guideline for available space, no example in this report required more than 3200 cells.

Some routines in this system display fairly consistent relationships between certain parameters of the inputs and the computing times. For example, applying GPROT to a wide spectrum of randomly generated polynomials should give computing times which are a consistent function of degree and infinity norm.

With the major system routines, however, times for inputs with the same defining characteristics can vary greatly. An immediate example is root isolation, where polynomials with the same degree and infinity norm can have vastly different root separation and hence computing time.

Therefore, this section presents illustrative examples of the computing times of the major routines, rather than timing tables based on randomly generated polynomials with incremental defining characteristics.

Norms are given in decimal digits (dd) and times in seconds (s).

GPRIR: 24K

Isolation of the roots of $\sum_{j=0}^{n} (z + i)^j$

n	time (s)
1	.100
2	6.53
4	30.9
6	115.
10	365.
12	643.

Isolation of the roots of $\sum_{j=0}^{n} (z + i)^j$ and refinement to less than .01

n	time (s)
1	9.12
2	27.4
4	130.
6	363.

PRIR: 26K

Isolation of the roots of $\sum_{j=0}^{n} x^j$

n	time (s)
1	.266
2	1.52
4	5.42
6	41.2
10	138.
12	220.

Isolation of the roots of $\sum_{j=0}^{n} x^j$ and

refinement to less than .01

n	time (s)
1	.266
2	11.7
4	64.0
6	181.

GPMRIR: 27K

Isolation of the roots of $\prod_{j=1}^{n} (x^2 - 2jx + 2j^2)^j$

n	time (s)
1	5.42
2	13.0
3	22.0
4	34.7
5	52.6
6	84.8

PMRIR: 28K

Isolation of the roots of $\prod_{j=1}^{n} (x - j - ij)^j$

n	time (s)
1	4.48
2	13.5
3	27.8
4	46.1
5	80.6
6	136.

CZREFA: 22K

Successive refinements of one root of the polynomial $\sum_{j=0}^{n} x^j$, initial rectangle width 1.0

refinement	time, n=4 (s)	time, n=6 (s)
1	12.0	27.5
2	15.8	25.6
3	11.4	46.2
4	13.2	43.5
5	24.4	41.9

GPNRAR: 22K

Number of roots in a rectangle

deg(G) = 5, $|G|_\infty$ = 3 dd, time = 15.1 s

deg(G) = 13, $|G|_\infty$ = 9 dd, time = 69.8 s

GPNRIC: 22K

Number of roots in a circle

deg(G) = 5, $|G|_\infty$ = 9 dd, time = 8.05 s

deg(G) = 13, $|G|_\infty$ = 26 dd, time = 39.4 s

The second part of this section is a study of the coefficients of polynomials comprising certain p.r.s.'s computed during a run of GPRIR. Coefficient size is of interest because it is a primary factor in determining available space requirements and computing time.

The polynomial used was $x^5 + (-4 + i)x^4 + (6 - i)x^3 + (-16 - i)x^2 + (41 - 15i)x - 60$. This polynomial has roots at $-1 \pm 2i$, $+3$, $1 + i$, and $2 - 2i$. Setting $\varepsilon = 1/3$ resulted in seven iterations of the main loop of GPRIR, four to isolate the roots and three to further refine them.

In each of these iterations, one p.r.s. stood out as having maximum coefficient sizes. (All p.r.s.'s were normal so degree sequences were the same.) These selected p.r.s.'s are presented in the following tables. P_1, P_2, P_3, P_4, P_5, P_6 is the p.r.s., with polynomial $P_j = \sum_{k=0}^{6-j} p_k x^k$. Coefficient sizes are given in decimal digits (dd). Note that the successive reductions of horizontal and vertical strip widths during the iterations were 32 to 16, 16 to 8, 8 to 4, 4 to 2, 2 to 1, 1 to 1/2, and 1/2 to 1/4 .

Pass	P_j	p_5 (dd)	p_4 (dd)	p_3 (dd)	p_2 (dd)	p_1 (dd)	p_0 (dd)
1	1	1	4	6	9	11	13
	2		3	6	8	10	12
	3			8	11	13	15
	4				17	20	22
	5					36	39
	6						51
2	1	1	1	3	3	5	4
	2		2	2	4	4	5
	3			7	6	8	7
	4				14	14	15
	5					30	29
	6						38
3	1	1	1	3	4	3	3
	2		2	2	3	3	4
	3			5	6	6	6
	4				12	12	12
	5					22	21
	6						27
4	1	1	1	2	2	3	2
	2		2	1	3	2	3
	3			5	4	5	4

4	4				11	10	11
	5					17	16
	6						20
5	1	1	1	2	3	3	3
	2		2	2	3	3	3
	3			5	5	5	5
	4				11	11	11
	5					18	18
	6						22
6	1	2	3	4	4	4	4
	2		3	4	4	4	4
	3			8	9	9	9
	4				13	13	13
	5					17	17
	6						16
7	1	4	4	5	5	6	5
	2		5	4	5	5	6
	3			12	12	13	12
	4				18	16	18
	5					29	28
	6						36

7.3 Closing remarks

This section presents brief discussions of two subjects--empirical and theoretical computing times; modular and integer methods.

The theoretical computing times of the main algorithms include terms in which the degree of the input polynomial is raised to a high power. Investigation of the empirical data, however, indicates much smaller exponents for these terms.

One important reason for the discrepancy is that the derivations of the theoretical times involve compounding several upper bounds on polynomial norms.

For example, suppose polynomial A_2 is obtained from polynomial A_1 by some operation such as translation. Let $A_2 = f_1(A_1)$. Then a bound b_2 on $a_2 = |A_2|_\infty$ is computed from A_1, say $b_2 = g_1(A_1)$. By definition, $a_2 \leq b_2$, but what else can be said? For most of the basic algorithms in this thesis, b_2 is a tight bound but one which is only rarely attained. That is, for certain inputs a_2 could be close to or equal b_2; however, in the majority of cases a_2 will be considerably less than b_2.

Now suppose several successive operations are performed, with $A_{j+1} = f_j(f_{j-1}(\ldots f_1(A_1) \ldots))$. Then $b_{j+1} = g_j(g_{j-1}(\ldots g_1(A_1) \ldots))$, and it is clear how the situation discussed in the preceding paragraph compounds itself. Thus in most cases b_{j+1} will be much larger than $|A_{j+1}|_\infty$. Correspondingly, the empirical computing times will usually follow curves described by functions with much smaller exponents than in the theoretical expressions.

Another factor in the theoretical computing times for root isolation and refinement is minimum root separation. One can expect the development of better bounds for this separation. In any case, however, it is likely that the vast majority of polynomials will have far larger separations than the computed minimums.

As an example of the differences between empirical and theoretical, the computing time of GPRIR for a polynomial of degree m has the term $(m + 1)^{11}$. So far, all empirical evidence indicates that the term is $(m + 1)^2$ to $(m + 1)^3$.

A second concern is the decision of whether to use modular or integer methods when applying the system. The theoretical times for modular and integer methods are the

same, unless the p.r.s.'s are not normal; in such cases, the times for integer methods increase. Empirical evidence, however, has so far indicated the integer methods to be superior in all cases.

It must be noted, of course, that these examples are fairly small, a limitation imposed by the speed of the system on which the test cases were run. Further study with much larger degrees and coefficients may alter the present indications.

In addition, further research will be done on obtaining better bounds for the coefficients of the polynomials in the p.r.s.'s . This would in turn lead to better estimates for the number of primes required, and hence possibly improve the times for the modular methods by a significant amount.

REFERENCES

[BRW71] Brown, W. S., On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors, JACM, Vol. 18, No. 4 (Oct., 1971), pp. 478-504.

[BRW71b] Brown, W. S., and J. F. Traub, On Euclid's Algorithm and the Theory of Subresultants, JACM, Vol. 18, No. 4 (Oct., 1971), pp. 505-514.

[CAB73] Caviness, B. F., University of Wisconsin Computing Center Technical Report, in preparation.

[COG67] Collins, G. E., The SAC-1 List Processing System, University of Wisconsin Computing Center Technical Report, July, 1967.

[COG68] Collins, G. E., and J. R. Pinkert, The Revised SAC-1 Integer Arithmetic System, University of Wisconsin Computing Center Technical Report No. 9, Nov., 1968.

[COG68b] Collins, G. E., The SAC-1 Polynomial System, University of Wisconsin Computing Center Technical Report No. 2, March, 1968.

[COG68c] Collins, G. E., The SAC-1 Rational Function System, University of Wisconsin Computing Center Technical Reference No. 8, July, 1968.

[COG69] Collins, G. E., L. E. Heindel, E. Horowitz, M. T. McClellan, and D. R. Musser, The SAC-1 Modular Arithmetic System, University of Wisconsin Computing Center Technical Reference No. 10, June, 1969.

[COG69b] Collins, G. E., Computing Time Analyses for Some Arithmetic and Algebraic Algorithms, Proceedings of the 1968 Summer Institute on Symbolic Mathematical Computation (Robert G. Tobey, ed.), IBM Federal Systems Center, June, 1968, pp. 195-232.

[COG71] Collins, G. E., The Calculation of Multivariate Polynomial Resultants, JACM, Vol. 18, No. 4 (Oct., 1971), pp. 515-532.

[COG72] Collins, G. E., The SAC-1 Polynomial GCD and Resultant System, University of Wisconsin Computing Center Technical Report No. 27, Feb., 1972.

[COG72b] Collins, G. E., Computer Algebra of Polynomials and Rational Functions, to appear in the American Mathematical Monthly.

[COG73] Collins, G. E., The Computing Time of the Euclidean Algorithm, Stanford Artificial Intelligence Laboratory Memo AIM-187, January, 1973.

[COG73b] Collins, G. E., and E. Horowitz, The Minimum Root Separation of a Polynomial, Stanford Computer Science Department Report No. STAN-CS-73-345, April, 1973.

[HEL70] Heindel, L. E., Algorithms for Exact Polynomial Root Calculation, Ph.D. Thesis, Computer Sciences Department, Univeristy of Wisconsin, 1970.

[HEL70b] Collins, G. E., and L. E. Heindel, The SAC-1 Polynomial Real Zero System, University of Wisconsin Computing Center Technical Report No. 18, August, 1970.

[HEL71] Heindel, L. E., Integer Arithmetic Algorithms for Polynomial Real Zero Determination, JACM, Vol. 18, No. 4 (Oct., 1971), pp. 535-548.

[KND68] Knuth, D. E., The Art of Computer Programming, Vol. I: Fundamental Algorithms, Addison-Wesley Publishing Co., Reading, Mass., 1968.

[KND69] Knuth, D. E., The Art of Computer Programming, Vol. II: Seminumerical Algorithms, Addison-Wesley Publishing Co., Reading, Mass., 1969.

[MAM66] Marden, M., The Geometry of the Zeros of a Polynomial in a Complex Variable, American Mathematical Society, Providence, R. I., 1966.

[MUD71] Musser, D. R., Algorithms for Polynomial Factorization, University of Wisconsin Computer Sciences Department Technical Report No. 134 (Ph.D. Thesis), Sept., 1971.

[RUC73] Rubald, C. M., University of Wisconsin Computer Sciences Department Ph.D. Thesis, in preparation.

[WIH62] Wilf, H. S., Mathematics for the Physical Sciences, John Wiley and Sons, Inc., New York, N. Y., 1962.

INDEX OF ALGORITHMS