Interval Computations
No 4, 1993

Precise Zeros of Analytic Functions Using Interval Arithmetic

Mark J. Schaefer

An interval arithmetic algorithm for the computation of the zeros of an analytic function inside a given rectangle is presented. It is based on the argument principle in the set of complex numbers **C**, is guaranteed to converge, and delivers its answers to a prespecified accuracy. The precision of computation is varied dynamically to maximize efficiency.

Вычисление точных нулей аналитических функций с помощью интервальной арифметики

М. Й. Шефер

Предлагается алгоритм на базе интервальной арифметики, вычисляющий нули аналитической функции на заданном прямоугольнике. Данный метод, основанный на принципе аргумента на множестве комплексных чисел **C**, гарантированно сходится и дает результаты с заранее заданной точностью. Разрядность вычислений может динамически изменяться с целью достижения максимальной эффективности.

1 Introduction

We develop a practical algorithm for the computation of all zeros of a function f inside a given rectangle, to an accuracy requested by the user at the start of the program. The function f must be analytic inside the rectangle and on its boundary, and must not have any zeros on the boundary. Furthermore, we assume that values of f and its derivative can be computed to any desired (finite) accuracy, which for instance holds true when f is an *analytic elementary* function. By this we mean a function expressed using a finite number of the binary operations of addition, subtraction, multiplication, and division, the unary operations defined by unary minus, the exponential function, and the trigonometric and hyperbolic functions, and basic terms consisting of complex constants and the identity term z (cf. [1], p. 153).

It is well known that interval methods are generally not capable of identifying a zero of multiple order as such, regardless of the precision of computation. For this reason, a zero of order m is printed m times and hence produces the same result as a tight cluster of m first-order zeros whose printed coordinates at the requested accuracy are identical. Such a cluster would eventually break up if the desired accuracy were chosen sufficiently large, but for a zero of multiple order we can never verify multiplicity.

The core of the algorithm consists of a routine for determining the number of zeros of f (counting multiplicities) inside a rectangle and is based on the argument principle obeyed by analytic functions. The evaluation of f at intervals along the boundary of such a rectangle plays a crucial part in this routine. We test a rectangle for zeros. If it is known not to contain any zeros, it is discarded. Otherwise, we bisect it and put the subrectangles on a list for later processing. Our general strategy has two major characteristics. First, it avoids non-termination from cases where the number of zeros cannot possibly be determined, which happens if a zero lies directly on the boundary of a rectangle. Second, it sets the precision sufficiently high to avoid the accumulation of too many rectangles for which the number of contained zeros is not determined due to a lack of precision, since this would jeopardize convergence of the algorithm. A detailed description of the algorithm follows in Sections 2 and 3.

The algorithm is designed for a variable-precision interval arithmetic whose precision of computation can be adjusted dynamically. In Section 4 we prove convergence of the algorithm, assuming that it is carried out using

this kind of arithmetic and that memory constraints can be ignored. Range arithmetic (see [1] and [2]) belongs to this category of arithmetic and was used to implement the algorithm. We mention in passing that our recent implementation of range arithmetic [2] assumes that programs based on it are written in the programming language C++. Section 5 concludes with some numerical examples used to test the algorithm in several different ways.

The general approach here is similar to that of a bisection algorithm presented by G. Collins and W. Krandick for infallible polynomial complex root isolation [6]. Both algorithms compute winding numbers to test rectangles for enclosed zeros of a function f or polynomial p, but their approach is based on exact methods from computer algebra and cannot easily be extended to analytic functions. Also related to our work is an older paper by P. Henrici and I. Gargantini for the simultaneous approximation of all zeros of a polynomial [7], tested in floating-point arithmetic but analyzed assuming the use of exact arithmetic.

2 Description of the basic algorithm

We first describe a basic version of the algorithm, followed in the next section by three enhancements to make it more efficient. The extent of these improvements will be demonstrated experimentally in Section 5.

Initially, the algorithm starts with just one rectangle R_S which is entered by the user and referred to below as the *starting rectangle*. We first attempt to obtain the number of zeros contained in R_S following a procedure described below. It is clear, however, that this attempt will fail if a zero of f lies on the boundary of R_S or even just very close to it; a sufficiently high precision setting would resolve the latter case, but we do not usually know which case prevails. In the Introduction we assumed that f has no zeros on the boundary of R_S, but from a practical point of view, it is certainly possible that the user has inadvertently entered a starting rectangle which violates this assumption. The starting rectangle is special in the sense that the algorithm cannot proceed in a sound way unless it has verified that no zeros of f lie on the boundary of R_S, since in part the problem is to locate only zeros inside this rectangle. Therefore, if we fail to determine the number of zeros inside R_S, the user is given a choice between trying again in higher precision or else reentering a different rectangle.

We permit the coordinates of R_S to be entered in the form of arithmetic expressions $\big($such as $\sqrt{2} + i * \cos(\pi/8)\big)$, which means that their interval representations may have positive width. Of course these widths decrease as the precision at which the associated expressions are evaluated increases. Therefore, a renewed attempt to compute the number of zeros inside R_S at higher precision always starts with a reevaluation of such expressions. Once the count for the number of zeros in R_S is successful, the coordinates of the two corner points used to represent R_S are assigned the midpoints of their respective interval values. All subsequent rectangles will also have *exact* coordinates, by which we mean intervals of width zero. The bisection of rectangles is done in exact arithmetic to obtain well-defined subrectangles independent of the current precision.

The most important component of the algorithm is the routine that determines the number of zeros contained inside a rectangle R and is based on the well known argument principle, which for completeness' sake is repeated here:

Argument Principle. Let S be a simple closed contour described in the positive sense, and let f be a function which is analytic inside and on S. Also, let f have no zeros on S. Then

$$\frac{1}{2\pi}\Delta_S \arg f(z) = N$$

where N is the number of zeros of f, counting multiplicities, interior to S.

The term $\frac{1}{2\pi}\Delta_S \arg f(z)$ represents the number of times the point $f(z)$ winds around the origin in the image plane as z follows S once in the positive direction, and is often called the *winding number* of f with respect to S. For a proof of this theorem (and more general versions of the argument principle which take into account poles of f) see any standard text on complex variables, such as [5].

The precision of computation is not varied during the execution of the zero counting routine, but only in between separate calls, according to a scheme explained later in this section. The routine may return unsuccessfully, or it may report a positive number of zeros in R, or no zeros in R. In the latter case, we may safely discard R. Otherwise, we bisect R and put the resulting subrectangles on a list where they await further processing. To better explain the details of the zero counting routine, we will illustrate the individual steps with the following simple example: $f(z) = z^2 - (1+2i)z + i$

and R defined by the four corner points: $A = -i$, $B = 2 - i$, $C = 2 + i$, and $D = i$. The function f has one zero inside R.

Our method begins by computing (interval) values of f at the corners of R and classifies each value according to its location: entirely within one of four quadrants, overlapping one of four half-axes (but not zero), or overlapping zero. In general, nine possibilities exist for each value, the last of which causes an unsuccessful return to the calling function because a function value overlapping zero prevents us from resolving the winding behavior of f with respect to R. An unsuccessful return will also result in the rare event that the computation of a value itself fails. In Section 4 we shall prove that this cannot occur provided the precision is sufficiently high. In our example, $f(A) = -3 + 2i$ is contained in the interior of the second quadrant, $f(B) = -1 - 6i$ in the interior of the third quadrant, and $f(C) = 3$ and $f(D) = 1$ overlap the positive real axis.

Next, for each side L of R we consider the relative locations of the values of f at L's two endpoints to determine if there exists a rectangle covering these values with sides parallel to the coordinate axes which does not contain the origin. In the example, this is true only for sides AB and CD. If no such rectangle exists, we cannot hope to enclose the image $f(L)$ in a rectangle not containing zero; this is a consequence of inclusion monotonicity of interval arithmetic. On the other hand, if such a rectangle exists, an attempt is made to compute a rectangle C_L enclosing $f(L)$, using the mean value form of interval extensions:

$$C_L = f\big(\mathrm{mid}(L)\big) + f_I'(L)\big(L - \mathrm{mid}(L)\big) \ .$$

Here $\mathrm{mid}(L)$ refers to the midpoint of L, and f_I' is an interval extension of f'. It is interesting to note that although the mean-value theorem does not carry over to the complex domain, the mean-value scheme for computing interval extensions does, provided the function to be extended is analytic. The proof is a simple application of the Cauchy-Riemann equations and is omitted. For our example, using the natural extension of f' for f_I' (see [8], Section 3.3), we obtain for side AB the interval $C_{AB} = [-6, 0] + i[-6, 2]$ and for side CD the interval $C_{CD} = [-2, 4]$.

In general, the attempt to compute C_L may fail: for example, we might encounter a division by a zero-overlapping interval. In Section 4 we show that this will not occur for sufficiently high settings of the precision and sufficiently short intervals. In any case, if a rectangle C_L enclosing $f(L)$

has been determined and does not cover the origin, then the net amount by which $f(z)$ winds around the origin as z moves along L is well defined by the previously computed values of f at the endpoints of L. In all other cases, L is recursively bisected in exact arithmetic, and the above computations for each half are repeated. This includes the computation of the value of f at the midpoint of L, the analysis of its location, and possibly an unsuccessful return to the calling function in case this value overlaps zero. If the net winding amounts for the two halves of L can be determined (if necessary through deeper recursion) they are added together to obtain the net winding amount for L itself. Finally, the net winding amounts for the four sides of R are summed to obtain the winding number for R.

Returning to our example, we see that since both C_{AB} and C_{CD} overlap the origin, it is necessary to bisect all four sides of R. Let $M_{AB} = 1 - i$, $M_{BC} = 2$, $M_{CD} = 1 + i$, and $M_{DA} = 0$ be the midpoints of the four sides of R. We find $f(M_{AB}) = -3 - 2i$, $f(M_{BC}) = 2 - 3i$, $f(M_{CD}) = 1$, and $f(M_{DA}) = i$. Among the eight subintervals now under consideration, only that with endpoints D and M_{DA} cannot possibly yield a rectangle containing the image $f(DM_{DA})$ without also containing zero. Furthermore, for each of the other seven intervals, the mean-value scheme produces a rectangle not overlapping zero. For example, $C_{AM_{AB}} = [-3.75, -2.75] + i[-2, 2]$. The interval from D to M_{DA} is split once more. Let $M_{DM_{DA}} = \frac{1}{2}i$ be the midpoint between D and M_{DA}, and note that $f(M_{DM_{DA}}) = \frac{3}{4} + \frac{1}{2}i$. The two new subintervals $DM_{DM_{DA}}$ and $M_{DM_{DA}}M_{DA}$ are eligible for the mean-value test. Applying this test to each gives a rectangle not overlapping zero in both cases. It now follows from the locations of $f(A)$, $f(M_{AB})$, $f(B)$, $f(M_{BC})$, $f(C)$, $f(M_{CD})$, $f(D)$, $f(M_{DM_{DA}})$, and $f(M_{DA})$ that f has exactly one zero in the interior of R.

In the general case it is necessary to enforce some kind of limit on the maximum permissible depth of recursive bisections of boundary elements, and to make this limit dependent on the current precision of computation. Given a particular precision setting, we do not obtain value intervals of arbitrarily small width by supplying argument intervals of sufficiently small width. Therefore, it is not useful to allow the argument intervals to become arbitrarily small without consideration of the current precision. In our implementation the depth is (somewhat arbitrarily) limited by the value of the precision setting as expressed in decimal digits. If this limit is reached, the result is once more an unsuccessful return to the calling function. This com-

pletes our discussion of the zero counting routine which we summarize in the following pseudo-code program. The program contains three functions using a C-like syntax (local declarations are omitted).

```
/* for each function, a return value of FAIL indicates failure */
/* first, two auxiliary functions */

int function locate_function_value(point P)
{
    attempt to compute f(P);
    if (computation of f(P) failed) return FAIL;
    if (f(P) overlaps 0) return 0;
    if (f(P) overlaps positive real axis) return 1;
    if (f(P) is contained in first quadrant) return 2;
    if (f(P) overlaps positive imaginary axis) return 3;
    if (f(P) is contained in second quadrant) return 4;
    if (f(P) overlaps negative real axis) return 5;
    if (f(P) is contained in third quadrant) return 6;
    if (f(P) overlaps negative imaginary axis) return 7;
    if (f(P) is contained in fourth quadrant) return 8;
}

int function winding_amount(interval I, integer f1, integer f2)
/* f1 and f2 are the locations of the values of f
   at the endpoints of I */
{
    if (f1 and f2 can be covered by a rectangle not overlapping 0) {
        attempt to compute C(I); /* mean-value estimate of f(I) */
        if (computation of C(I) succeeded
            && C(I) does not contain 0)
            return net winding amount for I;
            /* example: f1 is 2, f2 is 4
               => net winding amount for I is 2 */
    }
    if (bisection limit reached) return FAIL;
    P = midpoint(I);
    loc = locate_function_value(P);
    if (loc == FAIL || loc == 0) return FAIL;
    I1 = one_half(I); I2 = other_half(I);   /* split interval I */
    wind1 = winding_amount(I1, f1, loc);    /* recursive call   */
    wind2 = winding_amount(I2, loc, f2);    /* recursive call   */
    if (wind1 == FAIL || wind2 == FAIL) return FAIL;
    return (wind1 + wind2);
}
```

```
int function zero_count(rectangle R)
/* use of appropriate indices for array loc[] is assumed below */
{
    for each corner P of R {
        loc[*] = locate_function_value(P);
        if (loc[*] == FAIL || loc[*] == 0) return FAIL;
    }
    winding_number = 0;
    for each side L of R {
        wind = winding_amount(L, loc[*], loc[*]); /
        if (wind == FAIL) return FAIL;
        winding_number +=  wind;
    }
    return (winding_number / 8);
}
```

We now discuss the question of when to increase the precision of computation. Initially, the precision is set at the equivalent of 20 decimal digits, but this is more or less arbitrary, and other settings would also work. In the course of testing and bisecting rectangles, the number n_r of rectangles awaiting further processing can grow formidably. This slows down the algorithm and often indicates that the current precision of computation is too low, because a rectangle for which the number of contained zeros is not determined (either because of insufficient precision or because a zero lies on the boundary) is split in two and each half is stored for later processing, even if in reality such a rectangle covers no zeros. Let n_z be the number of zeros known to be covered by all rectangles awaiting further processing. Periodically, n_r and n_z are compared and the precision increased (somewhat arbitrarily) by the equivalent of eight decimal digits if $n_r > 4 * n_z$, since in that case at least one rectangle exists for which the number of zeros covered (namely, none) could have been obtained had the precision been higher. This is because in the worst case, each zero lies on the coincident corners of four adjacent rectangles, and a rectangle covering no zeros can always be verified to have this property provided the precision is sufficiently high (this is proved in Section 4).

From time to time, the remaining rectangles are inspected to determine whether they still represent a connected region. If not, and if the mutually disjoint regions can be enclosed within nonoverlapping rectangles, the original list is split into two or more independent lists, one each for each isolated region. For each new list, the number of zeros covered by its rectangles is

determined by calling the zero counting routine for a surrounding rectangle whose boundary is free of zeros, and by increasing the precision if repeated calls prove necessary. Of course, these new lists may themselves split up at a later time. If the region covered by the rectangles of a particular list becomes sufficiently small, the associated rectangles need no longer be processed.

3 Improvements to the basic algorithm

In this section we discuss three enhancements of the basic algorithm as presented in the previous section. The beneficial effect of these improvements will be demonstrated in Section 5.

The first improvement is made possible by recognizing that the explicit calculation of C_I according to the mean value scheme is frequently not required. Actually, the only reason for computing C_I lies in the hope of producing a rectangle which contains $f(I)$ but not zero. We may carry out a monotonicity test which often indicates that such a rectangle must exist. This test takes advantage of the fact that in our case the interval I is always one-dimensional and either horizontal or vertical, and that a rectangle already exists that contains the values of f at the endpoints of I but not zero (otherwise I would have been bisected). We illustrate the test for a horizontal interval I. Analogous results are obtained for vertical intervals by using the Cauchy-Riemann equations. Let $I = [a+ic, b+ic]$ with $a < b$, $f(z) = u(x, y) + iv(x, y)$, and $f'_I(I) = U + iV$. If neither U nor V contain zero, then the monotonicity of u and v over I implies that $f(I)$ is contained in a rectangle with diagonal corner points $f(a + ic)$ and $f(b + ic)$. This rectangle does not contain zero. Hence C_I need not be obtained. If $0 \notin U$ but $0 \in V$, we may still be lucky. In that case, $f(I)$ is contained in a rectangle whose left side coincides with $x = \min\big(u(a, c), u(b, c)\big)$, and whose right side coincides with $x = \max\big(u(a, c), u(b, c)\big)$. Therefore, if both $f(a + ic)$ and $f(b+ic)$ lie on the same side of the imaginary axis, this rectangle cannot contain zero, and again C_I need not be obtained. Similarly, if $0 \in U$ but $0 \notin V$, then if both $f(a + ic)$ and $f(b + ic)$ lie on the same side of the real axis, it follows that $f(I)$ can be enclosed in a rectangle not containing zero. Again C_I need not be obtained.

The second improvement stems from a better organization and storage of results to avoid repeating identical computations. Each rectangle R main-

tains four pointers to binary trees that store the results of recursive interval bisections along the four sides of R, i.e. the net winding amounts for all the subintervals processed. The net winding amount of f for a subinterval I is stored in terms of an integer value that counts the net number of times $f(z)$ enters a new quadrant or crosses a new half-axis as z moves from one end of I to the other. Two adjacent rectangles will share a tree if their overlapping sides are of equal length; otherwise the shorter of the two will point to a subtree of the other's tree. This change in the algorithm introduces the following slight subtlety: results stored from a time when the precision setting was less than the current value may have to be adjusted. Specifically, if one of the (interval) values of f at the endpoints of I was formerly found to overlap an axis, it may no longer do so. The value must be recomputed and reclassified. If the result is different from what it used to be, the net winding amount for I needs to be incremented or decremented by one.

The third improvement comes from applying Newton's method when working with a list whose rectangles cover only a single (first order) zero. An attempt is made to locate the zero rapidly. Once the iteration appears to have converged sufficiently, we attempt to verify its location by surrounding the point in question by a small square S and counting zeros inside. The size of S is chosen to satisfy the user's accuracy requirements. During the iteration of Newton's method, the precision may have to be repeatedly increased. What is actually done is to compare the width of the most recently computed iterate with the size of the future square S. Unless the former is an order of magnitude smaller than the latter, the precision is increased (in our program by the equivalent of four decimal digits, but this amount is somewhat arbitrary). The new iterate is subsequently assigned its current midpoint value. The iteration is also checked to ensure that it stays in the general vicinity of the region made up by the rectangles in this list. If it 'runs off', bisection is resumed for a while until the next application of Newton's method. Whenever the program returns to the bisection strategy, the precision is reset to its former value at the point when the Newton iteration first began.

4 Proof of convergence

We proceed to prove convergence of the basic algorithm as presented in Section 2. Specifically, we prove the following: let $\eta > 0$ be given, define

U_η as the union of the open disks of radius η centered on the zeros of f located inside the starting rectangle R_S, and let $K_\eta = R_S - U_\eta$. Let a rectangle $R \subset K_\eta$ be given, and assume that the precision of computation is sufficiently high. Then the verification that R contains no zeros of f will succeed.

We begin with a few definitions to establish the properties needed of a variable-precision interval arithmetic system in order to prove convergence of the algorithm. In the following, p and i are integer indexes with values greater than or equal to one. We will work with the set of complex intervals (rectangles) $I(\mathbf{C})$ and its associated topology (see [3], Chapters 5 and 6).

Definition. Given $z \in \mathbf{C}$, let $\left(I_z^{(i)}\right)$ denote any nested sequence of complex intervals containing z such that $\text{width}\left(I_z^{(i)}\right) \to 0$ as $i \to \infty$.

Definition. Let w be a continuous complex-valued function defined on an open domain $D \subset \mathbf{C}$. A *family of interval approximations* for w, denoted $\left(w_I^{(i)}\right)$, is a sequence of functions with domain in $I(\mathbf{C})$ and range in $I(\mathbf{C})$ which satisfies the following properties:

- (1) Given $z \in D$, a sequence $\left(I_z^{(i)}\right)$, and p sufficiently large, then

$$w_I^{(p)}\left(I_z^{(p)}\right) \text{ exists, } w(z) \in w\left(I_z^{(p)}\right) \subset w_I^{(p)}\left(I_z^{(p)}\right) \text{, and}$$
$$\lim_{p\to\infty} \text{width}\left(w_I^{(p)}\left(I_z^{(p)}\right)\right) = 0 \, .$$

- (2) The functions in $\left(w_I^{(i)}\right)$ are inclusion monotonic. Given complex intervals $I_{1z} \subset I_{2z} \subset D$, integers $i_1 \geq i_2 \geq 1$, and the existence of $w_I^{(i_2)}(I_{2z})$, then

$$w_I^{(i_1)}(I_{1z}) \text{ exists, and } w_I^{(i_1)}(I_{1z}) \subset w_I^{(i_2)}(I_{2z}) \, .$$

Informally, the idea is to think of $\left(w_I^{(i)}\right)$ as being defined by some variable-precision interval arithmetic routine designed to approximate an interval extension w_I of w, where i equals the precision setting. Note that the functions in $\left(w_I^{(i)}\right)$ need not be true interval extensions of w in the sense of Moore (see [8], Section 3.3), as we do *not* require $w_I^{(p)}(z) = w(z)$ for $z \in D$ and any $p \geq 1$ (here we use the same notation for degenerate intervals and corresponding complex numbers). Indeed, in practice it is often impossible to

compute exact function values. Also note that we do not in general assume the existence of $w_I^{(p)}(I_z)$ for any interval $I_z \subset D$ and any $p \geq 1$. In practice, if I_z is close to a singularity of w and p small, this computation could fail even if $w_I(I_z)$ exists. Following the next definition, we will state a theorem which can be viewed as an extension of Dini's Theorem from real analysis (see [4], p. 173), and which is key to proving convergence of the bisection algorithm.

Definition. Let L_p be a lattice of points in $\mathbf{C}$ whose real and imaginary coordinates can be expressed exactly with mantissas of at most p decimal digits, and with no restriction on the size of the exponents. Given any $z \in \mathbf{C}$, the interval $Z^{(p)} \ni z$ represents an interval defined by four (distinct) lattice points in L_p. The interval's sides are chosen as short as possible, but z must be contained in the interior of $Z^{(p)}$.

Theorem. Let w be a continuous complex-valued function defined on an open set $D \subset \mathbf{C}$, $\left(w_I^{(i)}\right)$ a family of interval approximations for w, and K a compact set contained in D. On K define for $p \geq 1$

$$r^{(p)}(z) = \begin{cases} \text{width}\left(w_I^{(p)}\left(Z^{(p)}\right)\right), & \text{if } w_I^{(p)}\left(Z^{(p)}\right) \text{ exists} \\ 1, & \text{otherwise.} \end{cases}$$

Then $r^{(p)} \to 0$ *uniformly* on K as $p \to \infty$.

Proof. First we show that for sufficiently large p, $w_I^{(p)}\left(Z^{(p)}\right)$ exists for all $z \in K$. If this were not so, there would exist a sequence (z_k) with accumulation point $\overline{z}$ in K and an increasing sequence (m_k) of positive integers such that $w_I^{(m_k)}\left(Z_k^{(m_k)}\right)$ never exists. Clearly, $w_I^{(p)}\left(\overline{Z}^{(p)}\right)$ must exist for sufficiently large p, say $p > \overline{p}$. Here we obtain a contradiction of inclusion monotonicity as defined for families of interval approximations. Choose an integer i such that $m_i > \overline{p}$ and $Z_i^{(m_i)} \subset \overline{Z}^{(\overline{p})}$. Then $w_I^{(m_i)}\left(Z_i^{(m_i)}\right)$ must exist and be contained in $w_I^{(\overline{p})}\left(\overline{Z}^{(\overline{p})}\right)$.

We now restrict our attention to those values of p for which $w_I^{(p)}\left(Z^{(p)}\right)$ exists for all $z \in K$. It is clear that $\left(r^{(p)}\right)$ is a monotone sequence that converges pointwise to zero on K. Assume now that the convergence is not uniform. Then there exist $\epsilon > 0$ and sequences (z_l) and (m_l) such that $r^{(m_l)}(z_l) > \epsilon$ for all l. The sequence (z_l) has an accumulation point in K, call it z^*. This time note the fact that width $\left(w_I^{(p)}\left(Z^{*(p)}\right)\right) \to 0$ as $p \to \infty$;

in particular, choose p^* such that width $\left(w_I^{(p^*)}\left(Z^{*(p^*)}\right)\right) < \frac{\epsilon}{2}$. Again we obtain a contradiction by choosing i such that $m_i > p^*$ and $Z_i^{(m_i)} \subset Z^{*(p^*)}$, for then

$$r^{(m_i)}(z_i) = \text{width}\left(w_I^{(m_i)}\left(Z_i^{(m_i)}\right)\right) \leq \text{width}\left(w_I^{(p^*)}\left(Z^{*(p^*)}\right)\right) < \frac{\epsilon}{2}\,.$$

Hence, the convergence must be uniform and our proof is complete.

To prove the convergence of the bisection algorithm, we now assume that an algorithm can be specified which computes the values of a family of interval approximations for f, and the values of another such family for f'. For example, this is possible when f belongs to the class of analytic elementary functions mentioned in the Introduction. If f has open domain $D \supset R_S$, it follows from the previous Theorem that there exists a number p_η with the property that if $p \geq p_\eta$, then for any $z \in K_\eta$, $f_I^{(p)}\left(Z^{(p)}\right)$ exists, has uniformly bound width, and does not overlap the origin. (The modulus function $|f(z)|$ is continuous and hence achieves a minimum positive value on the compact set K_η.) In the algorithm, the input to $f_I^{(p)}$ is always exact. Since by inclusion monotonicity $f_I^{(p)}(z) \subset f_I^{(p)}\left(Z^{(p)}\right)$, one cause for failure of the zero counting routine has thus been eliminated.

The second cause relates to the computation of complex rectangles containing the image $f(I)$ for intervals $I \subset R_S$, which in our algorithm is based on the mean value scheme for intervals. These rectangles are computed according to the formula

$$C_I^{(p)} = f_I^{(p)}\big(\text{mid}(I)\big) + f_I'^{(p)}(I)\big(I - \text{mid}(I)\big)$$

where mid(I) denotes the midpoint of interval $I \subset R_S$. It follows again from the previous Theorem, now applied to the function $f(z) - f'(z)(z - z)$, that there exist numbers q_η and $l_\eta > 0$ with the property that if $p \geq q_\eta$ and width$(I) \leq l_\eta$, then for any $z \in K_\eta$, $C_I^{(p)}$ exists, has uniformly bound width, and does not overlap the origin. This eliminates the other cause for failure of the zero counting routine. Of course, the same argument could have been used to prove convergence using the much less effective formula $C_I^{(p)} = f_I^{(p)}(I)$. However, the mean-value formula has second order approximation properties with respect to the smallest rectangle containing $f(I)$ (see [3], Chapter 3) and is especially useful for small intervals.

5 Numerical examples

As mentioned in the Introduction, we chose the Range Arithmetic package [2] to implement this algorithm. The complete program, written in the C++ programming language, is quite complex and consists of over 1000 statements. Much of the complexity is due to the improvements proposed in Section 3 but, as is apparent from the data in Table 1, these enhancements are necessary to get a program of greater practical value. The CPU times

Accuracy: 5 Function	Starting Rectangle	Zeros Found	CPU Times Versions A / B / C	CPU Times Version D
$z^{20}+1$	NE: $2+2i$ SW: $0+0i$	5	13.7 s / 9.4 s / 4.7 s	2.8 s
$5z^{20}-\cos(z)$	NE: $1+i$ SW: $\frac{1}{10}-\frac{1}{10}i$	5	43.9 s / 30.4 s / 15.4 s	7.4 s
$\cosh(ze^z)$	NE: $1+4i$ SW: $-1-i$	5	123.6 s / 99.1 s / 44.2 s	24.8 s
$\sin(z^2)$	NE: $3+2i$ SW: $-4-i$	10	172.6 s / 105.1 s / 44.7 s	19.7
$\sin\left(\frac{z^2+\pi^2}{z+\pi(2i-3)}\right)$	NE: $10+10i$ SW: $-10-5i$	27	971.2 s / 799.3 s / 233.5 s	173.9 s

Table 1

were obtained on an Intel 80486 microprocessor (50 MHz) using Symantec's 32 bit ZORTECH C++ compiler (Version 3.1). Table 1 compares the performance of four different versions of the algorithm: Version A is the basic algorithm as described in Section 2, Version B is the basic algorithm with monotonicity test, Version C adds to Version B the idea of shared binary trees, and Version D is the full-fledged version which also implements Newton's method. Next to each function appears the starting rectangle (its northeast and southwest corner points), the number of zeros found in that rectangle, as well as CPU times in seconds for each of the four versions. In all cases shown, 5 correct decimal places were requested at the start of program execution. When the same problems were run requesting 10 correct decimal places instead of 5, versions A, B, and C required substantially more time than before whereas version D required only slightly more time (except for problem four). This is not surprising since only version D incorporates

Newton's method whereas the other three versions converge linearly to the zeros. Notice also that problem four has a second order zero at the origin which does not benefit from Newton's method. The times for version D when 20 correct decimal places were requested are shown in Table 2.

Accuracy: 20 Function	Rectangles Processed	Memory Required	Max/Ave Depth	Max/Ave Precision	CPU Times Version D
$z^{20}+1$	64	10.9 KB	7 / 4.1	36 / 21.8	3.0 s
$5z^{20}-\cos(z)$	51	8.3 KB	6 / 3.4	36 / 22.5	8.1 s
$\cosh(ze^z)$	47	12.5 KB	8 / 4.4	40 / 22.1	26.5 s
$\sin(z^2)$	773	17.4 KB	8 / 2.1	52 / 29.5	32.7 s
$\sin\left(\frac{z^2+\pi^2}{z+\pi(2i-3)}\right)$	329	111.2 KB	19 / 6.3	40 / 20.4	176.2 s

Table 2

Table 2 also contains other data obtained when running version D on the same five test cases and the same starting rectangles as before (these rectangles and the numbers of zeros found were the same as in Table 1 and are not repeated here). The table shows for each function the total number of rectangles processed and the maximum amount of dynamically allocated memory needed to hold the shared binary trees (in units of 1024 bytes). Also shown are the maximum depth of recursive bisections that occurred during a call to the zero counting routine and the average depth of recursive bisections. For the calculation of the latter two values, only those subintervals I were considered for which it was determined that a rectangle exists which contains $f(I)$ but not zero. Also shown are the maximum precision of computation and the average precision, obtained by averaging over time, in units of decimal digits. The initial precision setting was always 20 decimal digits. Two observations can be made: as one would expect, more complicated functions (such as the fifth problem) require relatively small subintervals to resolve the winding behavior, and multiple zeros require greater amounts of high precision computations. The first of these observations is related to the fact that complex intervals in the form of rectangles suffer from rapid *overestimation* (see [9], Remarks to Chapter 1), while the second is a simple

consequence of the algorithm's slow convergence to multiple zeros on the one hand, and the need for higher precison in the vicinity of a zero on the other.

Actually, higher order multiple zeros often prove troublesome not only because of slow convergence, but because of the frequent need for very fine subdivisions of rectangle sides near such a zero. We illustrate this with the example $f(z) = z^5 - 5z^4 + 10z^3 - 10z^2 + 5z - 1$, which has a zero of order five at $z = 1$, and $w \in \mathbf{C}$ a point whose distance from 1 is 10^{-2}. Then $|f(w)| = 10^{-10}$. If I is an interval containing w such that $\text{width}(I) = 10^{-s}$, it is easy to verify that $\text{width}(C_I) \approx 10^{-2s}$, where C_I is obtained using the mean value form of interval extensions and Horner's scheme to evaluate f and f'_I. The aim is to get a rectangle C_I which does not contain zero. Since $f(w) \in C_I$, we can expect that $s \geq 5$ will usually be necessary to achieve this. A rectangle side containing w and of length $2 \cdot 10^{-2}$ can then be expected to require a bisection depth of $-\log_2(10^{-5}/(2 \cdot 10^{-2})) \approx 11$. This is consistent with the observation that this function required an average bisection depth of 11.1 to verify that the input square centered at $z = 1$ and with sides of length $2 \cdot 10^{-2}$ contains 5 zeros (counting multiplicities). The same problem took 387 kilobytes of dynamic memory and 48 seconds computing time. It is clear that the algorithm is inadequate for the computation of such zeros. Of course in the case of polynomials, a Euclidean-type algorithm could have been used to eliminate multiple zeros at an early stage, but this was not done because our algorithm is not primarily designed for polynomials and more efficient and specialized techniques exist for handling them (for example, see [1], Chapter 6 and [6]).

The data we obtained is of course influenced by our choice of variable-precision interval arithmetic (range arithmetic). Different interval arithmetic implementations differ in their representation of intervals (e.g., midpoint or endpoint representation) and in the balance they reach between efficiency of arithmetic operations and optimality of resulting interval width. Nevertheless, we expect that the general behavior of our algorithm would be similar when programmed on the basis of another interval implementation. In this context it is interesting to note that even within range arithmetic there is the option of representing real intervals with two ranged numbers as opposed to one and complex intervals with four ranged numbers instead of two, with the benefit of sharper intervals for the results of arithmetic operations, especially in the case of intervals whose width is of the same order

of magnitude as their distance from the origin. Naturally, the operations on such intervals are more expensive than on ordinary ranged numbers, not unlike the difference between operations on intervals represented by pairs of floating point numbers and on single floating point numbers. We tested a version of our algorithm (version E) in which the mean-value calculation was implemented using the more expensive interval representation. In some of the example problems, this resulted in reduced dynamic memory requirements due to shorter binary trees and different CPU times (both shorter and longer) when compared to version D. We believe that overall version E is slightly preferable to version D, but more examples would have to be considered to possibly establish a clear preference of one over the other.

References

[1] Aberth, O. *Precise numerical analysis.* Wm. C. Brown Publishers, Dubuque, Iowa, 1988.

[2] Aberth, O and Schaefer, M. J. *Precise computation using range arithmetic via C++.* ACM Transactions on Mathematical Software (December 1992).

[3] Alefeld, G. and Herzberger, J. *Introduction to interval computations.* Academic Press, New York, 1983.

[4] Bartle, R. G. *The elements of real analysis.* John Wiley & Sons, New York, 1976.

[5] Churchill, R. V., Brown, J. W., and Verhey, R. F. *Complex variables and applications.* McGraw-Hill, New York, 1974.

[6] Collins, G. E. and Krandick, W. *An efficient algorithm for infallible polynomial complex root isolation.* Proceedings of ISSAC'92.

[7] Henrici, P. and Gargantini, I. *Uniformly convergent algorithms for the simultaneous approximation of all zeros of a polynomial.* In: Dejon, B. and Henrici, P. (eds) "Constructive Aspects of the Fundamental Theorem of Algebra", Wiley-Interscience, London, 1969, pp. 77–113.

[8] Moore, R. E. *Methods and applications of interval analysis.* SIAM Studies in Applied Mathematics, SIAM, Philadelphia, 1979.

[9] Neumaier, A. *Interval methods for systems of equations.* Cambridge University Press, 1990.

Universität Tübingen
Wilhelm-Schickard-Institut
Sand 13
72076 Tübingen,
Germany