Finding the Complex Roots of Funetions

Explanation File of Program ZCircle

1. Introduction

The concept of complex numbers is a mathematical abstraction that has partieular importance to the physical and engineering sciences. It is very useful in analyzing the behavior of cyclic processes such as radio communication, mechanical vibration and general frequency-dependent phenomenon. As a result, there has been a great deal of study devoted to treating complex-valued functions and to determining specific properties such as *pales* and *zeros.*

ln this chapter, we will consider two types of complex-valued functions: polynomials having a finite number of terms, and *analytic* functions. Note that the qualifier "analytic" means that the function can be expanded in a complex Taylor series. This definition naturally includes finite length polynornials, which makes our classification mathematically vague. However, there is an important practical difference involved in this classification that relates to the distinctly different functionals involved and their influence on the algorithms to be employed. Thus, for the purposes of the following discussion, the function $P(z) = a_0 + a_lz + a_2z^2 + a_3z^3$ will be treated in special ways that may not be compatible with the function f(z) = sin z. However, the techniques that apply to f(z) can generally be applied to P(z).

Finding the complex roots of polynomials is an important goal in engineering analysis. For example, in Volume l of *Basic Scientific Subroutines,* the simultaneous differential equations associated with the compound pendulum were shown to result in a matrix equation. This equation eventually led to the determination of the eigenvalues that describe the modes of oscillation of the pendulum. Finding these eigenvalues required the evaluation of a particular determinant, and naturally resulted in finding the roots of a polynomial. If there were any damping effects involved in the physical formulation of the problem, they would be evident in the complex nature of roots.

The determination of the roots of a polynomial is also central to the Laplace transform analysis technique commonly employed in electrical engineering and elsewhere. The goal is usually to evaluate the pole (infinity points) and zero structure the mathematical simulation of a pro cess such as a feedback control system. As you will see later, the procedures for finding poles are very similar to those for locating zeros.

To begin with the discussion, we will first briefly review some of the basic and very important computational properties of complex functions.

2. Review of the Fundamental Properties of Functions in the Complex Domain

To establish a theoretical basis for the algorithms . to be presented in this chapter, we will first review a few of the fundamental features of complex functions. The complex variable, z, is defined as

$$z = x + iy$$

where *x* and *y* are real numbers and i = sqrt(-1). The complex domain elementary operations of addition, subtraction, multiplication, division, exponentiation, and roots were discussed in detail in Volume I of this series; they will not be discussed further in this section.

Some of the subroutines associated with these mathematical procedures are indirectly used in this chapter. For example, one of the techniques employed to find the complex roots of a polynomial--Newton iteration--requires treatment of the polynomial *P(z)* and its derivative, *dP / dz:* Subroutines for evaluating these functions are provided in Chapter 1 of this volume, and those subroutines in turn calI others which were described in Volume 1. AlI of the required subroutines are provided in this text.

The analyses to be presented are limited to one general class of complex functions: those that are *analytic.* ln simple terms, a function, *f(z),* is said to be analytic in sorne region of the complex plane if the function and *ail* of its derivatives exist in that region. It is immediately apparent that the polynomial *P(z)* is analytic over entire complex plane since *d"P/dz"* exists for aIl *n (n* = 1,2, 3, ...). ln fact, *d"Pdz*^{n} is zero for values of *n* that exceed the degree of the polynomial.

It also follows that if *f(z)* is analytic in sorne region, then *f(z)* can be represented by a polynomial in that same region. The reason for this is that if aIl *d"f(z)/ dz" exist,* a Taylor series polynomial representation can surely be constructed:

$$F(z) = z0 + (z - z0)\, f^{1}(z) + \frac{(z-z0)^{2}}{2\,!}\, f^{2}(z) + \ldots$$

By restricting our attention to analytic functions, we can apply many of the techniques discussed in the previous chapter, such as Newton iteration. This restriction is generally not very encumbering, and as we will see later, can even be ignored to sorne extent *after* we have developed the algorithms. We will impose another restriction on the analysis which, as it turns out, also does not significantly hamper the utility of the algorithms to be presented.

The polynomial root-seeking algorithms will be restricted to polynomials having real coefficients. ln that case, if z1 = *XR* + *iYR* is a root of *P(z),* then so is its complex conjugate Z2 = z; = *XR* - *iYR.* ln other words, the complex roots of polynomials having real-valued coefficients come in conjugate pairs. As we will see with the Lin and Bairstow polynomial algorithms, this conjugate pair property can capitalized on.

As in thermodynamics, a resourceful mathematician armed with a few rules can derive a wide spectrum of useful relations. ln this case, we can note that if f(z)is analytic in sorne region, then it can be represented by a Taylor series in that region. By sorting out the terms of the series appropriately, separation of the real and imaginary (sqrt(-1)*)* parts is possible:

$$F(z) = P(z) = \mu(x,y) + i\, v(x,y) \qquad (7.2.1)$$

where μ*(x,y)* and *v(x,y)* are real-valued functions.

The importance of this last result cannot be overemphasized. It is the key finding the complex roots of general analytic functions using only real-number computations which, after aIl, are the only ones possible in a computer. We can now trade the problem of finding the complex value of z so that *f(z)* is zero for that of finding the real values of *x* and *y* that *simultaneously* make μ*(x,y)* and *v(x,y)* zero :

$$f(z) = 0 \leftarrow>\_ \{ \mu(x,y) = 0 \text{ and } v(x,y) = 0$$

Besides the advantage of both μ*(x,y)* and *v(x,y)* being real-valued functions, there are mathematical interrelations that are very useful when interpreted geometrically. These interrelations can be established by taking the derivative of f(z). By recalling the fundamental definition of the derivative ("in the limit of...,"etc. *), this derivative can be calculated *two* ways (i.e., from two directions):

$$df/dz = \frac{d\mu}{dx} + i\,\frac{dv}{dx} \quad \text{or} \quad df/dz = \frac{dv}{dy} - i\,\frac{d\mu}{dy}$$

Here, dμ/dx, dv/dx, dv/dy, dμ/dy are for partial derivatives.

By equating real and imaginary parts, we get the Cauchy-Riemann differential equations:

$$d\mu/dx = dv/dy \quad \text{and} \quad d\mu/dy = dv/dx \qquad (7.2.2)$$

The Cauchy-Riemann equations offer the convenience of reducing the derivative analysis to a consideration of only one function, say μ*(x,y)*. However, there is also the following powerful geometrie interpretation of the Cauchy-Riemann equations.

Consider the two curves μ*(x,y)* = C1 and v*(x,y)* = C2 shown in figure 7.2.1.

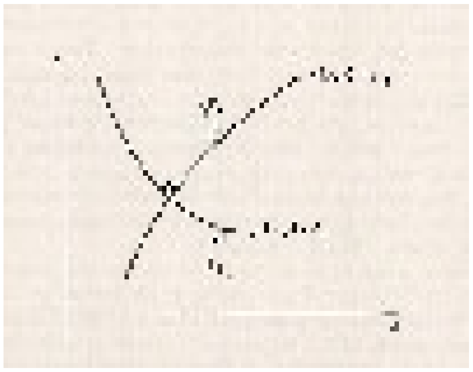


Figure 7.2.1 *If p.(x, y) and v(x, y) are the real and imaginary parts of the analytical function f(z), then the curees μ(x, y) =* c1 *and v(x, y) =* c2 *are perpendicular at their intersection.*

At every point on each curve a perpendicular to the curve can be constructed: P1 and P2. At the intersection of the two curves, it can be shown (by taking the vector dot product of the gradients-see Ref. 38) that the eosine of the angle between the two perpendiculars to these curves at that point is

$$\cos\theta = \frac{\mu x v x + \mu y v y}{\text{normalizing factor}}$$

However, using the Cauchy-Riemann relations, it follows that the numerator is zero and thus cos theta = 0. If the cosine is zero, then the two perpendiculars themselves are orthogonal, and in turn, the μ(x,y) = c1 and μ(x,y) = c2 curves are perpendicular at their intersection.

*See any introductory text on complex variables. A very good discussion is given in Kreysig (Ref. 38).

This orthogonality property is used as the basis for the root-seeking algorithm given in the next section.

We will conclude this section with a discussion of the classic *mapping* operation $z \rightarrow 1/z$. One important use of this transformation is to move the roots of a polynomial to a more convenient region. Consider the polynomial Q(z) which is derived from the Nth-degree polynomial *P(z)* as follows:

$$Q(z) = z^N P(1/z) \qquad (7.2.3)$$

It can be shown that Q(z) is also a polynomial of degree *N,* and that its coefficients are the same as those of *P(z)* but with the order reversed. It can also be shown that the roots of *P(z)* that are outside the unit circle $_1$ z $_1$ = 1 correspond one-to-o with the roots of Q(z) that are within the circle | z | = 1. Thus, a hard-to-find, farflung root can be moved in a very well-defined region. This holds an advantage when deciding where to start the search for the root. A further advantage is that the actual computation, the chances of fatal numeric overflows are reduced and replaced by nonfatal numeric underflows. However, a possible disadvantage is that after the transformation, two very large and widely separated roots may both end up close to the origin, and thereby close to one another. This can cause failures in some algorithms. As an exercise, experiment with this mapping procedure using algorithms presented later in this chapter.

The following sections are divided into two groups. The first group deals wi functions of the form *f(z)* = µ.*(x,y)* + i v*(x,y)*. It quickly tracks the logical sequence of development given in the previous chapter. The next group of sections specifically considers polynomials, both in terms of their treatment as general functions of the form µ. + i v, and as special finite series. The reason for this partitioning will become apparent as you proceed.

3. Interval Search

The first technique considered in Chapter 6 for locating real roots was the interval search (bisection method). Its key advantages were that it was conceptually simple to understand and very easy to code. The main disadvantages found were that was intrinsically very slow in terms of how many iterations were required to reach a chosen level of accuracy, and that the root had to be initially bracketed with two guesses.

Interval searching for complex roots is even slower and more complicated. Assuming the existence of an algorithm that would determine whether or not a root was within a given *area,* the basic bisection algorithm could be applied in *two* dimensions. N^2 iterations would be expected for the complex plane calculation. For example, if N = 20 (about 10-^8 relative accuracy) were required for the one-dimensional calculation, then N = 400 would correspondingly be necessary for the complex plane iteration. Also, determining whether or not a complex root exists within a given region is more complicated and time-consuming than establishing a crossing of the axis in the one-dimensional case. The conclusion is that the classical bisection
method for finding complex roots is too inefficient to merit further consideration, especially when *much* faster algorithms exist.

As with all root-seeking algorithms, a starting point must be chosen for the search. Because it is wasteful to probe a region that does not contain a root, it would be advantageous to have a routine that quickly tests for the existence of a root within some prescribed boundary.

Such an algorithm can be developed with the help of figure 7.3.1 (next page).

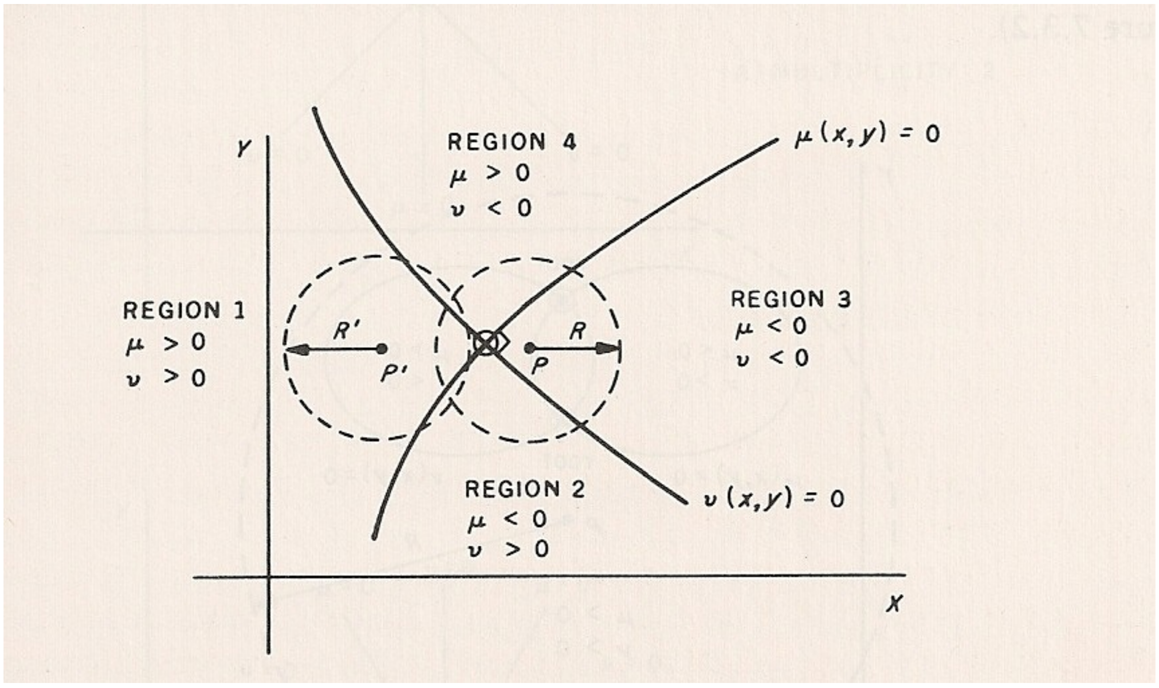


Figure 7.3.1 *The region surrounding a simple root. The desired root is located where the two curoes intersect. The area surrounding the root may be broken up into four regions that are defined by the four possible sign combinations of Ji. and P. The circle shawn with origin* P *is not ceniered on the root, but it does pass through each of the four regions, implying that a root is contained uiithin. The circle with* origin *P' does not enclose the root, nor does* it *pass through all four regions.*

Recall that the complex roots of *f(z)* occur where both μ*(x,y)* and v*(x,y)* are zero. Thus, a root must be located at the intersection of the curves μ*(x,y)* = 0 and v*(x,y)* = O. These curves intersect at right angles and thereby divide the nearby space into four quadrants. The μ.*(x,y)* = 0 curve separates the μ*(x,y)* > 0 and μ.*(x,y)* < 0 regions. Similarly, the v*(x,y)* = 0 curve separates the v*(x,y)* > 0 and v*(x,y)* < 0 regions. We can number the four quadrants according to the signs of μ and v as shown in the figure.

We now draw a circle about the point *P* that has a large enough radius to contain the root. This circle passes through each of the four regions. Note that if we were to travel counterclockwise around the circle, we would pass through each of the four regions in sequence. For exarnple, if the trip started in region 1, the sequence would be (1, 2, 3, 4). If, instead, we were to construct a circle of radius *R'* (centered on *P'),* which did not contain the root, the corresponding sequence might be (1, 4, 1, 2), which is quite different.

The algorithm suggested by these observations is the following. We draw a circle about some point in the complex plane and travel around it once counterclockwise, noting the sequence of regions passed through. If a complete cycle appears in this sequence, then an intersection of the μ*(x,y)* = 0 and v*(x,y)* = 0 curves *must* be contained within. The conclusion is inescapable. If, however, no complete cycle is observed, then it is *likely* that no root is contained within, with some exceptions (see figure 7.3.2).

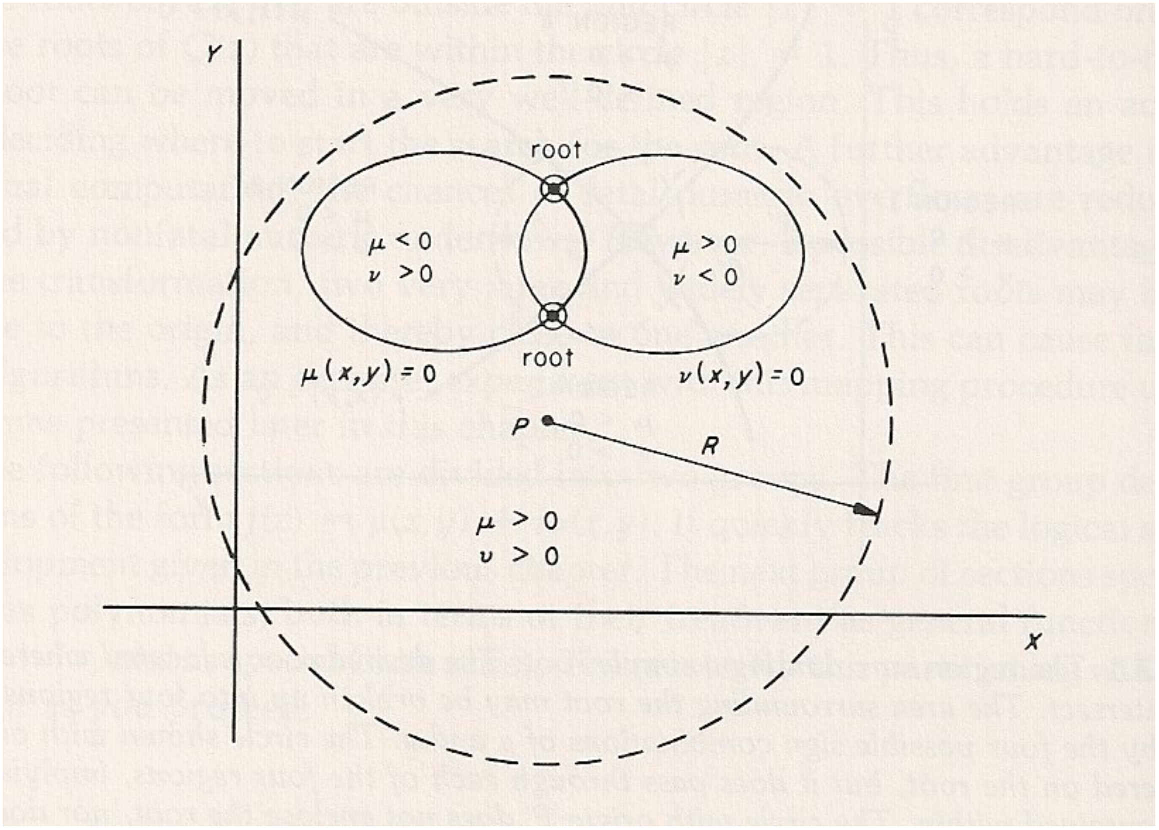


Figure 7.3.2 *A case in which two roots are contained within the test region, but the algorithm fails to note their existence.*

Unfortunately, complications are apparent before we even start to compose a program to implement this algorithm. For example, what if there are multiple roots (more than one at the sa me point) or several roots (more than one, but not coincident) contained within the search region ?

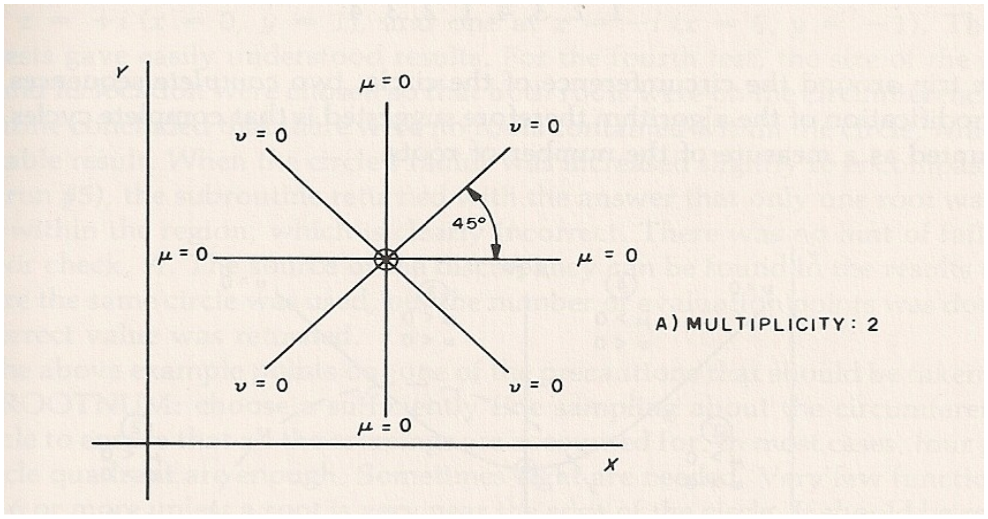
Hamming (see Ref. 16) provides a short but good discussion of this complication. ln brief, the $\mu(x,y) = 0$ and $v(x,y) = 0$ intersection pattern is very weIl defined for multiple roots (examples are shown in figure 7.3.3). We can visualize the patterns as follows. To each member of the multiple root there belongs a pair of $\mu = 0$ and $v = 0$ curves. When there is more than one root, there are that many pairs of curves. Near the intersection, symmetry is maintained, thereby giving the star patterns apparent in figure 7.3.3.

The pattern corresponding to two very close roots is similar to that for two multiple roots. ln the ex ample shown in figure 7.3.4, the pattern for two close roots follows that of the double root pattern shown in figure *7.3.3A* when the observation point is sorne distance away from the roots.

Figure 7.3.4 can also be employed to examine the influence of multiple or several roots on the *(p.,v)* sign sequence. Starting at point *A* on the circle and proceeding counterclockwise, the sequence of regions is

1, 2, 3, 4, 1, 2, 3, 4

In one trip around the circumference of the circle, two *complete* sequences occur, The modification of the algorithm therefore suggested is that complete cycles should be counted as a measure of the number of roots.



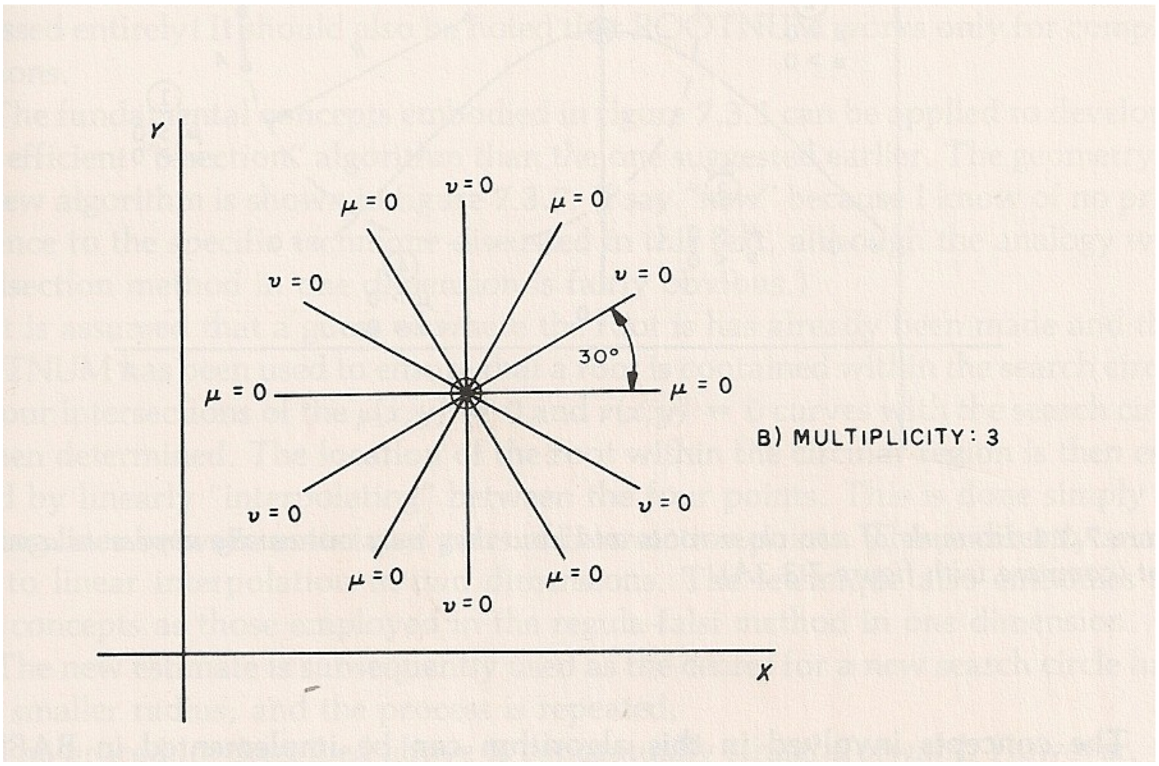


Figure 7.3.3 *Examples of the local intersection patterns associated with multiple roots.*

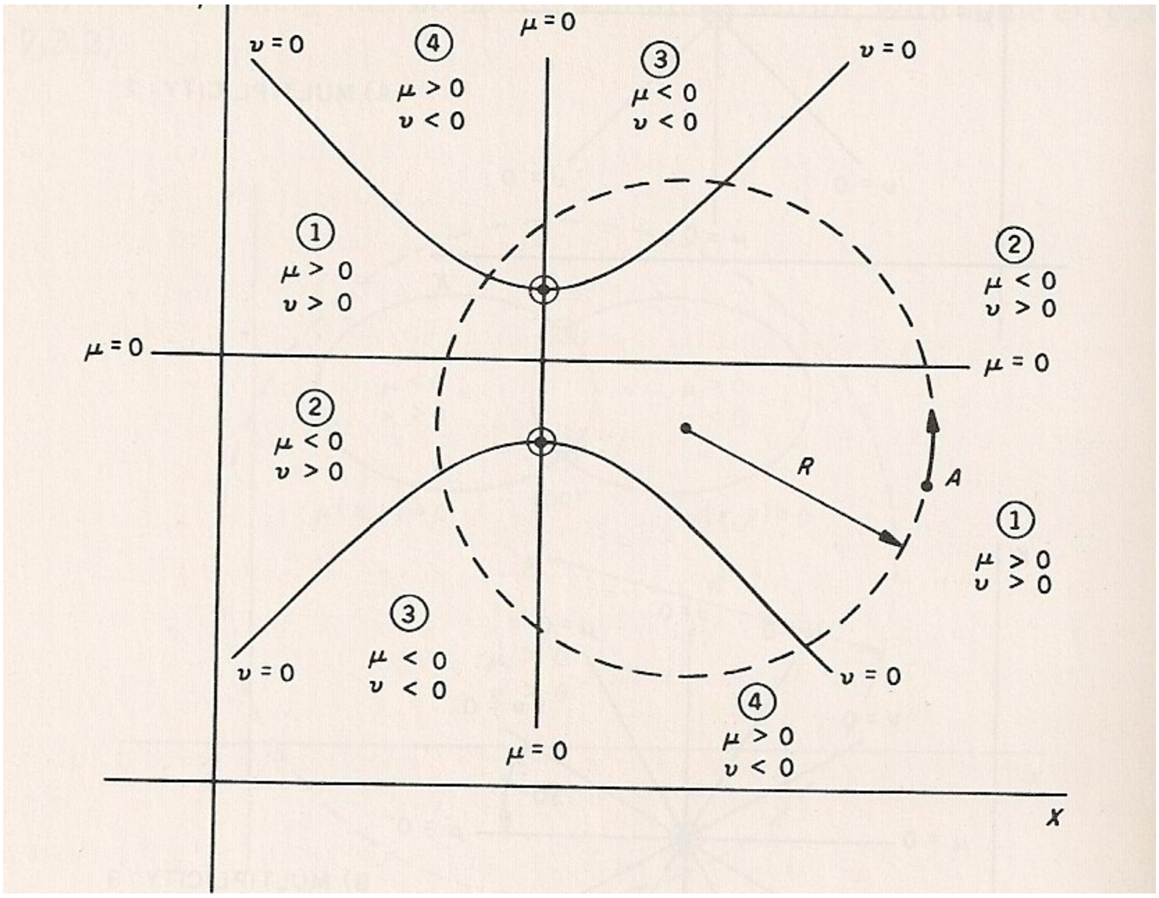


Figure 7.3.4 *Example of two close roots and how they may outwardly appear as a multiple root (compare with figure 7.3.3A).*

The concepts involved in this algorithm have been implemented in program Rootnum.

The inputs to ROOTNUM are the center of the search circle, (XO, YO); the radius of the circle, W; and the number of evaluation points per quadrant of the circle, M. The returned values are *N,* the number of complete cycles (and thus roots) found, and *A,* an error measure. If *A* * 0, then a noninteger number of cycles was encountered and the returned number of roots, *N,* may be in error. The cause of the error is most likely too few evaluation points, M.

To demonstrate ROOTNUM, the function *f(z)* = Z2 + 1 = $(x^2$ - *y2* + 1) + *2xyi* is examined (see listing 7.3.2 and figure 7.3.6). This function has two roots-one at z = +i *(x* = 0, *y* = 1), and one at z = -i *(x* = 0, *y* = -1). The first three tests gave easily understood results. For the four th test, the size of the search circle and its location were chosen so that both roots were on the circumference. The subroutine concluded that there were no roots contained within the circle, which is a reasonable result. When the circle's radius was increased slightly to encompass *both* roots (run #5), the subroutine returned with the answer that only one root was contained within the region, which is clearly incorrect. There was no hint of failure in the error check, *A.* The source of the discrepancy can be found in the results of run 6. Here the sa me circle was used, but the number of evaluation points was doubled. The correct value was returned.

The above example points out one of the precautions that should be taken when using ROOTNUM: choose a sufficiently fine sampling about the circumference of the circle to ensure that aIl the crossings are accounted for. ln most cases, four points per circle quadrant are enough. Sometimes eight are needed. Very few functions require 16 or more unless a root is very near the edge of the circle. It should be rernembered that situations such as that depicted in figure 7.3.2 can occur. and a root can be missed entirelyl It should also be noted that ROOTNUM works only for complex functions.

The fundamental concepts embodied in figure 7.3.1 can be applied to develop a more efficient "bisection" algorithm than the one suggested earlier. The geometry of this new algorithm is shown in figure 7.3.7. (I say "new" because 1 know of no prior reference to the specifie technique discussed in this text, although the analogy with the bisection method in one dimension is fairly obvious.)

It is assumed that a guess of where the root is has already been made and that ROOTNUM has been used to ensure that a root is contained within the search circle. The four intersections of the $\mu(x, y) = 0$ and $v(x,y) = 0$ curves with the search circle are then determined. The location of the root within the circular region is then estimated by linearly "interpolating" between the four points. This is done simply by drawing lines between associated pairs of intersection points. This is directly analogous to linear interpolation in two dimensions. The technique also embodies the same concepts as those employed in the regula-falsi method in one dimension.

The new estima te is subsequently used as the center for a new search circle having a smaller radius, and the process is repeated.

The procedure described above is conceptually straightforward. However, the details of implementation are very important to the success of the algorithm. Because no other literature reference appears to be available, sorne of the important details will be explained more fully below.

The first potential problem that must be guarded against is the possibility that the search circle contains multiple (or several) roots. Figure 7.3.8 shows a case in which a double root is encountered. Four intersection points are found, and a terrible new estimate far outside the search circle is made. A check could be inserted in the subroutine to ignore exterior projections and to go on to find new (μ, v) intersections, but such a check is actually counterproductive. For example, it may happen that the root is actually outside the search circle, and the exterior estimate is valid (see figure 7.3.9). That situation might occur if the radii of the sequential-search circles are reduced too quickly, or if you simply try potluck in choosing the inititial circle.

The alternative procedure is illustrated in figure 7.3.10. ln essence, the object is to locate two $\mu(x,y) = 0$ intersections (with the circle) that are as far apart as possible. The same procedure applies for the two $v(x,y) = 0$ crossings, but with the added restriction that they be as close as possible to 90° away From the $\mu(x,y) = 0$ crossings.

So far, we have assumed that the intersections (points *A,* B, C, and D in figure 7.3.10) have been precisely located. However, such is not the case. Only a finite number of evaluation points (M per circle quadrant) is used in searching for the intersections. Therefore, what is actually known are the locations of the two surrounding evaluation points, (xi-1, yi-1) and (xi, yi). We can interpolate to find a good estimate of the true intersection. As an example, for the situation depicted in figure 7.3.11, the interpolation equations are

$$x = \frac{x_i \mu_{i-1} + x_{i-1} \mu_i}{\mu_{i-1} + \mu_i} \qquad (7.3.1)$$

$$y = \frac{y_i \mu_{i-1} + y_{i-1} \mu_i}{\mu_{i-1} + \mu_i}$$

Note that these equations have the same form as those used in the secant method in

Chapter 6. [Actually, they *look* more like the regula-falsi formulae, but μ_i and μ_{i-1} are necessarily opposite in sign since they bracket $\mu(x,y) = O$].

Once the intersection points have been more accurately located, they can be called on to estimate the position of the root using the following equations:

$$x_r = - \frac{b_1 - b_2}{M_1 - M_2} \qquad (7.3.2a)$$

$$y_r = \frac{M_1 b_2 - M_2 b_1}{M_1 + M_2} \qquad (7.3.2b)$$

Where $$M_1 = \frac{y_B - y_A}{x_B - x_A}$$

$$M_2 = \frac{y_D - y_C}{x_D - x_C}$$

$$b_1 = y_A - M_1 x_A$$

$$b_2 = y_C - M_2 x_C$$

Once the next estimate for the root is found, a new circle (having a smaller radius) can be constructed, and the procedure is repeated.

This iterative process is implemented in program Zcircle. The subroutine appearing in that program contains several safeguards, including checks for divide-by-zero errors, and a special error trap for the situation in which the four required transition points cannot be found. This may happen when there is no root tobe found, or when the $\mu(x,y) = 0$ **and** $v(x,y) = 0$ curves do not intersect the search circle. It may also occur when the $\mu(x,y) = 0$ and $v(x,y) = 0$ curves ***do*** cross the circle, but the evaluation-point grid (number of points/quadrant) is not fine enough. This is the same situation as the one encountered in ROOTNUM (see the previous section).

Figures 7.3.14, 7.3.15, and 7.3.16 give three examples of the application of ZCIRCLE. The intermediate results (the values occurring after each iteration) were obtained by inserting PRINT statements within the subroutine.

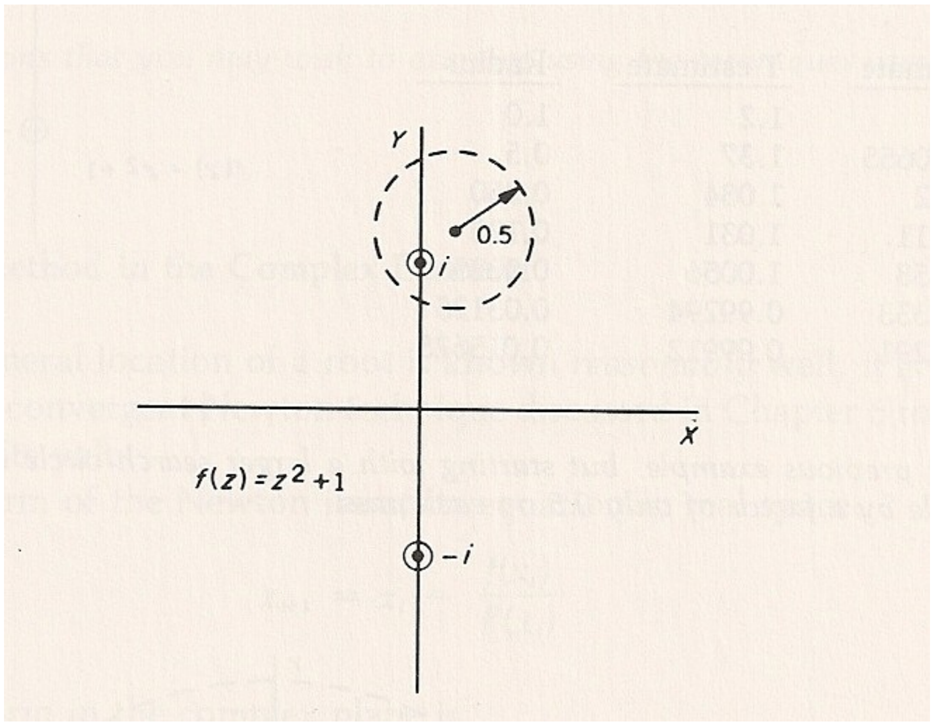
ln the first example (figure 7.3.14), a root was contained within the original search circle, but the radius of the circle was reduced too rapidly and the subroutine failed (gracefully). ln the second example, a larger initial circle was used, and the cir-de radius was reduced less rapidly. The result (after six iterations) was an order of magnitude more accurate than implied by the final size of the search circle. This is usually (but not always) the case after several successful iterations. ln the third ex-ample (figure 7.3.16), the initial search region contained two roots, but the algo-rithm unambiguously closed in on only one of them. This situation contained sym-metry which could have caused sorne difficulty in convergence. However, the de ter-mination of the $\mu(x,y) = 0$ and $v(x,y) = 0$ circle crossings employed a limited num-ber of evaluation points, and this broke the symmetry slightly, thereby favoring one root.

The precautions regarding the use of ZCIRCLE relate to the input parameters. The first search circle should be large enough to contain at least one root. However, if it is too large, the algorithm may fail. (Try an initial radius of 1000 in the third ex-ample.) Second, the radius should not be reduced too quickly. A conservative re-duction factor of 0.5 will usually work if the initial radius is large enough. Much

smaller factors are often possible. Third, the number of evaluation points per quad-rant should be at least four, and occasionally eight.

ZCIRCLE is an interesting subroutine. It is reasonably reliable, given the above precautions. It should be noted in passing that this subroutine can be used only with analytic functions in the complex domain. It should ***not*** be used for real-space problems (e.g., μ = (x,y*),* v = 0).

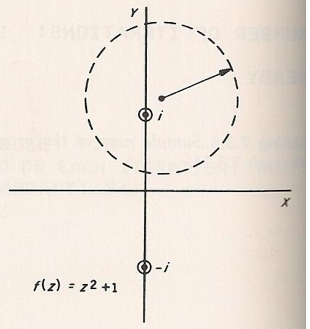
ln the next section, we will examine the complex domain analogy of Newton's method. Newton's method is superior to ZCIRCLE in terms of the convergence rate when the initial guess is close to the true root. ZCIRCLE, however, is more stable when the initial guess is not close to the root, or when there is considerable curvature to the functions μ(x,y) and v(x,y).



center of first circle:	X0 = 0.2
	Y0 = 1.2
radius:	W = 0.5
reduction factor:	E = 0.3
points / quadrant:	M = 4
max. number of iterations:	N=6
retumed result:	X=0
	Y=0
number of iterations:	2
reason:	The first circle contained the root. The new estimate after the first pass was: X = -0.086 Y = 1.16 Convergence was occurring. However, the radius of the next search circle was 0.3 X 0.5 = 0.15. The actual root lay outside this second search circle and a potential infinite loop was encountered, causing a retum to the calling program.

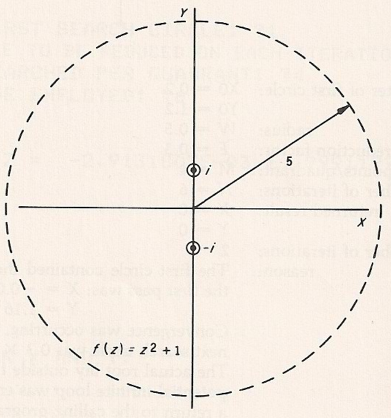
Figure 7.3.14 *An example in which the radius af the search circle was reduced too quickly. The clue that there was a failure is that a six-iteratian limit was allouied, but anly twa passes occurred.*

center of first circle:	$X0 = 0.2$
	$Y0 = 1.2$
radius:	$W = 1.0$
reduction factor:	$E = 0.5$
points/quadrant:	$M = 4$
max. number of iterations:	$N = 6$
returned result:	$X = -0.0029$
	$Y = 0.99912$
number of iterations:	6
reason:	The iteration proceeded well.



Iteration Number	*X* estimate	*Y* estimate	Radius
0	0.2	1.2	1.0
1	0.000655	1.37	0.5
2	−0.102	1.084	0.250
3	0.0411	1.031	0.125
4	0.0158	1.0056	0.0625
5	0.00333	0.99294	0.03125
6	0.00291	0.99912	0.015625

Figure 7.3.15 *Repeat of the previous example, but starting with a larger search circle and reducing the size of the circle by a factor of only 0.5 on each pass.*



center of first circle:	$X0 = 0$
	$Y0 = 0$
radius:	$W = 5$
reduction factor:	$E = 0.5$
points/quadrant:	4
max. number of iterations:	$N = 10$
returned result:	$X = 0.000549$
	$Y = 0.999919$
number of iterations:	10

Figure 7.3.16 *A case in which the initial search circle contained two roots. The center of the circle moved and the circle itself closed down onto one of them.*

From [BIBLI 01].

End of explanation file Zcircle