# THE LOCATION OF ROOTS OF EQUATIONS WITH PARTICULAR REFERENCE TO THE GENERALIZED EIGENVALUE PROBLEM

Gillian Frances Colkin

Submitted to the Council for National Academic Awards in partial fulfilment of the requirements for the degree of Master of Philosophy.

Research sponsored by Middlesex Polytechnic in collaboration with the National Physical Laboratory, Teddington, Middlesex.

January 1984

CONTENTS



(cont.)

CONTENTS (cont.)

## ABSTRACT

### THE LOCATION OF ROOTS OF EQUATIONS WITH PARTICULAR REFERENCE TO THE GENERALIZED EIGENVALUE PROBLEM.

G.F.Colkin

A survey is presented of algorithms which are in current use for the solution of a single algebraic or transcendental equation in one unknown, together with an appraisal of their practical performance.

The first part of the thesis consists of an account of the theoretical basis of a number of iterative methods and an examination of the problems to be overcome in order to achieve a successful computer implementation.

In the selection of specific programs for testing, the emphasis has been placed on methods which are suitable for use, in conjunction with determinant evaluation, for the solution of standard eigenvalue problems and generalized problems of the form $A(\lambda)\underline{x} = \underline{0}$, where the elements of $A$ are linear or non-linear functions of $\lambda$. The principal requirements for such purposes are that:

1. the algorithm should not be restricted to polynomial equations

2. derivative evaluation should not be required.

Examples of eigenvalue problems arising from engineering applications illustrate the potential difficulties of determining roots. Particular attention is given to the problem of calculating a number of roots in cases where a priori estimates for each root are not available. The discussion is extended to give a brief account of possible approaches to the problem of locating complex roots.

Interpolation methods are found to be particularly versatile and can be recommended for their accuracy and efficiency. It is also suggested that such algorithms may often be employed as search strategies in the absence of good initial estimates of the roots. Mention is also made of those features of practical implementation which were found to be particularly useful, together with a list of some outstanding difficulties, associated principally with the automatic computation of several roots of an equation.

ACKNOWLEDGMENTS

I should like to express my particular gratitude to Mr.S.J.Hammarling, formerly of the National Physical Laboratory and now with the Numerical Algorithms Group Ltd., for suggesting the topic for research and for his role as Director of Studies. His detailed advice at each stage of the work has been greatly appreciated. My thanks are extended to Dr.I.O.Grattan-Guinness of Middlesex Polytechnic and to Dr.M.Cox of the National Physical Laboratory for their supervision of the project. I am also grateful to Mr.A.F.Findlay, Head of the School of Mathematics at Middlesex Polytechnic, for his practical assistance.

Considerable use has been made of the computing facilities at Middlesex Polytechnic and I am indebted to the staff of the Computing Centre for their helpful support. I should also like to mention the instruction given by Mr.D.Bush on the use of the "RUNOFF" package to format the text.

(cont.)

The staff of the division of Numerical Analysis at the National Physical Laboratory provided a draft copy of the FORTRAN subroutine RTFSIC and associated documentation. I am grateful to them for allowing me to test and comment upon this program prior to publication and to reproduce details of the parameters. Finally, I should like to thank Dr.J.H.Wilkinson for permission to quote extracts from his unpublished notes on Differential Equations and Eigenvalue Problems [48].

---

# CHAPTER 1

# INTRODUCTION

Obtaining numerical solutions of a non-linear algebraic or transcendental equation in one unknown is a frequently occurring problem, for example in connection with differential equations arising in mathematical physics, and a number of iterative methods are discussed in standard numerical analysis textbooks. The aim of this investigation is to examine the practical implementation of such algorithms and to report on the current "state of the art" in the development of software in this area.

The problem of determining numerical solutions to an equation may be considered in two stages:

1. Estimating the number and approximate location of the required roots. This is often the most formidable part of the task and the question arises as to whether the computer can be of assistance.

2. Refinement of these first estimates by means of one or more iterative techniques. The aim will be to achieve a degree of accuracy specified by the user or, failing that, the maximum accuracy which can be attained by the machine for the particular problem posed.

A considerable amount of work has been done on the automatic solution of a polynomial equation, but even in such cases it is not possible to guarantee a complete solution. Furthermore, ill-conditioning of the roots may prevent the achievement of an acceptable degree of accuracy. The problems with a general equation are usually much greater since the number of roots is likely to be unknown and there may be discontinuities of the function and/or its derivatives. It will also be necessary in such cases to consider the accuracy and efficiency of the method chosen to evaluate the function. As such evaluation will involve approximation of infinite processes, the relevant theory is much more complicated than for a finite polynomial.

A short account will be given of the properties to be considered when choosing an algorithm and incorporating it into a computer routine. This will be followed by descriptions of some commonly-used methods and an appraisal of their performance in particular implementations. Many of the published programs are designed for the computation of real roots but consideration will also be given to the problem of

detection and estimation of complex roots. The two-dimensional nature of the complex variable makes it considerably more difficult to set up a systematic search procedure when good estimates of the roots are unavailable. In addition, the increased number of calculations to be performed makes the question of efficiency even more important than in the case of real roots.

Particular reference will be made to the standard and generalized eigenvalue problems of linear algebra which can give rise to algebraic or transcendental equations with real and/or complex roots. Particular features of such equations will be used to test aspects of various equation-solving routines.

---

# CHAPTER 2

# FEATURES OF ALGORITHMS

The topics considered here will be relevant to a wide variety of numerical problems but particular emphasis will be placed on the way in which general requirements influence the choice of an iterative method for equation-solving and the way in which it is implemented. Some of the criteria established have been adopted by most published computer routines; others are only occasionally incorporated or are still in course of investigation; most will be applicable to both real and complex root-finding procedures.

## 2.1 FIRST ESTIMATES

If a continuous function is known to have a simple root isolated within a certain interval (or region in the case of a complex root), the evaluation of such a root should present little difficulty. In such cases we might reasonably expect guaranteed convergence to any desired accuracy within the capacity of the machine. If such a

priori information is not available we might try instructing the computer to test a series of values of the variable, perhaps proceeding by fixed step lengths, until the function value changes sign. This will not of course be a satisfactory procedure for complex roots or for real zeros of even multiplicity. There is also a danger that an interval thus found may contain several closely-spaced roots.

An alternative strategy is to apply an iterative method with an experimental starting value and to examine the first few iterates for an indication of convergence. To facilitate such preliminary trials provision is being made increasingly for "reverse communication", an approach recommended by Gonnet [18]. Here control is in the hands of the user via his calling program; he examines each successive iterate and decides whether to accept this value and use it as the starting point for the next iteration, accept it as a solution or discard it in favour of a different initial estimate. Even with such provision it will still be the responsibility of the user to seek a reasonable starting value before using the computer at all.

> "For automatic computation the problem of the initial value looms large and forbidding. It is at once the chief characteristic of iterative algorithms and their principal curse" Acton [1]

Guidance may be obtained from one or more of the following considerations:

1. Theoretical bounds on the magnitudes of the roots and knowledge of their distribution.

2. Anticipation of likely roots from the nature of the practical problem which gave rise to the equation.

3. Experience of the behaviour of equations of a similar form.

Acton [1] points out that equations seldom arise as a "one-off" but are liable to be presented as a family of related problems. Careful analysis of the nature of such equations can often allow exploitation of their common features with valuable pay-off in increased efficiency.

## 2.2 CONVERGENCE

Any procedure based upon the use of an iterative formula will need to set criteria for termination of the program. When the process is convergent, an accuracy requirement (stopping criterion) will determine whether a sufficient number of iterations has been performed. A condition to detect failure to converge will also be required. Published routines differ in the freedom they offer to the user in setting limits.

## Stopping Criteria

If the sequence of successive estimates to a root a is denoted $\{x_i\}$ $(i=1,2,\ldots)$ we have the theoretical limit

$$\lim_{i\to\infty} |x_i - a| = 0$$

In practice, attainable accuracy is limited by the capacity of the machine. An absolute error criterion of the form

$$|x_r - x_{r-1}| < \varepsilon$$

where $\varepsilon$ is a fixed small quantity, is clearly unacceptable if the routine is to cope with roots of widely differing magnitudes. It is usual to use the relative condition

$$|x_r - x_{r-1}| < \varepsilon |x_r|$$

with the proviso that the root sought is not actually zero. In the case of a simple root at zero, for example, we may have that

$$\lim_{r\to\infty} \left| \frac{x_r - x_{r-1}}{x_r} \right| = 1$$

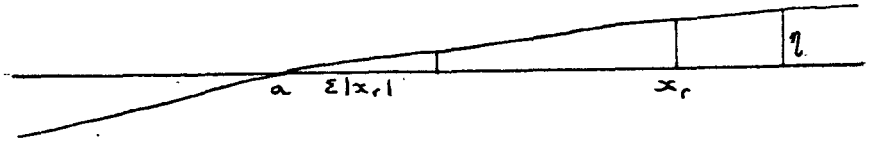
( Wilkinson [47] ), so that an origin shift would be required before considering relative error. Many procedures offer the alternative criterion

$$|f(x_r)| < \eta$$

for users whose main priority is a small function value rather than very accurate root location. Where the tolerance levels $\varepsilon$ and $\eta$ are chosen by the user, it is possible to suppress either of the criteria by setting the appropriate tolerance to zero. This is a useful feature when, for example, the user suspects that the function

value changes only slowly in the region of the root (Fig 2.1) so that there is a danger that the function value criterion will indicate a root while $|x_r - a|$ is unacceptably large. Such a situation will arise, for example, when a polynomial of high degree is evaluated with small x.

Fig. 2.1




The inexperienced user may find difficulty in fixing appropriate tolerance levels, particularly when very accurate results are required. It is desirable that the program designers give guidance when possible although such advice will necessarily be influenced to a large extent by the particular machine used for testing. A good example is provided by Barrodale and Wilson in the documentation of a Fortran program using Muller's method [6]. Here they recommend setting $\varepsilon = 10^{-3d/4}$ and $\eta = 10^{-d+1}$ for a simple root, where d represents the number of decimal digits of accuracy available.

The idea of an automatic stopping criterion has been included in texts for some time (e.g. Dahlquist and Bjorck [13], Wilkinson [47]) but has not yet come into general use in published routines. With standard conditions for termination, if the user requests a high degree of accuracy the procedure may reach a stage at which rounding errors in the function evaluation outweigh the gain in accuracy which would theoretically be obtained by performing further iterations. Thus although

$$| x_r - x_{r-1} | > \varepsilon | x_r |$$

we have that

$$| x_{r+1} - x_r | \geq | x_r - x_{r-1} |$$

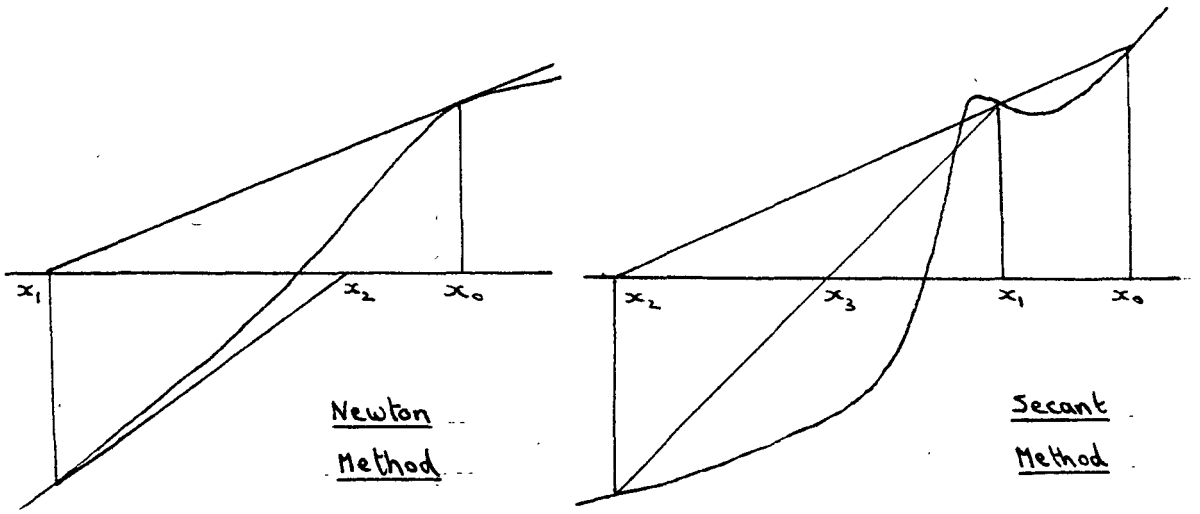
and convergence has effectively ceased. The range of values of x for which this phenomenon is observed is clearly dependent upon machine capacity and is referred to by Wilkinson as the "domain of indeterminacy". It is likely that subsequent iterates will show no obvious pattern of behaviour with the eventual result that failure will be indicated when the maximum number of iterations is reached. Instead we should prefer the program to indicate that convergence has taken place and to output the most accurate estimate obtained by the machine. This will also prevent time being wasted on further iterations which produce no improvement in the solution. For most algorithms it will not be satisfactory to apply the test

$$| x_{r+1} - x_r | \geq | x_r - x_{r-1} |$$

at the start of the iterative procedure, as it is common for the process to require several iterations in which to

"settle down". Fig 2.2 shows possible behaviour patterns for two well-known methods.

Fig. 2.2




In both cases one poorer estimate is obtained before convergence is apparent. Hence to implement an automatic stopping criterion of the form

$$|x_{r+1} - x_r| \geq |x_r - x_{r-1}|$$

we also require a condition to ensure that convergence has commenced. This can be of the form

$$|x_r - x_{r-1}| < \delta$$

or the relative condition

$$|x_r - x_{r-1}| < \delta |x_r|$$

where $\delta$ is a tolerance level considerably larger than attainable accuracy (say 0.01 for relative error). A procedure currently being developed at the National Physical Laboratory [20] uses the latter condition which again has the advantage of versatility and should suffice to ensure that sufficient iterations have been carried out to establish a steady convergence pattern. Such devices,

if successful, will enable the user to get the maximum possible accuracy from the implementation of the algorithm for his particular computer installation without the need for guesswork in determining the tolerance levels.

Failure Criteria

Most of the procedures to be discussed herein either set, or require the user to supply, a maximum number of iterations (or function evaluations) which should not be exceeded. This can refer to each individual root or to the total for all the required roots. This is essential in order to terminate execution or to switch to an alternative algorithm should the method fail to converge or should the rate of convergence be unacceptably slow. Writers of software are often able to suggest probable numbers of iterations for "well-behaved" functions. Such information will be based partly on experience with a variety of test cases and partly on their knowledge of the theoretical rate of convergence of the chosen algorithm. For a convergent iterative process there exists a positive real number p, known as the order of convergence, such that

$$\lim_{k \to \infty} \frac{e_{k+1}}{e_k^p} = c$$

where $e_i$, $e_{i+1}$ are the absolute errors in the successive iterates $x_i$, $x_{i+1}$ and c is a non-zero constant.

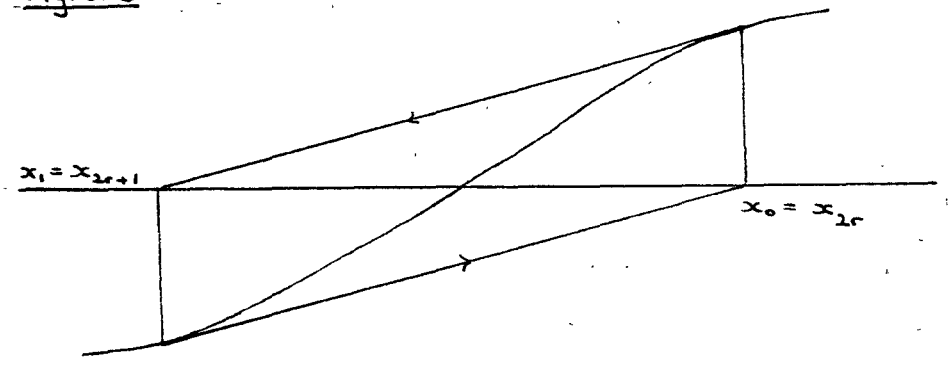
The simplest example of the use of this concept is provided by the bisection method for which p = 1 and c = 0.5 as the interval of uncertainty is halved in length after each application. The number of iterations required will thus be equal to the number of binary digits required in the answer; that is, about 3.3 iterations per decimal digit (Kronsjo [25]).

Generally speaking, the causes of failure become more numerous the more complicated the algorithm, so a change to a simple method such as bisection of the interval may give satisfactory results when other methods fail. For a simple isolated real root of a continuous function, the Bolzano-Weierstrass theorem ensures convergence for this algorithm.

If a particular method produces divergence, the values generated may cause overflow before the maximum number of iterations is reached. This situation can be allowed for by setting a bound on absolute values of x. If this bound is exceeded the program can then be terminated with a more helpful message which may enable the user to locate the region of difficulty. Alternatively the value of x can be set to its upper bound and another attempt made to obtain convergence.

Failure to converge will not necessarily result in large estimates, however. Some algorithms can display oscillatory behaviour or may follow no obvious fixed pattern. Provision for the detection of oscillations would probably result in an unnecessarily complicated program for such occurrences will be relatively rare. Fig 2.3 illustrates a possible oscillatory pattern for Newton's method.

Fig. 2.3




## 2.3 EFFICIENCY

There are two aspects to operational efficiency - time taken and storage capacity required. The latter will be less important in most examples of the type considered here, as large storage requirements commonly arise from either large amounts of data or the need to set up arrays of large dimensions. Routines for the solution of a single equation are only likely to require array space sufficient to accommodate the established roots and corresponding function values. The workspace requirements for the execution of iterative procedures are very modest.

It follows that storage will not be a major consideration except, perhaps, for microcomputers.

The question of execution time will depend partly upon machine characteristics and the standards of programming style adopted, but the amount of arithmetic required is by far the most important consideration. Kronsjo [25] refers to the latter as "computational complexity" which may be interpreted as the number of operations required to solve a problem of given size n, but warns that the fastest methods are not necessarily stable. Equation-solving algorithms are mostly simple in structure so that for all but the simplest functions (e.g. polynomials of low degree) the time taken is almost wholly dependent upon the speed of function evaluations. In the case of a polynomial the problem size may be specified by the degree of the polynomial but other forms of function must be evaluated by truncation of infinite series and the number of computations involved will be machine dependent. The onus must be on the user to ensure that function evaluations are carried out as efficiently as possible; a number of methods have been developed for polynomials (e.g. Kronsjo [25]). If the algorithm requires values of the derivatives, such evaluations must of course be included in the operation count and are likely to reduce considerably the efficiency of such algorithms, despite their superior speed of convergence.

Assessment of the efficiency of an algorithm is based on the two factors:

1. The order of convergence, $p$, which determines the number of iterations required to achieve a given accuracy with a certain initial approximation.

2. The number of calculations required to perform one iteration. When derivatives are not involved this can often be measured by the number of function evaluations.

The ideal algorithm would perform well in both these respects, but the relative importance of 1 and 2 may in practice depend upon the comparative costs of evaluation of the function and its derivatives and the cumulative effects of rounding errors.

Measures of Efficiency

Two simple indices have been proposed, based on the order $p$ and the number $n$ of function evaluations per iteration viz:
the Traub index $p/n$ and the Ostrowski index $p^{1/n}$ .
For a more precise measure it is necessary to introduce as parameters the numbers of arithmetic operations required to evaluate the function(s) and the iterative formula. Traub suggests the formula $E = p^{1/\theta}$, where $\theta$ represents the total cost of computation for function evaluation. If the function and its derivatives each involve the same number

of computations to evaluate, the formula reduces to the Ostrowski index quoted above.

Improving Efficiency

Attempts to speed up convergence by using an algorithm of higher order tend to yield diminishing returns as the greater complexity of the formulation not only increases the amount of computation but the program may also be more prone to failure and be more difficult to maintain.

When several roots of a single equation are required, efficiency can be improved by adopting a systematic search procedure. If approximate locations of the roots are known the order of their calculation may be predetermined. In the absence of such information it is desirable to ensure that the same root is not found repeatedly. This may be accomplished in the case of polynomials by a process of deflation, that is, supposing the root $x = a$ has been found, the polynomial is divided by $(x - a)$. By this means we not only remove known roots but we have now a lower degree polynomial to solve with correspondingly less computation. With a general function such a quotient cannot be found explicitly; instead an attempt can be made to suppress a previous root by proceeding with the function

$$\frac{f(x)}{(x - a)}$$

and perturbing the root slightly before proceeding to avoid a division by zero. This will not prevent multiple

roots from being found in accordance with their multiplicity. Wilkinson [46] has demonstrated that it is desirable when solving a polynomial equation with deflation to find the roots in increasing order of magnitude in order to avoid serious deterioration in the condition of the function. Less is known of the significance of order of root determination for other functions.

After suppressing the previous root it is often convenient to use this value as a "stepping-off" point in the search for the next root. This is not always sufficient, however, to ensure that the roots are found in numerical order. An example is the frequently used Muller method, whose behaviour in this respect is not as yet predictable. Observation of results for this method would seem to indicate sequential "runs" of roots broken by occasional "jumps". Predictably, roots obtained after such "jumps" require rather more iterations than roots which are near neighbours [Chapter 4].

In addition to providing initial estimates, careful observation of the nature of the equations can lead to improvements in efficiency. Anticipation of relationships between the roots, such as complex conjugates or the existence of geometrical or algebraic symmetry can speed up root-finding and also aid in the checking of results.

## 2.4 ACCURACY

Errors in numerical processes may be classified as truncation errors or rounding errors. The former arise when an infinite process is replaced by a finite number of calculations, for example the use of the first few terms of an infinite Taylor series. In the context of equation-solving we encounter such approximations both in the estimation of irrational function values and in the iterative formula itself since this usually involves the replacement of our given function by some polynomial or rational function of low degree. Theoretical predictions can often be made of truncation errors; such estimates enable us to calculate an order of convergence and hence predict the likely number of iterations required for a certain degree of accuracy, given the multiplicity of the root.

Rounding errors arise from several sources, principally:

1. Inaccuracies in experimental data
2. Conversions to and from binary representation
3. Limited machine capacity

Such errors are more difficult to predict than truncation errors and consequently are not easy to allow for in program design. Most of the current knowledge in this area is derived from practical observation, and experience

is the best guide when fixing realistic tolerance levels. Optimistically small limits may take us inside the region of indeterminacy where we risk not only wasting time on superfluous iterations which give no improvement in accuracy but also possible instability. Wilkinson [47] has noted that the danger of instability is particularly apparent in the case of Muller's method when further iterations may result in a move outside the region of indeterminacy and even subsequent convergence to a different root from that sought. In settling for comparatively large tolerances for safety we do not achieve the accuracy of which our machine is capable.

Fixing tolerance levels before commencing calculation is further complicated by the often unpredictable occurrence of ill-conditioned roots. Here a small change in one or more coefficients results in a large change in the computed solution so the effect of rounding errors becomes drastic and the region of indeterminacy correspondingly large. The example

$$f(x) = \prod_{r=1}^{20} (x-r) ,$$

quoted by Wilkinson [46], has become a classic illustration of this phenomenon, as the computed values of the larger roots are complex with imaginary parts of considerable magnitude. In cases such as this the problem is inherent in the function itself and little improvement can be effected by switching to an alternative algorithm;

it becomes particularly important to aim for maximum attainable accuracy in such circumstances and, where possible, to test the results for accuracy. Multiple roots are particularly prone to ill-conditioning. Dahlquist [13] derives formulae for method-independent error estimates but since these are dependent upon derivative values they are likely to present difficulties in practice. It is to be hoped that an "automatic stopping" criterion will take us some way towards overcoming these problems.

It has been noted earlier that attempts to increase the rate of convergence of an algorithm will generally result in a more complicated formulation. It follows that the gain in taking higher order processes becomes progressively less as the reduction in truncation error is counterbalanced by increased rounding errors. The major build-up of rounding errors will, however, be in the calculation of function values unless the function is particularly simple, and at least the same accuracy will be required in this computation as in the use of the iterative formula. Attainable accuracy in the root will usually be less than machine precision. In the particular case of the eigenvalue problem Wilkinson [47] has found that the required accuracy rarely exceeds ten significant figures but this will be insufficient for working accuracy. Thus a double precision facility will be required for such problems on many installations.

To summarise, we can predict a probable number of iterations from the asymptotic behaviour

$$\lim_{k \to \infty} \frac{e_{k+1}}{e_k^p} = c$$

but such a pattern is likely to be masked by rounding errors before it becomes apparent.

## Testing Programs and Results

Test data should consist of a wide variety of functions to illustrate simple, multiple and closely-spaced roots. If the magnitude of the function values is likely to be rapidly changing in practical applications such examples should, of course, be included. The inclusion of complicated functions may enable us to examine the effect of error accumulation in the function evaluations. For similar reasons we should not confine testing to exact data involving few digits. When a root a of odd multiplicity is obtained, the function values f(a - h) and f(a + h) on either side of the root should be examined for a sign change. Care must be taken in the selection of h so as to be outside the region of indeterminacy whilst not being influenced by other roots in the vicinity.

The effect of the order of calculation of the roots on their accuracy has been mentioned in connection with efficiency. For a polynomial equation Wilkinson [46] states that:

> "There is a danger that the zeros of quotient polynomials may gradually diverge from those of the original polynomial"

He does not, however, regard this problem as sufficiently serious to prohibit use of the deflation technique. Pre-determination of the order of calculation of the roots can prove difficult for a general equation when we may not even know the number of roots in the interval under consideration. When suppression of previous roots has been employed and the accuracy of the results is in doubt, further iterations can be performed using the original equation in order to "purify" the solutions. The sensitivity of a root to rounding errors may be estimated by means of experimental perturbations of the input data [13] but further work may be necessary to separate the effects of the algorithm from the condition of the problem itself.

## 2.5 ROBUSTNESS AND SECURITY

These are factors of importance to any purchaser of software and particularly so in the case of "basic" routines, that is, those which will be used frequently for a variety of applications. Equation-solving is just such a situation and failure to produce roots of reasonable

accuracy for "well-behaved" functions would render a program unacceptable. The robustness of a procedure is its ability to solve satisfactorily a wide range of problems; this will include, in equation-solving, such phenomena as multiple and closely-spaced roots, polynomial and transcendental equations, very large, small or rapidly changing function values and, perhaps, discontinuities.

It should, however, be remembered that striving for versatility may reduce the efficiency of the algorithm for certain types of problem. Acton [1] stresses the importance of "suiting the tool to the task" and exploiting any special features of a set of equations to be solved. It is thus highly desirable that the programmer should be aware of the type of equation which the user is likely to encounter and to include, if possible, in his testing some functions which have arisen from similar applications or which possess comparable features.

The possibility of accepting a value which is not actually an approximation to a root has potentially more serious consequences. Such an eventuality will be referred to as insecurity. In the results obtained [Chapter 4] it will be seen that routines frequently accept a discontinuity as a valid root. It is advisable for the user always to request a print-out of function values and estimates of derivatives, if available, to guard against such occurrences. Many published programs

also provide a count of the number of iterations performed for each root; such information can be helpful as a steep rise in the number of iterations may indicate a move away from the region of interest or an unreliable estimate of a root. Problems can also arise with indeterminate quantities of the form 0/0 such as

$$\frac{\sin x}{x} \qquad \text{( where x is small )}$$

Although theoretically stable, not all machines will give reliable results near the limit; for example the Commodore hand calculator SR4148R gave the value of the ratio as approximately 13 when $x = 10^{-12}$. If possible formulae should be rearranged to avoid this situation; alternatively, a power series approximation to a transcendental function may be of use in the detection of limits (and might, incidentally, be quicker to evaluate than the library function). Other safeguards against the acceptance of incorrect answers include function evaluation at neighbouring points, consideration of the practical problem from which the equation arose and repeating the calculation with different initial estimates and/or an alternative algorithm.

## 2.6 MAINTENANCE AND CONVENIENCE FOR USER

Suitable presentation of software is another feature generally applicable to procedures which it is anticipated will have frequent and widespread use. It is not appropriate here to discuss in detail those aspects of programming which make a routine easy to correct and update but the needs of the user should be considered when choosing an algorithm and deciding upon the manner of its implementation. Generally, the more complex the method, the less readable the program and the more provisions will need to be made for possible types of failure. Similarly, the more sophisticated we wish to make the program in order to improve efficiency, the more information will be required from the user and the number of input parameters may become unwieldy. If the "reverse communication" system is adopted in order to allow flexibility to the experienced user it may be necessary to provide subsidiary programs to enable the routine to be used by the non-specialist in a straightforward way for the solution of "simple" problems. Clear documentation, sample calling programs and guidance on the choice of convergence criteria can be of great assistance to users.

Possible sources of failure cannot all be predicted, but the designer of the program should endeavour to forsee as many difficulties as possible and arrange for explanatory messages to be output. Failure to do this can result in the program terminating prematurely for reasons

which are not at all obvious to the user, run-time errors being notoriously difficult to trace on many systems. Such failure often manifests itself as overflow or underflow messages which can arise from numerous points in the procedure. At the very least, provision should be made for checking the validity of the input (which may arise from another subroutine and not be seen by the user) and for limiting the number of iterations to a fixed or user-supplied maximum.

Time and care at the testing and documentation stages can contribute greatly to the reliability and length of life of the program.

---

# CHAPTER 3

# ITERATIVE METHODS

## 3.1 CLASSIFICATION

Although methods of obtaining numerical solutions to equations have been of interest to both pure and applied mathematicians from ancient times, the development of a classification system for algorithms has come about concurrently with the increased use of computing machinery. A recent comprehensive theory of such algorithms was put forward in 1964 by J. F. Traub [43]. Iterative formulae are classified according as

1. they are single-point or multi-point
2. they require or do not require the use of a memory facility.

Single point methods introduce only one new value of the independent variable at each iteration, all required function and derivative values being calculated at this point or being re-used from previous iterations.

Multi-point methods require two or more previously unused values of the independent variable at each stage.

A memory facility will be needed when the iterative formula involves previously calculated values of the function or its derivatives.

Within each of the four resulting categories, Traub examines the order and efficiency of various algorithms. Some general observations can be made concerning algorithms of particular types which may influence the choice of method for a given equation.

## 3.2 ONE-POINT METHODS

The procedure without memory will be of the form

$$x_{r+1} = \phi( x_r , f_r , f_r' , f_r'' , \ldots)$$

for some function $\phi$. Traub proves that the informational efficiency as measured by the index p/n, where p is the order of convergence and n is the total number of function and derivative evaluations at the new point, cannot exceed unity for methods of this type. Hence an optimal one-point method without memory has efficiency equal to unity, and such a method can be constructed for roots of any multiplicity and any chosen order p. It is also shown that such a method must depend explicitly on the first (p - 1) derivatives of f.

If the use of memory is permitted, the general one point method may be represented:

$$x_{r+1} = \phi( x_r , f_r , f_r' , f_r'' , \ldots , x_{r-1} , f_{r-1} , f_{r-1}' , f_{r-1}'' , \ldots )$$

Traub conjectured that the order of a one-point method, with or without memory, cannot exceed (n + 1), n being defined as above. This result has since been proved, for example by Brent, Winograd and Wolfe in 1973 [9].

The order of the method approaches (n + 1) as we increase the extent of re-use of previously calculated values and the limit is approached sufficiently rapidly for the introduction of large amounts of previous data to be of little practical use. Hence methods of this type will usually make use of either one or two previous data points only. It further follows that the Traub efficiency index will be less than (1 + 1/n) so that it will also be desirable to limit the number of evaluations to be carried out at each iteration.

Derivative Methods

It can be concluded from the above that, even with the use of unlimited memory capacity, a one point method will require the evaluation of at least one derivative of the function if the convergence rate is to be of order two or more. For many functions this will prove to be a serious drawback to the procedure. Kronsjo [25] expresses the difficulties thus:

> "In the case of numerical differentiation the problem is inherently unstable and so no good computational methods can be expected to exist at all"

The problem is likely to be exacerbated in the case of eigenvalue problems by the formulation of the function as a determinant. This adds a formidable amount of computation to the probable numerical inaccuracy.

Kronsjo's remarks may be considered an overstatement, however, as there are a number of particular equations for which it is feasible to evaluate derivatives, for example:

1. Differential equations where the given equation may be used to generate values of the derivatives from the function values.

2. Functions for which analytical differentiation is straightforward; polynomials are the most obvious such case. The methods of Newton and Laguerre have been used with considerable success in this context [46]. When closely-related functions (e.g. sine, cosine, exponential) appear in the function and its derivatives, the computation of the latter can be quite economical.

3. Functions defined in the form of integrals.

Wilkinson has pointed out [46] that in the region of a simple root the relative error in the computed derivative will be smaller than in the function value. Hence a derivative method (such as Newton's) will have the advantage over interpolation methods (such as Muller's) of stability within the zone of indeterminacy. It may then be advisable to choose a derivative method in cases where this is feasible and a high degree of accuracy is desired.

Attention is being given to the development of methods which replace the derivative by some suitable approximation. This will involve interpolation and hence, for a single point method, requires use of memory. This approach may yield formulae which are already in common use; for example, Newton's method with the gradient approximated by the straight line through the points $(x_{i-1}, f_{i-1})$ and $(x_i, f_i)$ must clearly yield the secant method. Some new formulae have, however, been produced from such considerations. In addition, Dahlquist and Bjorck [13] remind us that methods which are mathematically equivalent are not necessarily numerically equivalent in that the behaviours of rounding errors may be quite different. The introduction of the memory requirement also increases the risk of instability as with certain predictor-corrector methods for solving ordinary differential equations.

## Direct Interpolation Methods

Many of the non-derivative methods currently in use consist of fitting a simpler function of specified form $g(x)$ through the point $(x_i, f_i)$ and one or more previously calculated data points. Solution of the equation $g(x) = 0$ then provides, in most cases, an improved estimate of the required root. With these objectives we can define a large class of single point methods with memory. The following descriptions cover interpolation methods which are widely used:

### The Secant Method

Let $x_{i-1}$ and $x_i$ be two successive distinct estimates of the root. The straight line through $(x_{i-1}, f_{i-1})$ and $(x_i, f_i)$ has equation

$$g(x) = \left(\frac{x - x_{i-1}}{x_i - x_{i-1}}\right) f_i + \left(\frac{x - x_i}{x_{i-1} - x_i}\right) f_{i-1}$$

and solution of the equation $g(x) = 0$ yields the next estimate

$$x_{i+1} = x_i - \left(\frac{x_i - x_{i-1}}{f_i - f_{i-1}}\right) f_i$$

$x_{i+1}$ may be inside or outside the interval $(x_{i-1}, x_i)$; these will be referred to as interpolation and extrapolation steps respectively. Householder [22] shows that the asymptotic behaviour of the algorithm is dependent upon the signs of $f$ and $f''$ in the following manner:

If $f_{i-1}$ and $f_i$ both have the same sign as $f''$, the

subsequent iterations will all be extrapolations (Fig 3.1), otherwise the pattern consists of cycles of one extrapolation followed by two interpolations (Fig 3.2).

Fig 3.1

$f_i'' > 0$
$f_i > 0$
$\Big\} \; i = 0, 1, 2, \ldots$

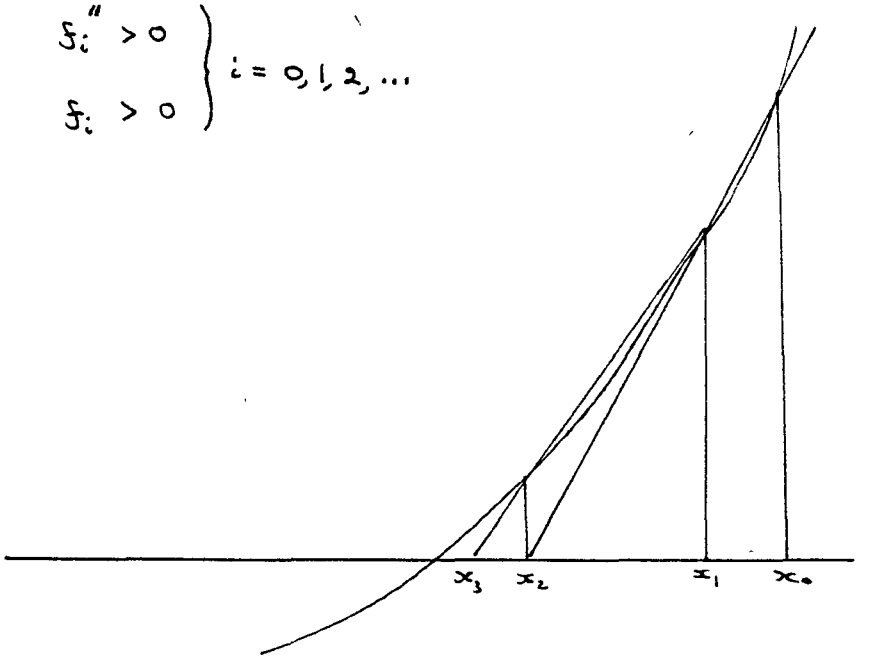
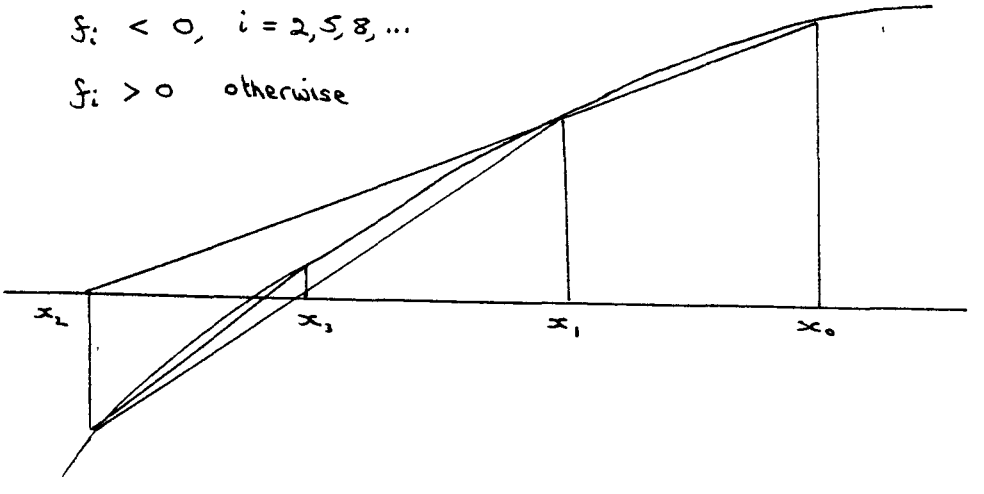



Fig 3.2

$f_i'' < 0, \quad i = 0, 1, 2, \ldots$

$f_i < 0, \quad i = 2, 5, 8, \ldots$

$f_i > 0$ otherwise



Ostrowski [35] proves, using interpolation theory and the mean value theorem, that the order of convergence of the secant method is $(1 + \sqrt{5})/2 = 1.618$. Since the method requires only one new function evaluation at each iteration subsequent to the first, the Traub and Ostrowski efficiency indices are each approximately 1.618 which is superior to the Newton method. A non-rigorous derivation of the order of convergence, based on the methods of Acton [1] and Kronsjo [25], is given in Appendix A.

The Regula Falsi Method

This method has a similar formulation to the secant method but consists entirely of interpolation steps. At each stage we use the point $(x_i, f_i)$ and the most recent previous iterate which produced a function value of opposite sign to $f_i$. Ultimately we are required to retain one end point throughout and this reduces the order of convergence to linear. The advantage over the bisection method is not then substantial so the regula falsi method is rarely used without modification.

Muller's Method

This direct interpolation method consists of fitting the unique parabola $g(x)$ through the points

$$(x_{i-2}, f_{i-2}),\ (x_{i-1}, f_{i-1}) \text{ and } (x_i, f_i)$$

In his original formulation of the method, Muller employed Lagrangian interpolation to give

$$g(x) = \frac{f_i(x - x_{i-1})(x - x_{i-2})}{(x_i - x_{i-1})(x_i - x_{i-2})} + \frac{f_{i-1}(x - x_i)(x - x_{i-2})}{(x_{i-1} - x_i)(x_{i-1} - x_{i-2})}$$
$$+ \frac{f_{i-2}(x - x_i)(x - x_{i-1})}{(x_{i-2} - x_i)(x_{i-2} - x_{i-1})}$$

but Traub [43] suggested that use of the Newtonian interpolation formula

$f_i + f[x_i, x_{i-1}](x - x_i) + f[x_i, x_{i-1}, x_{i-2}](x - x_i)(x - x_{i-1})$

where $f[x_i, x_{i-1}]$ and $f[x_i, x_{i-1}, x_{i-2}]$ denote the first and second divided differences respectively, will simplify the solution of the quadratic and will also reduce the amount of computation necessary to evaluate the next estimate. Thus we obtain

$$x_{i+1} = x_i - \frac{2f_i}{w + \{w^2 - 4f_i f[x_i, x_{i-1}, x_{i-2}]\}^{1/2}} \qquad \text{(i)}$$

where $w = f[x_i, x_{i-1}] + (x_i - x_{i-1}) f[x_i, x_{i-1}, x_{i-2}]$

Iteration hence proceeds using the three most recent values of x at each stage. Computer routines which use the Muller method generally adopt the Traub formula (i). This formulation also ensures that the selected root of the quadratic is always the one closer to $x_i$.

It can be shown that the order of convergence of the Muller method is approximately 1.84. Three starting values are needed but each successive iteration requires one function evaluation only so that the efficiency index is again equal to the order of convergence.

A frequently cited property of the Muller method is its ability to produce complex iterates from purely real starting values. This can be useful in a preliminary search for complex roots but, conversely, may involve the use of complex arithmetic when only real roots are sought. Small imaginary parts may be suppressed in these circumstances by setting the square root term in (i) to zero (for example in a program by Barrodale and Wilson [6]). It should be noted, however, that computation of complex roots is not confined to Muller's method. Provided we allow a complex starting value, methods such as secant or Newton can also converge to a complex root.

Higher Order Polynomial Interpolation

Theoretically it would be possible to fit polynomials of higher degree, but the potential difficulties of solving the resulting equation and selecting the appropriate root are prohibitive. In addition, the order of convergence of such methods could not exceed two, and the consequent reduction in the number of function evaluations would be slight [46]. Since the occurrence of instability is frequently associated with the use of the memory facility, this would constitute a further disadvantage of higher-order methods of this type.

## Interpolation by Rational Functions

The idea of approximation using rational functions is mentioned by Ostrowski [35] and is developed in greater detail by Jarratt and Nudds [24].

Three point rational interpolation uses a function of the form

$$g(x) = \frac{x - A}{Bx + C}$$

so that the equation $g(x) = 0$ has the unique solution

$$x_{i+1} = A$$

$$= x_i - \frac{(x_i - x_{i-1})(x_i - x_{i-2}) f_i (f_{i-1} - f_{i-2})}{(x_i - x_{i-1})(f_{i-2} - f_i) f_{i-1} - (x_i - x_{i-2})(f_i - f_{i-1}) f_{i-2}}$$

The point $x_{i-2}$ is then rejected and the next iteration uses the points $x_{i-1}$ , $x_i$ , $x_{i+1}$ .

It is convenient for practical implementation to use Thiele's interpolation formula which gives

$$x_{i+1} = x_i - f_i [(x_i - x_{i-1}) + (g_i - f_i) h_i] / g_i$$

where $g_i = \frac{x_i - x_{i-1}}{h_i - h_{i-1}} + f_{i-1}$ and $h_i = \frac{x_i - x_{i-1}}{f_i - f_{i-1}}$

are the second and first reciprocal differences respectively [31].

For simple roots the rate of convergence of three point rational interpolation is the same as that of the Muller method but the formulation is simpler and complex arithmetic is not involved in the determination of real roots. The method also has the advantage over polynomial interpolation of more successful performance in the region

of a simple pole of the function. On the other hand, the convergence rate for multiple roots is only linear which compares unfavourably with Muller.

The method may be generalized by using a polynomial of higher degree as the denominator. Again, the improvement in speed of convergence is not sufficient to warrant the use of such formulae unless the function is exceptionally expensive to evaluate. If a higher degree polynomial is incorporated in the numerator, the solution of the equation $g(x) = 0$ will no longer be unique and such methods are unlikely to be convenient for practical purposes.

## Inverse Interpolation Methods

For a continuous function $f(x)$ having a simple root or a root of odd multiplicity there exists an interval $[a,b]$ containing the root such that an inverse function $f^{-1}$ can be defined as follows:

Let $y = f(x)$ where $x \in [a,b]$

then $f^{-1}(y) = x$ with $y \in [f(a),f(b)]$

At the root $y = f(x) = 0$ so that $x = f^{-1}(0)$.

Hence, if an approximation $g(y)$ to the inverse function $f^{-1}$ can be found, an estimate of the root can be obtained by evaluating the interpolating function at zero. It is not then necessary to solve a polynomial equation.

## Inverse Linear Interpolation

The straight line through the points $(f_{i-1}, x_{i-1})$ and

$(f_i, x_i)$ may be written:

$$g(y) = \left(\frac{y - f_{i-1}}{f_i - f_{i-1}}\right) x_i + \left(\frac{y - f_i}{f_{i-1} - f_i}\right) x_{i-1}$$

Putting $y = 0$ gives

$$g(0) = x_{i+1} = x_i - \left(\frac{x_i - x_{i-1}}{f_i - f_{i-1}}\right) f_i$$

which is the secant method as before.

Inverse Quadratic Interpolation

Interpolation using the three points $(f_{i-2}, x_{i-2})$, $(f_{i-1}, x_{i-1})$ and $(f_i, x_i)$ gives:

$$g(y) = \frac{x_i (y - f_{i-1})(y - f_{i-2})}{(f_i - f_{i-1})(f_i - f_{i-2})} + \frac{x_{i-1} (y - f_i)(y - f_{i-2})}{(f_{i-1} - f_i)(f_{i-1} - f_{i-2})} + \frac{x_{i-2} (y - f_i)(y - f_{i-1})}{(f_{i-2} - f_i)(f_{i-2} - f_{i-1})}$$

$$\text{so } g(0) = x_{i+1} = \frac{f_{i-1} f_{i-2} x_i}{(f_i - f_{i-1})(f_i - f_{i-2})} + \frac{f_i f_{i-2} x_{i-1}}{(f_{i-1} - f_i)(f_{i-1} - f_{i-2})} + \frac{f_i f_{i-1} x_{i-2}}{(f_{i-2} - f_i)(f_{i-2} - f_{i-1})}$$

which is usually used in the form:

$$x_{i+1} = x_i - \frac{f_i}{f[x_i, x_{i-1}]} + \frac{f_i f_{i-1}}{(f_i - f_{i-2})} \left(\frac{1}{f[x_i, x_{i-1}]} - \frac{1}{f[x_{i-1}, x_{i-2}]}\right)$$

The order of convergence for simple roots is of the same order as for the corresponding direct method i.e. 1.84 approximately. The principal advantage over the direct method is the simpler formulation and the uniqueness of each estimate. It is also convenient to avoid complex arithmetic in the determination of real roots.

Wilkinson [47] finds, however, that inverse methods are not generally satisfactory for the eigenvalue problem, particularly in cases of multiple roots (clearly roots of even multiplicity cannot be determined by such means as the inverse function is not uniquely defined in any region containing the root)

## 3.3 MULTI-POINT METHODS

Methods of this type involve the use of two or more previously unused data points at each iteration and are being investigated with a view to reducing dependence on derivative values in addition to increasing efficiency. In particular, such methods will not be subject to the restriction that the order cannot exceed (n + 1) where n is the number of function evaluations required for each estimate. The multi-point methods which have been formulated are dependent upon approximations to derivatives by function values and/or derivatives of lower order and are based on one or more single point methods. When one method only is used, a recursive formula may be set up which is simple to program. For example, Newton's method may be employed as follows:

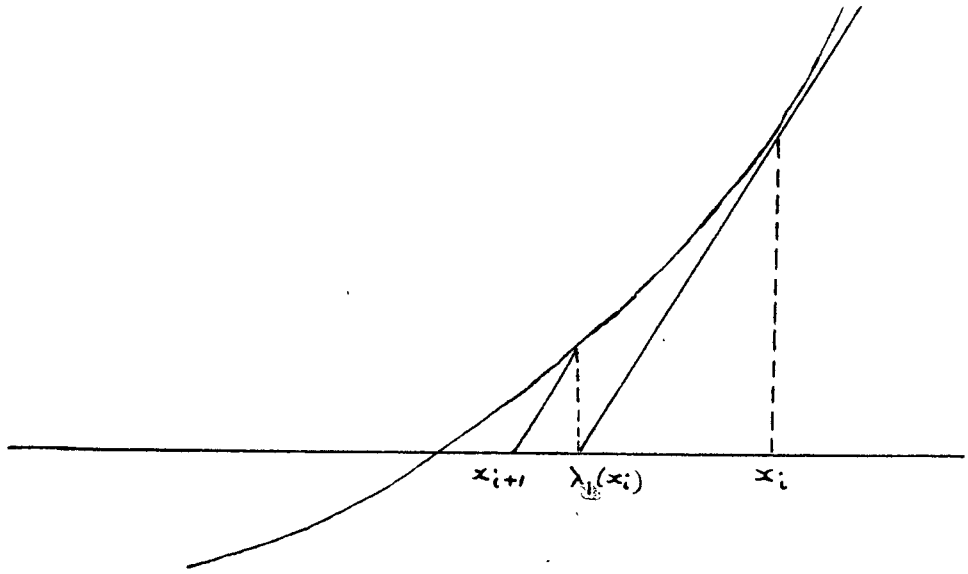
Let $\lambda_0 = x_i$ and $x_{i+1} = \lambda_p(x_i)$

where $\lambda_j(x_i) = \lambda_{j-1}(x_i) - \dfrac{f[\lambda_{j-1}(x_i)]}{f'(x_i)}$

When p = 1, this reduces to the standard Newton method. Each successive value of p defines a line parallel to the

tangent at the point $(x_i, f_i)$. Fig 3.3 illustrates the case p = 2.

Fig 3.3



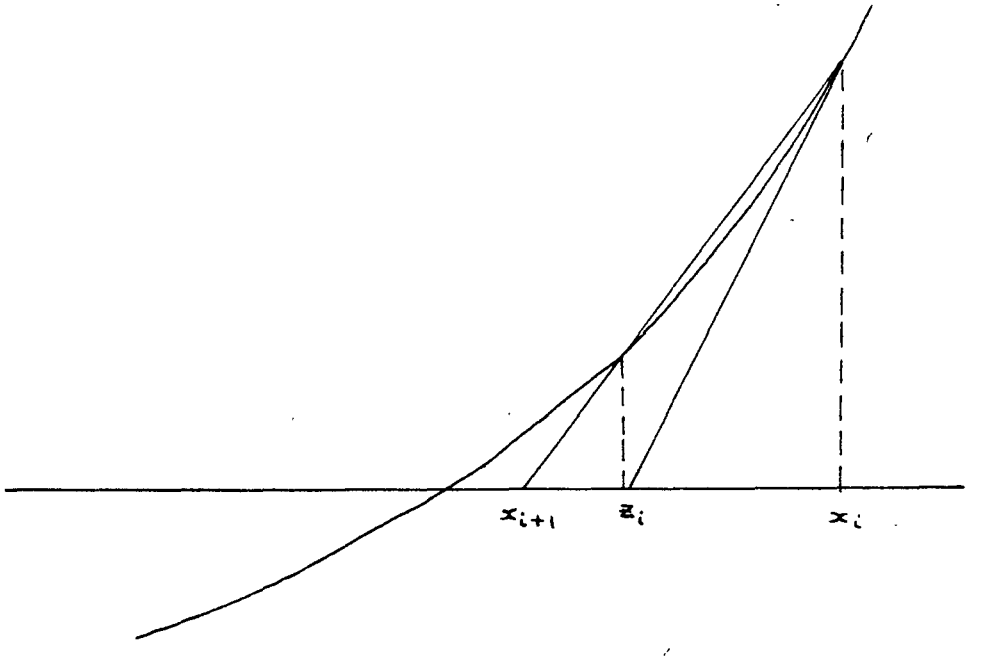

The method is of order 3 for a simple root and requires two evaluations of the function and one of its first derivative at each iteration. The computation can be arranged as a series of iterations with a fixed small value of p or as one loop with p increasing until the required degree of accuracy is obtained.

The composition of two different iterative methods is illustrated by the Newton-secant method viz:

$$z_i = x_i - \frac{f(x_i)}{f'(x_i)} \quad \text{and} \quad x_{i+1} = z_i - \frac{f(z_i)(z_i - x_i)}{f(z_i) - f(x_i)}$$

Here $z_i$ is found by Newton's method and is then combined with $x_i$ by the secant method as shown in Fig 3.4.

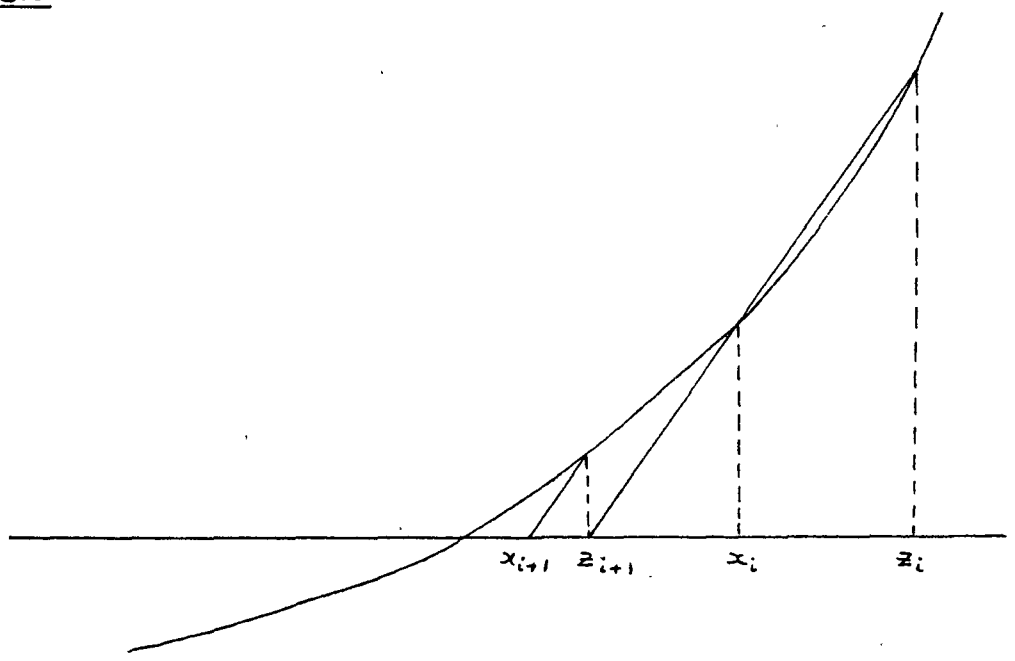
Fig 3.4




The order and number of function evaluations per iteration are the same as for the previous example.

For each of the above examples the Traub efficiency index $p/n = 1$ but this can be improved considerably. It has been conjectured by Kung and Traub [27] that a multi-point iterative method without memory based on n function evaluations per iteration has optimal order $2^{n-1}$ and such methods have indeed been constructed. The introduction of a memory facility can give further improvement. A simple example of this type is the composition of secants method illustrated in Fig 3.5.

Fig 3.5




Each iteration uses the two points $(x_i, f(x_i))$ and $f(z_i, f(z_i))$ thus:

Let $$\Delta_i = \frac{f(z_i) - f(x_i)}{z_i - x_i}$$

then define $$z_{i+1} = x_i - \frac{f(x_i)}{\Delta_i}$$

and $$x_{i+1} = z_{i+1} - \frac{f(z_{i+1})}{\Delta_i}$$

The order of the method is $(1 + \sqrt{2})$ and two function evaluations are required at each iteration, so the Traub efficiency index is approximately 1.21. Multi-point methods do not feature prominently in published programs to date, but their favourable rate of convergence would indicate that they are worthy of development. Some experimental observation of accumulated rounding errors would also be helpful as the number of computations per

iteration will necessarily be somewhat greater than for each of the individual algorithms on which the formulation is based.

## 3.4 MULTIPLE ROOTS

The likelihood of ill-conditioning in the case of multiple roots is apparent from the flatness of the curve in the vicinity of such a root; it would hence be optimistic to expect the same order of accuracy as might be attained for a simple root. In such circumstances we must be prepared, therefore, to accept a larger region of indeterminacy or resort to double precision arithmetic. The order of convergence of an iterative method is also generally reduced when the root is not simple; for example, in the case of a double root Newton's method has linear convergence whilst the order of convergence of the Muller method is about 1.23, so that Newton's method is a particularly poor choice. The case of a multiple root of even multiplicity is further complicated by the fact that there is no change in sign of the function on either side of the root.

If a is an m-fold zero of the function f(x) we have that $f(x) = (x-a)^m g(x)$ where $g(a) \neq 0$. The function

$$u(x) = \frac{f(x)}{f'(x)}$$

then has a simple root at x = a and can be used in place of f(x) provided that the derivative $f'(x)$ can be found.

The functions $f^{(m-1)}(x)$ and $f^{1/m}(x)$ will also have simple zeros at this point.

For all these adapted functions we have the problem of determination of derivatives and/or the multiplicity of the root. Furthermore, the function evaluations will generally involve more computation than for the original function.

In the absence of a priori knowledge of the multiplicity m, Traub [43] shows that

$$m = \lim_{x \to \alpha} \frac{\ln|f(x)|}{\ln|u(x)|}$$

u(x) being defined as above. Thus, if a derivative method is being used to determine the root, values of the ratio

$$\frac{\ln|f(x)|}{\ln|u(x)|}$$

may be output after each iteration until convergence is apparent and a switch made to an iterative formula appropriate to m. When it is not practicable to determine m, selection of the best algorithm is difficult; indeed Traub conjectures that:

> "It is impossible to construct an optimal iterative formula which does not depend explicitly on m and whose order is multiplicity independent."

The above remarks also apply to some extent to closely packed roots which should be regarded with similar caution and may constitute a more important practical problem.

## 3.5 SELECTION OF METHODS FOR TESTING

The specific programs to be discussed in the next chapter have been selected on the basis of their suitability for use with a wide range of functions (including transcendental) and their potentiality for development to cover cases of complex roots. As some of the examples to be considered arise from eigenvalue problems, attention will be given to the capacity of the chosen routines to cope with the special features of such functions. For these reasons, derivative methods will not be considered further; the computational difficulties and amount of work involved, particularly in association with determinants, effectively restricting their usefulness to polynomial equations.

Published programs for the solution of equations are of two distinct types:

1. Those which guarantee convergence for roots of continuous functions. In return for such guarantee, of course, the user must be able to supply a valid interval for each of the required roots (i.e. an interval (a,b) such that f(a) and f(b) are of opposite

signs and which is known to contain no other roots)

2. Methods which require only one starting value and which will search for a number of roots with or without further intervention from the user. The requirement for initial estimates is clearly less stringent, the responsibility for this search being transferred to the machine. It is evident that failure will be a far more frequent occurrence with such routines, but since many functions arising in practice cannot be fully analysed theoretically, the inclusion of such programs must be regarded as vital.

The bisection method will satisfy the criteria for type 1 but its convergence rate is too slow for it to be practicable as the sole method. It is, however, frequently used in conjunction with other methods in order to retain the necessary interval for the root. The other algorithms to be considered may all be classified as interpolation methods of types described herein.

---

# CHAPTER 4

## SOME TEST RESULTS FOR REAL ROOTS

The selection of algorithms for testing has been confined to those which are available as fully documented user programs in order that comment may be made on features of communication with the user. Six programs are considered, three of which are "interval methods" as defined in the previous chapter, the remainder being "search methods" which may require only one initial estimate for all the required roots. Particular attention will be given to the state of development of the latter type because of its potential use in the determination of complex roots. Direct comparison of these two types of method is difficult as the search routines must in many cases incorporate suppression of known roots; this is unnecessary where a separate interval is supplied for each root. The tolerance criteria offered are also markedly different. Choice of a suitable routine for a given problem will be governed by the availability of initial estimates and the relative importance of reliability and efficiency.

The results quoted in this chapter were obtained using the DEC-10 machine at Middlesex Polytechnic, except where otherwise stated.

## 4.1 THE SELECTED PROGRAMS

### Interval Methods

#### A) NAG Fortran Library Routine C05AZF [33]

This program implements a modified version of the procedure "zeroin" due to Bus and Dekker [10]. This algorithm is based on the secant method but seeks to retain a valid interval for the root at each stage in order to guarantee convergence. If the asymptotic behaviour is of the type illustrated in Fig 3.1 this forces the adoption of a regula falsi procedure with consequent deterioration in the rate of convergence. To remedy this shortcoming a scheme is devised which retains at all stages two points, b and c, having function values of opposite signs. The secant method is applied using the best currently available estimates a and b (in the sense of smallest function values) where $|f(a)| > |f(b)|$. This estimate is accepted if it lies in the current interval [b,c] and is closer to b than the mid-point of [b,c]. Otherwise bisection is applied to the interval [b,c]. A further modification introduces a rational interpolation step in the case of two successive estimates being very close together whilst the interval [b,c] remains

unacceptably large. This accelerates the process in some cases of slow convergence and the authors claim particular success with zeros of odd multiplicity.

NAG routine C05AZF employs reverse communication and the documentation provides a sample calling program (a direct communication routine is also available). All the convergence criteria are based on interval length, but provision is made for termination if the computed function value is zero. Another error indicator allows for possible detection of a pole of f(x) in the given interval.

B) Subroutine ZEROIN - Forsythe, Malcolm and Moler [17]

This is a FORTRAN version of an ALGOL procedure by Brent [8]. The basic algorithm is similar to that of Dekker but inverse quadratic interpolation is employed whenever the points a,b and c (as defined above) are distinct (i.e. extrapolation steps); rational interpolation is not used. We should expect the more frequent use of a higher order interpolation formula to reduce the number of function evaluations required for "well-behaved" functions but Bus and Dekker [10] have found that Brent's method is less successful in the region of a high order inflexion point.

The implementation of Brent's algorithm used here incorporates some safeguards against underflow and sets a mixed absolute and relative tolerance condition of the form 4.0 EPS|x| + TOL where EPS is the relative machine precision and TOL is the user-supplied tolerance for final interval length. The routine does not use reverse communication so it was necessary for these tests to incorporate a further output parameter to count the number of function evaluations. The point b is automatically accepted as a root if the computed value of f(b) is zero. The final function value is not included in the output parameters; the user may consequently need to exercise caution in the interpretation of results.

C) N.P.L. Algorithms Library, Real Procedure ZERO - Cox and Lehrian [11]

Let the most recent valid interval for a root be [a,b] where $|f(b)| < |f(a)|$ and let m be the mid-point of [a,b]. The algorithm implemented in this ALGOL60 procedure uses rational interpolation/extrapolation where this produces an estimate within the interval [m,b], otherwise bisection is used.

An absolute tolerance for the root is set by the user who is also required to supply a maximum number of iterations for each root, a value of 25 being recommended. If this limit is reached the best currently available estimate is output; such provision is important when reverse communication is not used as it prevents the total

loss of useful information. Again, an estimate x will be accepted as a root if the computed value of f(x) is zero but otherwise it is left to the user to examine the magnitude of f(x).

The use of rational interpolation in place of the secant method may be expected to produce faster convergence than the Bus and Dekker method. The program was found, however, to be not completely portable and some rearrangement was found to be necessary for use on the DEC10 which incorporates a one-pass ALGOL compiler, this necessitating a re-ordering of the procedures.

## Search Methods

### D) Function ROOT1 - Gaston H. Gonnet [18]

The purpose of Gonnet's paper "On the Structure of Zero Finders" is to demonstrate the advantages of a reverse communication procedure and to present a FORTRAN function subprogram ROOT1 which implements this approach to zero finding. A return to the calling program after each iteration allows the user considerable flexibility; for example, he is able to set a maximum number of function evaluations and control the stopping criteria. Such input parameters can be adjusted between iterations if they appear to be unsatisfactory. It may also be possible for the user to detect difficulties and arrange for termination of execution thus avoiding waste of time in performing unhelpful iterations.

Gonnet does not provide a sample calling program for his routine, although he gives an outline of the procedure to be followed; reverse communication will inevitably impose greater demands upon the user than the direct method. Appendix B is a copy of the program used for the tests herein and is designed for interative use at a terminal. A root suppression procedure has been incorporated to facilitate the calculation of several roots from one initial estimate and each root accepted is used as a "stepping-off" point for the next search.

The algorithm uses Muller's method where possible; failing that, the secant method and then bisection are attempted. ROOT1 returns the next estimated value of x and the user must then compute the next function value and test for acceptability before returning for further iterations if desired. The user is required to provide one initial estimate $x_0$; at the second entry ROOT1 returns the estimate $x_0 + f(x_0)$, the next iteration uses the secant method and thereafter the general scheme oulined above is followed. A default value of 30 is set for the maximum number of iterations per root, but this can clearly be reduced if desired. After 30 iterations have been performed the bisection method is used exclusively whenever a sign change interval has been detected as the function is then considered likely to possess features of difficulty.

Other useful features of ROOT1 include instructions for conversion to double precision, the calculation of an estimated derivative in certain circumstances and the provision of several error indicators. It also provides the length of the interval of uncertainty whenever such an interval is available. In addition to the freedom of reverse communication, the author claims a favourable comparison with other algorithms in respect of the number of function evaluations over a variety of functions.

E) A FORTRAN Program for solving a non-linear equation by Muller's method - I.Barrodale and K.B.Wilson [6]

Subroutine ROOTS together with the function deflation subroutine TEST is designed to calculate the required number of real and/or complex roots with or without user-supplied initial estimates and uses the Traub version of Muller's method throughout. The three starting values for the algorithm are taken as $x_0 - 0.5$, $x_0$ and $x_0 + 0.5$ where $x_0$ is the user's estimate of the root sought; if $x_0$ is not supplied, it is set to zero. The "stepping-off" points for second or subsequent roots are supplied by the user or, by default, are set to zero. Special features include a halving of the step length in cases where divergence is indicated by a large function value and a modification to the current estimate in "flat" regions of the curve in order to improve the efficiency for multiple roots.

Two convergence criteria are offered viz. relative change in x values and absolute function value. ROOTS provides default values of $0.5 \times 10^{-5}$ and $1.0 \times 10^{-6}$ respectively for these, values which may necessitate an amendment to the data statement, particularly when the program is to be implemented in double precision (a subprogram is provided for this purpose). A further change to the root-finding routine is required if the user wishes to extract the numbers of function evaluations taken for each root since this is calculated but not included in the parameter list.

Failure is indicated in some cases of invalid input and also when the user- supplied maximum number of iterations is exceeded. The direct communication approach adopted is straightforward to use but could result in loss of valuable information in cases of failure (other than maximum iterations in which case the best available estimate can be output). In particular, if the number of roots requested exceeds the number which can be found by the algorithm, an overflow failure may be anticipated.

F) N.P.L. Algorithms Library FORTRAN Subroutines RTFS1C/Z - S.J.Hammarling, P.D.Kenward and H.J.Symm [20]

These reverse communication routines are the most flexible and ambitious of those considered herein. They are designed for the calculation of complex roots, although all the computation is performed using real vectors so that the double precision version RTFS1Z may be used when double precision complex arithmetic is not implemented. The original version of the program used Traub's version of Muller's method only; an option of three-point rational interpolation has now been included as a result of favourable experience with the procedure ZERO of Cox and Lehrian (described in C above). In both cases linear interpolation is attempted if the three point method would cause irrecoverable overflow. A parameter list for RTFS1C/Z has been included in Appendix C as an indication of the scope of the program and the consequent demands upon the user. These subroutines are part of an extensive library and call upon a number of other subroutines; in particular routines are available which allow a close check to be kept on possible sources of overflow. Storage and timing requirements may be somewhat greater than for self- contained routines, but this should only be significant for very simple functions.

RTFS1C/Z seek to improve the reliability and accuracy of the search method by offering a versatile selection of options which enable the user to exploit to the full his knowledge of the function under consideration. Suppression of known roots is incorporated at each stage of the routine. The inclusion of the parameter BOUND is a guard against unexpectedly large estimates of the root and also provides an automatic termination when no more roots can be found. A perturbation is made to accelerate convergence when this appears to be slow. Other features of particular note are the provision for scaling of function values, the inclusion of an automatic stopping criterion in addition to the standard tests, a choice of methods for obtaining initial estimates for second and subsequent roots and a comprehensive set of values of the parameter INFORM to monitor progress and to indicate the acceptance criteria used for each root. The documentation includes a sample main program and further advice for users. A version RTFS1R which uses rational interpolation to calculate real roots only is in course of preparation. This will, in addition to the facilities offered by the complex routines, enable the user to detect and retain intervals for the roots whenever possible. (This was found to be a useful feature of program D above.)

## 4.2 EFFICIENCY

### Simple Roots

The numbers of function evaluations required to obtain simple roots of some straightforward functions are compared in table 4.1. In the case of program E, real roots only are requested and for program F the rational interpolation method has been used exclusively to ensure that all iterates are real. Single precision computation has been used throughout and the accuracy requirements have been chosen well within machine precision. The "interval" methods all offer termination criteria based on interval length; the tolerance level set here is an absolute value of $10^{-4}$ for the final interval length TOL except where otherwise stated. (Program B combines the relative and absolute criteria but the relative tolerance is comparatively small.) For the "search" methods, the criteria adopted have been $\varepsilon < 10^{-4}$ or $\eta < 10^{-6}$ where $\varepsilon$ is the relative tolerance $|(x_r - x_{r-1})/x_r|$ and $\eta = |f(x_r)|$, although it should be noted that program D will only invoke the former test if the routine has detected a valid interval for the root (in practice, for simple roots, this will usually be the case.) These tolerance levels will correspond roughly to an accuracy of four significant figures for the particular functions under consideration.

Where it has proved possible, the search routines have been instructed to find all the required roots from a single user-supplied estimate, suppression being employed after each root is found. When this has been found unsatisfactory a separate estimate has been used for each root; in such cases root suppression has not been necessary.

Functions which would be fitted exactly by any of the interpolation formulae used have been omitted.

On pages 66-68 are listed the functions tested, together with the numbers of roots requested, the initial intervals for programs A,B and C and the initial estimates for programs D,E and F.

Functions Tested in Table 4.1

| | FUNCTION | NO. ROOTS | INITIAL INTERVAL(S) | INITIAL VALUE(S) |
|---|---|---|---|---|
| 1. | $(x-1)(x-2)(x-3)(x-4)$ | 4 | (0.1,1.1) | 0.0 |
| | | | (1.1,2.1) | |
| | | | (2.1,3.1) | |
| | | | (3.1,4.1) | |
| 2. | $(x^2+x-1)(x^2-x-1)$ | 4 | (-2.0,-1.0) | 0.0 |
| | | | (-1.0,0.0) | |
| | | | (0.0,1.0) | |
| | | | (1.0,2.0) | |
| 3. | $x^3-2x-5$ | 1 | (2.0,3.0) | 2.0 |
| 4. | $x^3+x$ | 1 | (-0.5,1.0) | 1.0 |
| 5. | $x^2-(1-x)^n$ with | | | |
| | a) n=3 | 1 | (0.0,1.0) | 0.0 |
| | b) n=5 | 1 | (0.0,1.0) | 0.0 |
| | c) n=10 | 1 | (0.0,1.0) | 0.0 |
| 6. | $[1+(1-n)^4]x-(1-nx)^4$ | | | |
| | with a) n=1 | 1 | (0.0,1.0) | 0.0 |
| | b) n=4 | 1 | (0.0,1.0) | 0.0 |
| | c) n=8 | 1 | (0.0,1.0) | 0.0 |
| 7. | $x^3+10^{-4}$ | 1 | (-1.0,0.0) | 0.0 |
| 8. | $\frac{x+1}{x^2+2}$ | 1 | (-3.0,0.0) | 0.0 |

(cont.)

Functions Tested in Table 4.1 (cont.)

| | FUNCTION | NO. ROOTS | INITIAL INTERVAL(S) | INITIAL VALUE(S) |
|---|---|---|---|---|
| 9. | $\frac{x^2-x-1}{x^2-x+1}$ | 2 | (-1.0,0.0) | -1.0 |
| | | | (1.0,2.0) | 2.0 |
| 10. | $\frac{x^2-2}{(x-2)^2}$ | 2 | (-2.0,-1.0) | -1.0 |
| | | | (1.0,2.0) | 1.0 |
| 11. | $x-4\sqrt{x-1}$ | 2 | (1.0,2.0) | 1.0 |
| | | | (10.0,15.0) | 15.0 |
| 12. | $\exp(-x)-x$ | 1 | (0.0,1.0) | 0.0 |
| 13. | $2x\exp(-1)+1-2\exp(-x)$ | 1 | (0.0,1.0) | 0.0 |
| 14. | $(x-1)\exp(-nx)+x^n$ | | | |
| | with a) n=1 | 1 | (0.0,1.0) | 0.0 |
| | b) n=5 | 1 | (0.0,1.0) | 0.0 |
| | c) n=10 | 1 | (0.0,1.0) | 0.0 |
| 15. | $x\ln(x)-1$ | 1 | (1.0,2.0) | 2.0 |
| 16. | $\sin(x)-0.5$ | 4 | (0.0,1.0) | 0.0 |
| | | | (2.0,3.0) | |
| | | | (6.0,7.0) | |
| | | | (8.0,9.0) | |
| 17. | $\sqrt{x}-2\sin(x)$ | 1 | (0.1,1.0) | 1.0 |
| 18. | $\tan(x)-1$ | 1 | (0.0,1.0) | 0.0 |
| 19. | $\ln(x)-\cos(x)$ | 1 | (1.0,2.0) | 1.0 |
| 20. | $x-\cos\left(\frac{0.785-x\sqrt{1+x^2}}{1+2x^2}\right)$ | 1 | (0.0,1.0) | 0.0 |

(cont.)

Functions Tested in Table 4.1 (cont.)

Notes:

1. A number of the functions listed here are taken from the paper "A Comparison of Non-Linear Equation Solvers" by D.Nerinckx and A.Haegemans [34]. Function 20 is due to Kronsjo [25].

2. Functions 5a),b) and c) each have an inflexion point in [0,1]. Functions 6b) and c) each have one turning point and one inflexion point in [0,1]. Functions 14a),b) and c) are increasingly close to the x axis for increasing n.

3. For functions 6b), 6c) and 7 the absolute tolerances for the interval methods are $10^{-6}$,$10^{-8}$ and $10^{-5}$ respectively.

Table 4.1 Numbers of Function Evaluations for Simple Roots

(The figures given indicate the total number of evaluations for all the required roots in each case.)

| PROGRAM | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| FUNCTION | | | | | | |
| 1 | 32 | 30 | 31 | 29 | 27 | 27 |
| 2 | 33 | 29 | 30 | 26 | 23 | 32 |
| 3 | 7 | 6 | 7 | 6 | 6 | 5 |
| 4 | 7 | 7 | 8 | 10* | 8 | 8 |
| 5 a) | 7 | 6 | 7 | 8 | 7 | 5 |
| b) | 8 | 7 | 8 | 9 | 8 | 6 |
| c) | 8 | 8 | 9 | 11 | 11 | 5 |
| 6 a) | 9 | 8 | 7 | 8 | 7 | 5 |
| b) | 7 | 7 | 9 | 6 | 6 | 6 |
| c) | 6 | 6 | 8 | 5 | 5 | 5 |
| 7 | 16 | 16 | 13 | 19 | 11 | 8 |
| 8 | 9 | 8 | 8 | 7 | 8 | 8 |
| 9 | 16 | 14 | 13 | 11 | 12 | 12 |
| 10 | 17 | 15 | 12 | 11 | 14 | 13 |
| 11 | 14 | 14 | 14 | 13 | 14 | 9 |
| 12 | 7 | 5 | 6 | 6 | 6 | 5 |
| 13 | 7 | 6 | 6 | 7 | 6 | 5 |
| 14 a) | 7 | 7 | 6 | 7 | 5 | 5 |
| b) | 7 | 6 | 7 | 12 | 14 | 7 |
| c) | 8 | 7 | 8 | 15* | 12 | 9 |

(cont.)

Table 4.1 (cont.)

| PROGRAM | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| FUNCTION | | | | | | |
| 15 | 6 | 6 | 6 | 5 | 6 | 5 |
| 16 | 26 | 25 | 26 | 28 | 33 # | 32 |
| 17 | 7 | 7 | 8 | 5 | 8 | 9 |
| 18 | 8 | 7 | 7 | 8 | 8 | 7 |
| 19 | 6 | 6 | 6 | 6 | 6 | 5 |
| 20 | 5 | 5 | 6 | 7 | 8 | 8 |
| TOTAL | 290 | 268 | 276 | 285 | 280 | 251 |

* Underflow occurred but the correct solution was obtained.
( the DEC10 machine gives underflow warnings.)

# The smallest negative root was obtained in place of the root in the interval (8,9).

## Multiple and Closely-Spaced Roots

Results for a small selection of such functions (listed on page 72) are shown in table 4.2. The tolerance criteria adopted are indicated for each individual function. (Most of these necessitated a change to the default values set in program E as function values $\geqslant 10^{-6}$ did not yield satisfactory values for the roots.) The Muller version of program F has been included since, although complex arithmetic was used, for these functions only real roots were found.

Functions tested in Table 4.2

| FUNCTION | NO. ROOTS | INITIAL INTERVAL(S) | TOL | INITIAL VALUE | $\varepsilon$ | $\eta$ |
|---|---|---|---|---|---|---|
| 1. $x^n$ with | 1 | (-0.5,1.0) | $10^{-4}$ | 1.0 | 0.0 | $10^{-12}$ |
| a) n=3 | | | | | | |
| b) n=5 | 1 | (-0.5,1.0) | $10^{-4}$ | 1.0 | 0.0 | $10^{-20}$ |
| c) n=9 | 1 | (-0.5,1.0) | $10^{-4}$ | 1.0 | 0.0 | $10^{-36}$ |
| 2. $(x-1)^n$ with | | | | | | |
| a) n=3 | 1 | (0.0,4.0) | $10^{-4}$ | 0.0 | $10^{-4}$ | $10^{-12}$ |
| b) n=5 | 1 | (0.0,4.0) | $10^{-4}$ | 0.0 | $10^{-4}$ | $10^{-20}$ |
| c) n=9 | 1 | (0.0,4.0) | $10^{-4}$ | 0.0 | $10^{-6}$ | $10^{-36}$ |
| 3. $\frac{(x-1)^3}{x^2+1}$ | 1 | (0.0,4.0) | $10^{-4}$ | 2.0 | $10^{-4}$ | $10^{-12}$ |
| 4. $(3x-2)^3$ | 1 | (0.0,1.0) | $10^{-4}$ | 0.0 | $10^{-4}$ | $10^{-6}$ |
| 5. $\sin(x)-1+10^{-6}$ | 2 | (1.56,1.57) (1.57,1.58) | $10^{-6}$ | 1.5 | $10^{-6}$ | $10^{-8}$ |
| 6. $(x^2+1)\times$ $(x^2-2x+1-10^{-6})$ | 2 | (1.0,1.1) (0.9,1.0) | $10^{-6}$ | 1.0 | $10^{-6}$ | $10^{-8}$ |

Table 4.2 Numbers of Function Evaluations for Multiple and Closely-Spaced Roots

In the case of program F, (1) indicates the rational interpolation method and (2) the Muller method.

| PROGRAM | A | B | C | D | E | F | |
|---|---|---|---|---|---|---|---|
| FUNCTION | | | | | | (1) | (2) |
| 1 a) | 44 | 42 | 26 | 36 | 18 | 29 | 37 |
| b) | 45* | 34 | 47 | 63 | 22* | 48 | 76 |
| c) | 47* | 37* | 48 | >80 | 40* | >80 | >80 |
| 2 a) | 48 | 46 | 28 | 36 | 9 | 25 | 36 |
| b) | 47* | 42 | 47 | 63 | 19 # | 34 | 44 |
| c) | 51* | 44* | 58 | >80 | 14 # | 64 | >80 |
| 3 | 45 | 40 | 29 | 32 | 11 # | 25 | 16 |
| 4 | 41 | 39 | 25 | 39 | 17 | 27 | 34 |
| 5 | 26 | 29 | 23 | 9 | 13 | 22 | 12 |
| 6 | 25 | 32 | 31 | 11 | 17 | ** | 15 |
| TOTAL | 419 | 385 | 362 | >449 | 180 | >354 | >430 |

* Underflow occurred but the correct solution was obtained.

\# The accuracy of these results was comparatively poor - see following comments.

** Failed to find the second root; the first root being repeated.

## Comments on Results

### Interval Methods

For the modest accuracy levels set, higher order methods converge only slightly faster than the Bus and Dekker method; this superiority is somewhat more apparent in the examples of Table 4.2. In addition, the rational interpolation method of Cox and Lehrian manages to avoid the problem of underflow. The comparative inefficiency of Bus and Dekker for multiple roots is demonstrated by the successive iterates for the function $x^3$ [Appendix D, p.All]. This example shows every fourth iterate moving away from the root, this pattern of behaviour persisting throughout.

### Search Methods

In so far as comparison is possible, these methods require a similar number of function evaluations to the interval methods for simple roots, except in cases of polynomials where suppression of roots results in greater efficiency for the last two roots. The total number of function evaluations in Table 4.1 is rather less for rational interpolation than for the Muller methods; this advantage occurs principally for curves which are "flat" in the region of the root (functions 5,7 and 14). The superiority of rational interpolation becomes marked in cases of multiple roots, unless special provision is made in the Muller routine (as in program E). This would seem

to indicate that the asymptotic behaviour has not been reached; the situation would perhaps be different at higher accuracy levels. The progress of the Muller version (method 2) of program F is hampered by complex iterates and, in addition, the real parts may oscillate about the root. This behaviour is illustrated for the function $x^3$ in Appendix D (pp.A13-A16); by contrast rational interpolation (method 1) shows monotonic convergence to the root, all iterates being real. Observations at the National Physical Laboratory confirm that the convergence of rational interpolation is more rapid than that of Muller in the early stages. It also follows that failure to converge can manifest itself after a comparatively small number of iterations with the former method. The Muller method appears, however, to be more satisfactory for closely-spaced roots. In such cases root suppression frequently fails for the rational interpolation method, for example function 5 of table 4.2.

In cases of multiple or closely-spaced roots we expect ill-conditioning and consequently a lower degree of attainable accuracy than in the examples of Table 4.1. For the functions 1,2,3 of Table 4.2 this is counterbalanced to some extent by Wilkinson's comments regarding functions "in which the parameters are exact numbers requiring few digits for their representation". He goes on to state that "Even if rounding errors do occur at some stages of the computation, the fact that part of it is performed without error may lead to answers of

exceptional accuracy" [46]. This is the observed outcome for all routines except that of Barrodale and Wilson which achieves results consistent with the theoretically attainable accuracy for such functions. This would seem to be a consequence of the accelerating device employed for flat regions of the curve. It was noted, however, that setting the parameter REALRT to .FALSE. in program E produces numbers of iterations comparable with those achieved by the other routines. (The strategy when REALRT = .TRUE. is simply to set any square roots of negative numbers to zero.)

## 4.3 DIFFICULTIES ASSOCIATED WITH SEARCH ROUTINES

In addition to the functions listed in the previous section, a number of further tests were carried out. The "interval methods" were found to be very reliable throughout and the occurrence of underflow for some functions was not found to inhibit the achievement of correct solutions. Particular caution would seem to be required only in cases where the sign change interval contains a discontinuity of the function rather than a root; for instance, none of the routines tested warned of this situation for the function $\tan(x) - x$ and a root was claimed at $x = \pi/2$. Output of the function value is usually sufficient protection in such situations.

As might be anticipated, search routines are less reliable and performance is often very sensitive to choice of input parameters. The following discussion covers those aspects which appear to be crucial:

## Choice of Convergence Criteria

### Relative Error Test

Unlike absolute interval length, the relative error criterion is not always a good indicator of the accuracy of the result. The value to be set for $\varepsilon$ will depend, in part, upon the condition of the roots sought. This is particularly apparent for roots of high multiplicity such as function 2c) of Table 4.2 and the extreme example $(x-1)\exp(-1/(x-1)^2)$ illustrated in Appendix D (p.A17). Such cases are characterised by a slow convergence rate requiring an alternative method or some means of accelerating convergence even when $\varepsilon$ is not small. A potentially more serious problem is the possibility of accepting an invalid root on the basis of the relative criterion. As a test case the routines were instructed to find two real roots of the function $\exp(-x) - x$. In each case the valid root was found successfully in six iterations. Program D terminated with an error indicator after a further eleven iterations, but the other two programs each produced a further root. The erroneous results here were clear from the large function values. Appendix D (p.A17) shows the results for program F and illustrates how false roots may be eliminated by a

suitable reduction in the bound on values of x. In the tests conducted, failures of this type with the relative criterion have been few and always distinguished by large function values or arithmetic error warnings. Further indicators of false roots are an unexpected increase or decrease in the number of iterations per root or a move away from the region of interest. An example of such phenomena is provided by the function:

$$\sum_{r=1}^{20} \frac{(2r - 5)^2}{(x - r^2)^3}$$

(hereinafter referred to as Brent's function). This is used as a test function by Cox and Lehrian [11] and has nineteen real roots, lying in the intervals $r^2 < x < (r + 1)^2$, $[r = 1,2,\ldots,19]$. Such a function poses particular problems for search routines as the roots are separated by poles of the function and, in addition, the curve is flat in the region of each root. The best results were obtained by program F using a relative criterion [Appendix D,p.A18]. Two invalid roots are claimed, the first near the pole at x = 289 and the second beyond the valid range; in each case the change in the number of iterations is marked. The example also serves as a reminder that results obtained following an underflow message should not be accepted without further verification. The provision of an absolute interval of uncertainty by program D is a further safeguard when using the relative criterion; on the other hand, failure to detect such an interval prevents the use of this criterion

and the result is frequently an unnecessarily large number of iterations.

Function Value Tests

If the function value test for convergence is to be invoked it is essential to have some a priori knowledge of the magnitude of such values in the region of the root. When function values are small throughout a large interval we are in danger of accepting a poor approximation to the root. Conversely, if function values are large compared with relative changes in x we may demand an unattainable accuracy if $\eta$ is set too small. To illustrate the above remarks, let

$$f(x) = ( x - 0.1 )^4 ( x - 0.2 )^3 ( x - 0.3 )^2 (x - 0.4)$$

and $$g(x) = \prod_{r=1}^{20} ( x - r )$$

and contrast the magnitudes of f(x) and g(x) when x is a root to within one unit in the fourth significant digit:

$$f(0.09999) = 2.524 \times 10^{-23}$$

$$g(0.9999) = 1.217 \times 10^{13}$$

There are also many instances in which small function values do not indicate the presence of a root; for example, with certain choices of initial estimate, programs D and E both claimed a negative root for Brent's function with $\eta = 10^{-6}$. Program F offers two function value criteria based on values of the original and suppressed functions respectively. The former test was used exclusively for this study to enable comparison with other routines. Examination of the suppressed function

values also has the potential disadvantage that the behaviour of this function is not likely to be as well known as that of the original function. It must be concluded that function value tests are frequently unreliable when used as the sole criterion for convergence.

Automatic Stopping

An automatic stopping criterion of the type described in Chapter 2 is implemented in program F only. Additional safeguards are also included which seek to ensure that a convergence pattern has been established. The convergence test will not be brought into operation until $|x_r - x_{r-1}| < tol_1 |x_r|$ and the authors suggest that $tol_1 = 0.05$ is "normally adequate". The object of the test is to ensure that results are of the maximum accuracy consistent with the condition of the function and the number of digits employed in the computation. This was well reflected in the output for the functions of tables 4.1 and 4.2; simple roots being obtained to near machine accuracy (approximately eight digits) and multiple roots to a precision in accordance with the theory as indicated in Appendix E.

It was observed, however, that the automatic stopping criterion quite frequently produces totally incorrect roots which cannot be accounted for by ill-conditioning. Such extraneous roots occur particularly in cases where a number of roots are requested. The following may be cited as examples:

1. The function $f(x) = \sin(x) - 0.5$. Four roots were sought starting from the origin. Three roots were obtained correctly but the result 7.2941 was produced in place of the root 6.8068.

2. The function $f(x) = \prod_{r=1}^{20} (x-r)$. The explicit form of this polynomial has a number of extremely ill-conditioned roots [46], but this difficulty is not encountered with the factored form [Appendix E] which, nevertheless gave rise to a number of incorrect roots with $tol_1 = 0.05$ [Appendix D,p.A19]. The standard relative error criterion, however, produced all twenty roots correctly.

3. Automatic stopping may be used in conjunction with other tests and Appendix D (p.A20) shows the results for Brent's function with the option of automatic stopping or the standard relative error criterion. Nineteen roots are claimed but of those accepted on the basis of automatic stopping, only two are near to actual roots, whereas those obtained by the standard test are all of acceptable accuracy.

It is apparent that at present further evidence needs to be obtained before accepting roots on the basis of automatic stopping. The best procedure may prove to be use of automatic stopping to refine an estimate, having established by other means that a root is indeed present. Further investigation may also be required into appropriate values for $tol_1$. The user will not usually be able to anticipate the condition of the roots and if he wishes to know the accuracy of the results obtained it will be necessary to examine the convergence pattern displayed by the last few iterates.

## Initial Estimates

Good approximations to the roots are not always necessary for success when the function is a polynomial and the search can often be commenced from some distance away provided that the function values remain within machine capacity. The situation is less satisfactory, however, for other functions and many observed instances of failure can be attributed to poor starting values. When the first estimate is not sufficiently near to a root, the first iteration will produce a large increment; this may result in convergence to a root which is not the nearest to the starting point. For example, program D applied to Brent's function with $x_0 = 2.5$ produced the root near $x = 11$ rather than that near $x = 3$. When several solutions of an equation are required this tendency to "skip" roots may result in an incomplete picture as it can

prove difficult for the algorithm to "backtrack" in search of missing roots; in Appendix D (p.A18) program F misses a number of the smaller roots of Brent's function.

If there are no roots in the vicinity, poor starting values will often cause a move to a region of large values of x or f(x) or to a region in which the function is undefined, thus causing complete failure. The following are examples:

1. f(x) = (4x-7)/(x-2). The output in Appendix D (pp.A21-22) shows that very good first estimates are likely to be required when the root is close to a pole of the function. Cases of failure caused iterates of large magnitude, so that the imposition of a bound on x prevents an excessive number of iterations in program F.

2. f(x) = x ln(50x) + 0.005. This function has two close roots of small magnitude so that good estimates are likely to be required to prevent a move to negative values of x. All the programs obtained the larger root without difficulty starting from x = 1.0. Programs E and F also obtained the smaller root, despite an overflow in the case of program E, but program D failed with a large negative value of x.

Occasionally, a poor first estimate may give rise to the situation illustrated in Appendix D (pp.A23-25). Here the rational interpolation method is applied to the function $f(x) = x^3 - 2x - 5$ starting from $x = 0.0$ and a large number of iterations is performed before convergence to the real root is established. This may perhaps be attributed to the presence of complex roots in a neighbourhood of the starting point. The Muller version of the program performed much better for this example.

Muller and three point rational interpolation formulae each require three points for the first iteration. It follows that, if the user is to supply one estimate $x_0$ only, the routine will need to construct $x_1$ and $x_2$ from $x_0$. The procedure adopted in program D is to set $x_1 = x_0 + f(x_0)$ and then to use one iteration of the secant method. This has proved a serious shortcoming of the program since it presupposes that $f(x_0)$ is of a similar order of magnitude to $x_0$. If either of the quantities $x_0$ , $f(x_0)$ is negligible in comparison with the other, failure will result with either repeated argument or function values. This difficulty is particularly noticeable on second and subsequent roots as the function value will be small when the initial estimate is taken close to the root just found, for example, if

$f(x) = (x-0.1)^4 (x-0.2)^3 (x-0.3)^2 (x-0.4)$

then $f(0.101) = 1.1489 \times 10^{-17}$

$\Rightarrow 0.101 + f(0.101) = 0.101$ to machine precision and program D fails with a repeated argument message.

Program F calculates $x_1 = x_0(1.0 + 2.0\ step)$ where step may be chosen by the user, and $x_2 = (x_0 + x_1)/2.0$. This method was found to be much more satisfactory.

Problems Associated with the Requirement for Several Roots

When separate estimates are not available for each root subsequent to the first, a decision must be made concerning self-starting points for each search. Program F offers two alternatives to the user viz.

1. A point close to the root just found.

2. The complex conjugate of the root just found.

Program E sets $x_0 = 0.0$ by default.

All the routines tested gave good results for simple functions such as numbers 1,2 and 16 of Table 4.1. For functions 9,10 and 11 of this table it was found necessary to provide separate estimates for each root.

Particular problems must be anticipated with functions possessing singularities or undefined over a segment of the real line. It was found that the routines tested frequently failed or required a very large number of iterations after the first root in such circumstances. Widely spaced roots can create a further complication. Appendix D (pp.A26-A29) contains test results for the function $f(x) = x - 4\sqrt{x - 1}$ which has two real roots at $x = 1.07180$ and $x = 14.9282$ (each correct to 6 significant

figures). An attempt to take the square root of a negative argument causes failure or an incorrect solution with program D; the other two routines require an excessive number of iterations to obtain a second root. The results for program E with REALRT= .FALSE. and $x_0 = 1.0$ show that it can sometimes be helpful to conduct the search in the complex plane even though the required roots are both real. This tactic is less successful, however, when $x_0 = 15.0$.

Global convergence properties of iterative processes have not yet been fully examined theoretically. In practice, regions of convergence will depend also upon machine characteristics. Continuity of the function over the real line is not sufficient to guarantee the successful computation of all roots from a single starting point. Appendix D (pp.A30-A32) shows the results obtained with the functions $f(x) = x^{20} - 1$ and $f(x) = (x^2 - x - 1)/(x^2 - x + 1)$ respectively.

Table 4.3 illustrates some of the problems which may be encountered when a large number of roots is required. For each function, program F was instructed to seek twenty roots, the tolerance for relative error being $1.0 \times 10^{-6}$ in each case. In the case of function B, function values were calculated in the form $a \times 2^c$ (A separate routine for such scaling being included in the library.) The results obtained for function D reflected the moderate ill-conditioning at the ends of the range [46]. From the

limited data, no clear preference may be given to either method; the only apparent conclusion being the greater variability in the numbers of iterations with the Muller method. The order of root determination was in each case roughly monotonic but for function C a number of the later roots were skipped, particularly with method 1. For function D the behaviour of method 2 was more erratic in this respect.

In practical examples where several roots are required the exact number of roots in existence may be unknown. In such circumstances we would wish the program to indicate that as many roots have been found as is possible. Programs D and E can only terminate by means of an error condition; this may occur only after many superfluous iterations. The parameter BOUND in program F provides a natural termination in most cases as the suppression of all known roots usually causes iterates to become large in magnitude quite rapidly.

Table 4.3 Results for Functions with a Large Number of Roots

The Functions:

$$A = \prod_{r=1}^{10} ( x - r )$$

$$B = \prod_{r=1}^{20} ( x - 1/2^r )$$

$$C = \sin (x)$$

$$D = \cos (20 \cos^{-1} x)$$ [The Chebyshev polynomial of order 20]

| FUNCTION | METHOD | STARTING VALUE | NO.ROOTS FOUND | MEAN NO. ITERATIONS | STANDARD DEVIATION |
|---|---|---|---|---|---|
| A | 1 | 0.0 | 20 | 9.95 | 1.98 |
| | 2 | 0.0 | 20 | 8.65 | 5.18 |
| B | 1 | 1.0 | 20 | 17.4 | 5.51 |
| | 2 | 1.0 | 1 | 96 | |
| C | 1 | 3.0 | 9 | 8.11 | 2.09 |
| | 2 | 3.0 | 20* | 7.85 | 2.66 |
| D | 1 | 1.0 | 19 | 26.6 | 20.48 |
| | 2 | 1.0 | 10* | >41.6 | >30.89 |

* underflow occurred but the roots obtained were correct.

(cont.)

Table 4.3 (cont.)

The causes of failure were as follows:

Function B, method 2 - the first root was repeated.

Function C, method 1 - x exceeded the imposed bound of 100.0.

Function D, method 1 - one of the roots was found twice.

method 2 - the maximum number of iterations (1600) was exceeded.

## 4.4 CONCLUSIONS

Program F is noticeably superior to the other search routines tested in terms of reliability and scope. It also shows at least comparable efficiency except in cases of multiple roots where program E provides effective acceleration. The major factors governing this success would appear to be the inclusion of a parameter to impose a bound on argument values and the use of relative increments in setting "stepping-off" points. In its present state of development automatic stopping will probably be of use principally in refining estimates of roots obtained on the basis of other tests. Its reliability as the sole criterion for acceptance of roots remains in doubt.

Search routines generally do not eliminate the requirement for a good knowledge of the function. If this cannot be obtained analytically, a certain number of experimental function evaluations is likely to be required in order to fix appropriate initial estimates, bounds and convergence criteria. If a suitable interval is available for a root, a method which retains such an interval is likely to be the best choice.

Whilst rational interpolation worked more efficiently in many cases, there is some evidence that the Muller method will cope better with the situation of closely-spaced roots and may be capable of obtaining more roots when a large number are required. Both these points may be relevant to eigenvalue tracking problems.

Root searching from a single starting point is generally more successful for continuous functions so if possible any poles should be removed when formulating the function. Also any obvious scaling factors should be employed.

Despite the substantial advances outlined above, the complete solution of a general equation is still far from being an automatic procedure. In comparison with methods designed specifically for polynomial equations (e.g. Laguerre's method) a much greater amount of experimentation will be demanded of the user in his selection of input parameters.

---

# CHAPTER 5

# EIGENVALUE PROBLEMS

The standard eigenvalue problem is of the form $A\underline{x} = \lambda\underline{x}$ where A is a given n x n matrix with constant coefficients, $\underline{x}$ is an n x 1 vector and $\lambda$ a scalar. The requirement may be for all or selected eigenvalues with or without the corresponding eigenvectors $\underline{x}$. This situation has been studied extensively and a number of effective algorithms are available. Generalized eigenvalue problems involving several matrices $A_0, A_1, \ldots, A_r$ or a matrix $A(\lambda)$ whose elements are non-linear functions of $\lambda$ have received comparatively little theoretical treatment and, in the latter case, most existing algorithms have been developed to solve specific practical problems only.

The aim of this chapter is to indicate how eigenvalue problems can arise and to discuss the role of equation-solving techniques in obtaining numerical solutions to such problems. Consideration will also be given to special features of the equations thus obtained

and the consequent difficulties which may be encountered in the application of currently available algorithms. Complex roots will be sought in many eigenvalue problems, and this requirement may restrict the choice of appropriate methods and, in most cases, will substantially increase both the work-load and the risk of failure.

## 5.1 DIFFERENTIAL EQUATIONS AND EIGENVALUE PROBLEMS

The standard eigenvalue problem is encountered in a wide variety of practical situations including economic modelling, Markov processes and geometry in addition to mathematical physics. The generalized problems have, to date, arisen mainly from engineering applications. Wilkinson [48] states that:

> " the primary reason for the practical importance of the algebraic eigenvalue problem is its close relationship with the problem of determining the explicit solution of a homogeneous system of linear differential equations with constant coefficients."

Also of frequent occurrence are ordinary and partial differential equation problems involving a variable parameter $\lambda$. For given boundary conditions, solutions will exist for particular values of $\lambda$ only. The determination of such numerical solutions again gives rise to an eigenvalue problem. In this section the relationship between eigenvalue problems and differential equations will be outlined and mention will be made of relevant practical applications.

## The Form of Solution of Initial Value Problems for Ordinary Differential Equations

### a) Explicit First Order Systems

The simplest type of initial value problem may be expressed in the form

$$\frac{d\underline{x}}{dt} = A\underline{x} \text{ given that } \underline{x} = \underline{x}^{(0)} \text{ when } t = 0 \qquad \text{(i)}$$

where A is an nxn matrix with constant coefficients (real or complex).

Guidance on the form of solution is gained by consideration of the one-dimensional case $\frac{dx}{dt} = ax$ which is known to have the general solution $x = \alpha e^{at}$ where $\alpha$ is an arbitrary constant. If $x = x_0$ when $t = 0$, we then have the particular solution $x = x_0 e^{at}$.

This leads to a trial solution for (i) of the form $\underline{x} = \alpha \underline{q} e^{\lambda t}$ where $\underline{q}$ is a non-zero vector.

This will be a valid solution if and only if $\lambda \underline{q} = A\underline{q}$.

If A has n linearly independent eigenvectors ($\underline{q}_i$, i=1,2,...,n) this leads immediately to the general solution

$$\underline{x} = \sum_{i=1}^{n} \alpha_i \underline{q}_i e^{\lambda_i t}$$

In this case, if the matrix A has been diagonalized by a similarity transformation, each $\underline{q}_i$ reduces to the elementary unit vector $\underline{e}_i$ and the equations are completely decoupled.

When A is defective, i.e. there exist fewer than n linearly independent eigenvectors, additional terms will need to be introduced to give the general solution.

A simple example is

$$A = \begin{bmatrix} a & 1 & 0 \\ 0 & a & 1 \\ 0 & 0 & a \end{bmatrix}$$

which has the eigenvalue $\lambda = a$ with multiplicity 3 and all the eigenvectors are parallel to $\underline{e}_1$.

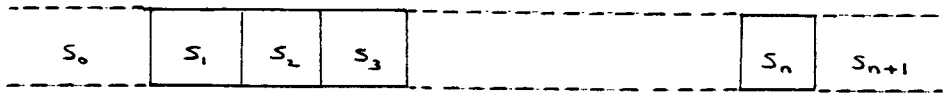
The solution of (i) is in this case

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} t^2/2\, x_3^{(o)} + tx_2^{(o)} + x_1^{(o)} \\ tx_3^{(o)} + x_2^{(o)} \\ x_3^{(o)} \end{bmatrix} e^{at}$$

In general, the solution of (i) may be written $\underline{x} = \exp(At)\underline{x}^{(o)}$.

Example

Let $u_1, u_2, \ldots, u_n$ be the concentrations at time t of a given molecule in the finite segments $S_1, S_2, \ldots, S_n$ of an infinite tube, the segments being separated by porous partitions.




The diffusion rate between adjacent segments is directly proportional to the difference in concentrations, so that with appropriate choice of units we have

$$\frac{du_1}{dt} = -u_1 + (u_2 - u_1)$$

$$\frac{du_i}{dt} = (u_{i-1} - u_i) + (u_{i+1} - u_i) \qquad [i=2,3,\ldots,n-1]$$

$$\frac{du_n}{dt} = (u_{n-1} - u_n) - u_n$$

the concentrations in the infinite sections $S_0, S_{n+1}$ being taken as zero.

Hence $\frac{d\underline{u}}{dt} = A\underline{u}$ where A is the nxn matrix

$$\begin{bmatrix} -2 & 1 & 0 & 0 & \ldots & 0 \\ 1 & -2 & 1 & 0 & \ldots & 0 \\ 0 & 1 & -2 & 1 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & 0 & & -2 \end{bmatrix}$$

The solution is given by $\underline{u} = \sum_{i=1}^{n} \alpha_i \underline{q}^{(i)} e^{\lambda_i t}$

where $\lambda_i = -2 + 2\cos\left(\frac{i\pi}{n+1}\right)$

and $q_j^{(i)} = \left(\frac{2}{n+1}\right)^{1/2} \sin\left(\frac{ij\pi}{n+1}\right) \qquad [i = 1,2,\ldots,n+1]$

as quoted by Gregory and Karney [19]. As $t \to \infty$, $u_i \to 0$ as each $\lambda_i$ is negative $[i = 1,2,\ldots,n+1]$. This is in accordance with the ultimate physical state of the system. The above theory is a finite approximation to the heat equation $\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$ governing conduction along a continuous rod.

b) Implicit First Order Systems

The system of differential equations $A\underline{x} = B\frac{d\underline{x}}{dt}$ may again be analysed by assuming a solution of the form $\alpha\underline{q}e^{\lambda t}$. This leads to an eigenvalue problem of the form $A\underline{x} = \lambda B\underline{x}$ with corresponding characteristic equation $\det(A - \lambda B) = 0$. The theoretical treatment of this equation presents greater difficulties than the standard case; in particular, if A and B are both singular we can have $\det(A - \lambda B) = 0$. Such a system is said to be incomplete and any value of $\lambda$ is a valid solution. Wilkinson [48] gives examples of this situation but considers that incomplete systems are likely to be the result of incorrect formulation of a practical problem. If $\det(B) \neq 0$ the system can, in theory, be reduced to the standard form $B^{-1}A\underline{x} = \lambda\underline{x}$ but if $\det(B) = 0$ and $\det(A) \neq 0$ there may exist less than n finite solutions for $\lambda$. Wilkinson [48] observes further that:

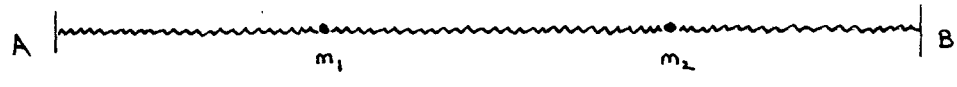
> "... when A and B are general Hermitian matrices there may be nothing distinctive about the problem $A\underline{x} = \lambda B\underline{x}$. No general purpose algorithm has been derived which gives an effective reduction while retaining the Hermitian property."

The situation is, however, more favourable when at least one of A and B is positive definite as a solution in exponentials may then be formulated.

Systems of the form $B\ddot{\underline{x}} = -A\underline{x}$ with A and B both positive definite arise commonly in connection with vibration problems (both mechanical and electrical). For a solution of the form $\underline{x} = \alpha \underline{u} e^{i\sqrt{\lambda}t}$ we require $\lambda B\underline{u} = A\underline{u}$. There are n positive eigenvalues $\lambda_j$ and n independent eigenvectors $\underline{q}_j$ each of which gives rise to the two solutions $\alpha_j \underline{q}_j e^{i\sqrt{\lambda_j}t}$ and $\beta_j \underline{q}_j e^{-i\sqrt{\lambda_j}t}$. When A and B are both real, it follows that the general solution may be expressed in the form

$$\sum_{j=1}^{n} u_j \left( \alpha_j \cos(\sqrt{\lambda_j}t) + \beta_j \sin(\sqrt{\lambda_j}t) \right)$$

Example




The diagram represents two particles of masses $m_1$ and $m_2$ connected to each other and to fixed points A and B by three springs each of stiffness s, other resistances being negligible. Let $x_1$, $x_2$ denote the displacements of $m_1$ and $m_2$ from their respective equilibrium positions.

The Lagrange equations then give $\omega^2 B\underline{x} = A\underline{x}$ where

$$A = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} \qquad \text{and} \qquad B = \begin{bmatrix} m_1/s & 0 \\ 0 & m_2/s \end{bmatrix}$$

Transforming to the standard problem $\omega^2 \underline{x} = B^{-1} A\underline{x}$ gives the characteristic (pulsatance) equation

$$\omega^4 - 2s\omega^2 \left( \frac{1}{m_1} + \frac{1}{m_2} \right) + \frac{3s^2}{m_1 m_2} = 0$$

The eigenvectors corresponding to the two roots $\omega_1^2, \omega_2^2$ represent the normal modes of the vibration. All other possible patterns of motion may be expressed as combinations of these two extremes.

c) General Systems of Linear Differential Equations with Constant Coefficients

Generalized problems of the form $(\lambda^r A_r + \ldots + \lambda A_1 + A_0)\underline{x} = \underline{0}$ may be derived from homogeneous systems of linear differential equations with constant coefficients of the form

$$A_r \frac{d^r \underline{x}}{dt^r} + \ldots + A_1 \frac{d\underline{x}}{dt} + A_0 \underline{x} = \underline{0}$$

by again assuming a solution of the form $\underline{x} = \alpha \underline{q} e^{\lambda t}$. Problems of this type commonly arise in the quadratic form

$$(\lambda^2 A_2 + \lambda A_1 + A_0)\underline{x} = \underline{0}$$

Again, a comprehensive theory is not available but special cases have been considered, notably the overdamped physical system which is referred to by Lancaster [28] and Ruhe [40]. Here $[a_1(\underline{x})]^2 > 4a_0(\underline{x})a_2(\underline{x})$ for all vectors $\underline{x}$ where $a_i(\underline{x})$ is the inner product $\langle A_i \underline{x}, \underline{x} \rangle$ and $A_0$, $A_1$, $A_2$ are all self-adjoint. For such systems the eigenvalues are all real and there exist n linearly independent eigenvectors. Quadratic problems of this type may be converted to the linear symmetric form.

Existence of solutions in the general quadratic case is not, however, guaranteed. For example, the following equation, quoted by Lancaster [28]:

$$\lambda^2 I - A_0 = \underline{0} \quad \text{where} \quad A_0 = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

Examples

1. Damped oscillations can arise from the introduction of a dissipation function dependent on $\dot{\underline{x}}$ into the Lagrange equations. Such a function typically takes the form of a mechanical or electrical resistance. The resulting differential equation can hence be written

$$A_2 \frac{d^2 \underline{x}}{dt^2} + A_1 \frac{d\underline{x}}{dt} + A_0 \underline{x} = \underline{0}$$

2. Terray and Lancaster [42] discuss a quadratic eigenvalue problem arising from a study of heat transfer to fluids flowing between parallel plates. In this case they prove that:

> "The spectrum of $A(\lambda)$ consists of at most a countable set of eigenvalues with infinity as the only possible limit point."

Limited observations are also obtained concerning the eigenvectors. Such observations are likely to be rather too general to assist in the numerical calculation of specific roots for which reference must usually be made to the practical problem concerned.

## Numerical Solution of Differential Equation Problems involving a Variable Parameter

Ordinary and partial differential equations representing physical situations often have coefficients dependent upon a parameter, $\lambda$ say, which arises from the variation of the constraints with time or position. Feasible solutions will exist for particular values of $\lambda$ only. The usual method adopted is to use a finite difference aproximation to the derivatives over a mesh of suitable size. This results in an eigenvalue formulation from which the appropriate values of $\lambda$ may be found. Examples are:

1. The ordinary differential equation

$$\frac{d}{dx}\, k \left(\frac{dy}{dx}\right) + (\lambda g - \ell)y = 0$$

where k,g and $\ell$ are continuous functions of x and boundary conditions are given. This governs, for instance, the temperature distribution in a heterogeneous bar [23].

2. The equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \lambda u$$

(quoted by Peters and Wilkinson) [36] which could arise, for example, in potential problems.

3. The flutter problem in aerodynamics which leads to a quadratic eigenproblem having velocity as the variable parameter $\lambda$. Solutions with the real part of $\lambda$ positive are of interest as they represent "flutter velocity" [47].

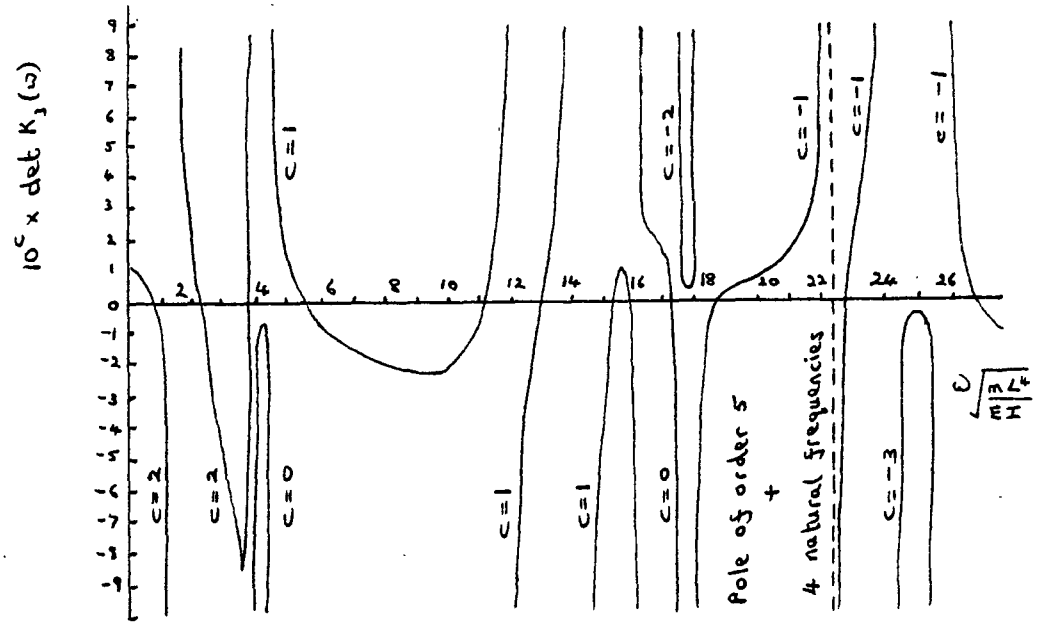
Non-Polynomial Eigenvalue Problems

Attempts to describe and compare algorithms for the general problem have been made by Ruhe [40] and Lancaster [28], although Lancaster states that "... because much of the activity is very recent or still under development there is no comprehensive survey of results." Ruhe formulates a definition of overdamping in terms of a generalized Rayleigh quotient which may be extended to cover the general eigenproblem. Such a system is known to possess n real eigenvalues and again the linear symmetric theory is of assistance. There remain, however, practical examples which do not come into this category and for which little is known about the existence of solutions. The following are examples which have arisen in practical applications:

1. Wittrick and Williams [49] discuss the problem of determining the natural undamped frequencies of vibration of a linearly elastic structure for which they derive an equation of the form $K(\omega)D = 0$ where D is the matrix of displacements and $\omega$ is the frequency to be determined. If D is not of full rank, solutions are sought which satisfy either $\det K(\omega) = 0$ or $D = 0$. The elements of the dynamic stiffness matrix K are, in general, non-linear functions of $\omega$. In the case of a finite number of degrees of freedom these functions are quotients of two polynomials; for an infinite number of degrees of freedom they are generally transcendental.

The potential difficulties of such problems are well-illustrated by Fig 5.1 which is reproduced from Wittrick and Williams paper. The large number of poles and the wide range of function values would make considerable demands on the most sophisticated equation-solving routines.

Fig 5.1




Variation of the determinant of the 3×3 dynamic stiffness matrix $K_3(\omega)$ of Simpson and Tabarrok's frame

2. A problem encountered recently at the National Physical Laboratory [15] concerned wave propagation along dielectric tubes. Analysis based on Maxwell's equations produced an eigenvalue equation of the form $A(\beta)\underline{x} = \underline{0}$ where $\beta$ is the phase coefficient to be determined and the eigenvector $\underline{x}$ contains the components of the electric and magnetic field vectors. A is an eight by eight non-symmetric matrix whose elements are Bessel functions involving the parameter $\beta$. The requirement was for as many roots as possible within a given interval together with the associated eigenvectors. The method of formulation

ensured that all the required roots were real but their distribution pattern was not known a priori.

3. The characteristic equation for the earth-ionosphere wave-guide (as encountered, for instance, in the propagation of radio waves) motivated the development of a procedure by Bahar and Fitzwater [5] for tracing the loci of complex roots as the electromagnetic parameters are varied along the propagation path. As in the previous example, the coefficients involve Bessel functions and the root-finding exercise is again complicated by the presence of poles and the wide variation in magnitude of the function values. It is considered that initial estimates of the number of roots and their locations may be unavailable for such problems.

## 5.2 METHODS OF NUMERICAL SOLUTION

### Formulation

The solution of the eigenproblem $\mathbf{A}(\lambda)\underline{x} = \underline{0}$, where $\mathbf{A}$ is any square matrix whose elements are functions of a scalar parameter $\lambda$, by means of the scalar equation $\det[\mathbf{A}(\lambda)] = 0$ may be applied generally regardless of the properties possibly possessed by $\mathbf{A}$. Such methods hence have the strength of versatility, but may fail to take into account any special features of the problem. Evaluation of the determinant may be carried out using either Gaussian

elimination with row and/or column interchanges or by a sequence of orthogonal transformations such as in the method of Householder. In the opinion of Wilkinson [47] "The weakness of such methods lies in the volume of work required and not in their stability".

The Problem $Ax = \lambda x$

For the standard problem $A\underline{x} = \lambda\underline{x}$ the deciding factor in the choice of method is likely to be the number of roots required. The QR algorithm is a widely accepted method for obtaining the full set of eigenvalues. Wilkinson observes, however, that "in practice it is uncommon for all the eigenvalues of a large matrix to be required." For example, the behaviour of solutions containing terms of the form $e^{\lambda_i t}$ where $\lambda_i$ is real will be largely governed by the largest (dominant) eigenvalue. In such cases many of the difficulties associated with the search for a large number of roots of an equation will be irrelevant.

Determinant evaluation routines, such as those contained in the NAG library, have been designed to take advantage of particular forms of matrix such as tridiagonal and positive definite which are of common occurrence. A potential disadvantage of the equation-solving approach is the wide range of values assumed by determinants; provision of a scaling factor will usually be necessary for success.

The Problem $Ax = \lambda Bx$

If A and B are symmetric and B is positive definite, the Cholesky factorization $B = LL^T$ may be used. Then $A\underline{x} = \lambda LL^T\underline{x} \Rightarrow L^{-1}A\underline{x} = \lambda L^T\underline{x} \Rightarrow L^{-1}AL^{-T}(L^T\underline{x}) = \lambda(L^T\underline{x})$ which is of standard form.

When A and B are of band form $L^{-1}$ may still be a full matrix, so it is likely to be uneconomic to calculate L explicitly. Instead Peters and Wilkinson [36] suggest working with the successive minors $\det(A_r - \lambda B_r)$ which form a Sturm sequence. Alternatively Crawford [12] gives an adaptation of the Cholesky factor method which produces a reduction to standard form whilst retaining the symmetric band form. Reduction of such systems to standard form is only justified if the system is small and all the eigenvalues are required; even in such cases there is a risk of ill-conditioning with respect to inversion unless one of A and B is positive definite. Eigenvectors when required may then be found using the inverse iteration scheme $(A - \lambda B)\underline{x}_r = k_r B\underline{x}_{r-1}$ where k is a normalizing factor.

Other methods discussed by Peters and Wilkinson [36] and [37] are iterative techniques which are useful for obtaining complete sets of eigenvalues and vectors but which are generally more demanding on storage than the equation-solving approach and are hence less well suited to the isolation of specific eigenvalues.

With general matrices A and B, it is possible to express the problem in standard form as either $B^{-1}A\underline{x} = \lambda\underline{x}$ or $(1/\lambda)\underline{x} = A^{-1}B\underline{x}$, provided that at least one of A and B is non-singular. It is, however, usually preferable to apply the QZ algorithm directly to reduce A and B to upper triangular form. This is effected by applying an equivalence transformation of the form $XAY(Y^{-1}\underline{x}) = \lambda XBY(Y^{-1}\underline{x})$ [37] and may also be employed when both A and B are singular. Peters and Wilkinson use Gaussian elimination with complete pivoting for the factorization but suggest that other methods such as the singular value decomposition, might also prove appropriate in this context.

The problem $(\lambda^r A_r + \ldots + \lambda A_1 + A_0)\underline{x} = \underline{0}$

A reduction to the form $A\underline{x} = \lambda B\underline{x}$ may be made, which will in this case be of dimension rn with

$$A = \begin{bmatrix} 0 & I & 0 & \ldots & 0 \\ 0 & 0 & I & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & \ldots & I \\ -A_0 & -A_1 & -A_2 & \ldots & -A_{r-1} \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} I & 0 & 0 & \ldots & 0 & 0 \\ 0 & I & 0 & \ldots & 0 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & \ldots & I & 0 \\ 0 & 0 & 0 & \ldots & 0 & A_r \end{bmatrix}$$

The alternative method is to work directly with the equation $\det(\lambda^r A_r + \ldots + \lambda A_1 + A_0) = 0$. This obviously has the advantage of reducing storage requirements but does not enable use to be made of the considerable experience which has been gained in the development of

algorithms for the standard problem. Wilkinson [47] also mentions that caution may need to be exercised in the evaluation of the elements of the determinant as, being explicit polynomials, these may prove to be ill-conditioned. He considers, however, that this is unlikely to be serious in practical problems for which r is usually small.

The General Problem $A(\lambda)\underline{x} = \underline{0}$

When the elements of A are non-polynomial functions of $\lambda$ transformation to standard form will not usually be possible and the choice will normally be between a matrix iterative method and determinant evaluation. Ruhe [40] describes three methods of the former type, namely:

1. An algorithm based on inverse iteration and a generalization of the Rayleigh quotient.

2. Generalizations of the QR algorithm based on work by Kublanovskaya [26].

3. Formulation as a sequence of linear problems obtained from the Taylor series

$$A(\lambda + h) = A(\lambda) + hA'(\lambda) + h^2/2\ R(\lambda,h)$$

Since all these methods involve computation of the derivative $A'(\lambda)$, questions are inevitably raised as to the accuracy and efficiency of such a procedure. Wilkinson [47] states that:

> "... the evaluation of a derivative involves far more work than that of a function value. The contrast is even more marked for the generalized eigenvalue problem for which the relevant function is of the form $\det(A_r\lambda^r + A_{r-1}\lambda^{r-1} + \ldots + A_0)$"

We might expect this problem to be further exacerbated in the general case when formulae for the derivatives are known. For non-polynomial problems, however, an analytical expression for the derivative may not be available. Ruhe [40] suggests that in some cases the difference approximation

$$A'(\lambda_s) = \frac{A(\lambda_s) - A(\lambda_{s-1})}{\lambda_s - \lambda_{s-1}}$$

may be used and comments favourably on the results whilst conceding that the limiting accuracy is poorer.

## 5.3 SOLUTION BY EQUATION-SOLVING TECHNIQUES

### Choice of Algorithm

The reliability of "interval" methods is a strong argument in favour of their use whenever possible. This will apply in particular to problems which can be solved by the Sturm sequence method with bisection as this method is extremely stable. It also has the virtue of flexibility as it enables the user to direct the search towards specific roots or to gain an overall picture of the root distribution. Once appropriate intervals have been isolated a switch can be made to an interpolation method in order to accelerate convergence, although the gain may not be worthwhile if the eigenvalues are

clustered. For the band symmetric problem $A\underline{x} = \lambda B\underline{x}$, Peters and Wilkinson [36] use a combined linear interpolation and bisection method for which they report results of "almost the optimum accuracy for the precision of the computation".

When the problem is non-symmetric or when bisection fails due to roots of even multiplicity, a "search" strategy must be employed. The possibility of complex roots must also be considered. Laguerre's method has gained wide acceptance for the standard problem because of its global convergence properties and rapid rate of convergence; it is not however appropriate for non-polynomial problems for which first and second derivatives are not readily available. The algorithm most frequently used for the generalized problem is Muller's method which has produced results of high accuracy [37]. Wilkinson [47] finds inverse interpolation methods less satisfactory but there seems to have been less experience with rational interpolation in connection with eigenvalue problems. Peters and Wilkinson [37] prefer an inverse iteration method to equation-solving, however, for eigenvalue tracking problems as in such cases the eigenvalues will be required in chronological order together with the corresponding eigenvectors.

The order of root determination is particularly important if the characteristic equation is expressed as an explicit polynomial in order to minimize ill-conditioning following deflation. As the order of computation cannot easily be predetermined, the explicit form is generally avoided and suppression of roots is employed rather than explicit deflation. The role of suppression is simply to prevent repeated convergence to the same root; it should not, therefore, adversely affect the attainable accuracy of subsequent roots.

## Initial Estimates

In the case of the standard problem $A\underline{x} = \lambda\underline{x}$ we have that $|\lambda_i| \leqslant \|A\|$ $(i=1,2,\ldots,n)$ for any matrix norm. The usual choice is $\|A\|_1$ or $\|A\|_\infty$ both of which are quick to compute. This provides a starting interval for the bisection process or a bound for the search routine if this can be set in the calling program. Gerschgorin's theorems also provide a method of fixing bounds but these are not so straightforward, particularly when n is large. These theorems can be of use if they reveal isolated discs since each of these will contain one eigenvalue only. Such an outcome is not likely to occur frequently but would give valuable information in the complex case when the bisection method is not applicable. When m eigenvalues have been found $(m \geqslant 2)$, Wilkinson [47] suggests the "stepping-off" points $\frac{1}{3}(\lambda_{m-1} + 2\lambda_m)$ and $\frac{1}{3}(2\lambda_{m-1} + \lambda_m)$ for the next search when using an

interpolation method, although an increase in the shift might be necessary if the first iterate is not near a root. He goes on to say, however, that "... iterative methods appear to best advantage when approximations to the eigenvalues are available from an independent source." At present practical considerations are probably the only way of obtaining estimates for generalized problems which Ruhe [40] considers to require better estimates than the linear case.

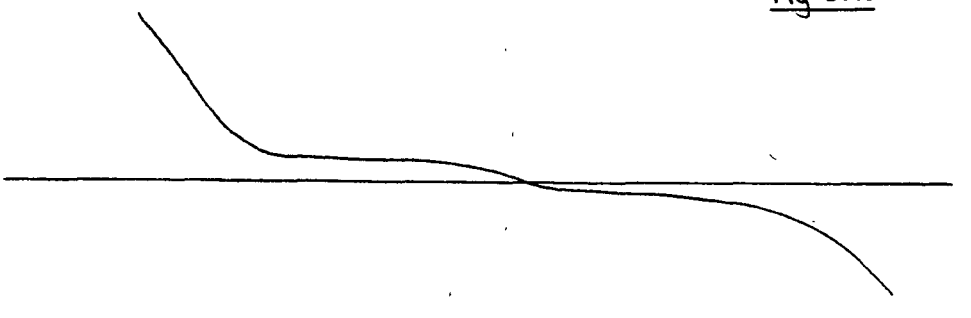
## Checking of Results

The results of the investigation described in the previous chapter indicate that it is highly desirable to have some means of checking the validity of roots obtained by iteration, particularly where the function value varies widely (as will often be the case with eigenvalue problems). The test usually recommended for the standard problem is to use the relation $\sum_{i=1}^{n} \lambda_i = \mathrm{tr}(A)$. This is not foolproof as the selected algorithm might lead to the introduction of errors which cancel on addition. Wilkinson [47] considers the test to be effective, however, when a suppression technique is employed as the zeros are then located independently of each other.

In real examples any complex roots will occur in conjugate pairs but it is a sensible precaution not to accept a conjugate automatically. If the second root is also obtained iteratively little extra work will be involved but the agreement between the conjugate roots will be a guide to their accuracy. It may also be the case that a computed eigenvalue with a small imaginary part actually represents a real root.

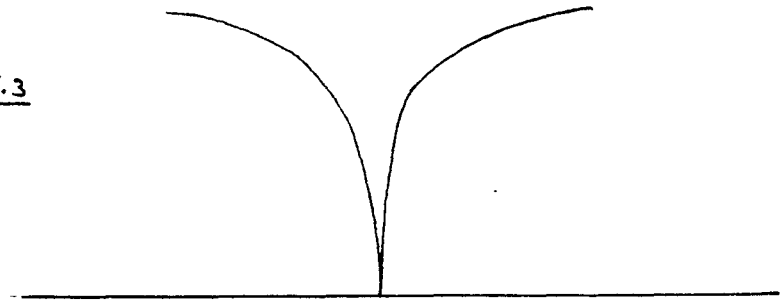
A possible approach to checking results for the generalized problem $A(\lambda)\underline{x} = \underline{0}$ is illustrated by example 2 of section 5.1. The problem was solved at the National Physical Laboratory using the routine RTFS1R (described in section 4.1) with an option of either Muller or rational interpolation and a relative or automatic stopping criterion. Appropriate scaling was employed in the calculation of the elements of A and in the evaluation of the determinant from the upper triangular form [15]. The typical behaviour of the function near a root is as illustrated in Fig 5.2 and indicates that function value convergence tests would be inappropriate.

Fig 5.2



Another method of solution attempted was the minimization of the singular values of $A$ which proved to be highly unsuitable for numerical estimation being of the form shown in Fig 5.3 (It may be noted in this connection that Ruhe [40] finds the minimization of $\|A(\lambda)\|$ to be a somewhat unsatisfactory method.)

Fig 5.3



The singular value decomposition

$$Q \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & O \\ & & \ddots & \\ & O & & \sigma_n \end{bmatrix} P^T$$

with $Q^TQ = P^TP = I$, was, however, very useful for calculating the eigenvectors and for checking the eigenvalues by examining the magnitude of the ratio $\sigma_n/\sigma_1$. This was found to be very much smaller for valid roots than for the false roots which appeared occasionally as a result of automatic stopping. Identification of valid roots by this method was assisted by row and column scaling of the matrix $A$ before decomposition. The calculation of the eigenvectors is described in Appendix G.

## 5.4 NUMERICAL EXAMPLES

The National Physical Laboratory's equation-solving routine [20] was used in conjunction with NAG routines F03AFF and F03AHF [33] for real and complex determinant evaluation respectively, to solve a variety of eigenvalue problems. These routines enabled all function values to be scaled in the form $a \times 2^c$ .

For the standard problem, the main program used, a list of the matrices tested and the results using Muller's method are given in Appendix F. The data chosen was from a selection by Gregory and Karney [19]. Isolated eigenvalues were found to machine precision, whilst multiple roots were of poorer limiting accuracy and, predictably, required a much larger number of iterations. The difficulties encountered in problems 3 and 11 were successfully overcome by respectively increasing the bound and suppressing the zero root. The rational interpolation method applied to the same examples gave similar results excepting number 7 for which the smallest root was found twice. In the majority of examples the Muller method required slightly fewer iterations.

Four generalized eigenvalue problems were examined, the details being given in Appendix F. The first example is of the form $A\underline{x} = \lambda B\underline{x}$, with A and B band symmetric. All 20 eigenvalues were calculated successfully but the total number of iterations was similar to that obtained by Peters and Wilkinson with a linear interpolation and

bisection method, despite the higher order iterative method employed.

The remaining three examples are all taken from the paper by Ruhe [40], the first involving an exponential term in $\lambda$ and the others being of quadratic form with complex roots. Example 1 has 16 distinct eigenvalues which were obtained successfully by both the rational interpolation and Muller methods, the former being slightly more efficient in this case.

The second example was solved by Muller's method with only a slightly greater number of iterations than the methods tested by Ruhe. Taking into account the fact that these require derivative values, Muller may be considered superior from the efficiency point of view. The latter also has the virtue of requiring only one initial estimate, subsequent roots being located automatically.

The final example involves a variable parameter $\alpha$ and is ill-conditioned for small values of $\alpha$. A standard error test was sufficient to solve the problem in the case $\alpha = 0.5$, but an excessive number of iterations was required when $\alpha = 0.0$. In this case there are triple roots at $\lambda = \pm i$ and a double root at the origin. The automatic stopping criterion was applied effectively in this case, the results reflecting the poor limiting accuracy attainable.

# CHAPTER 6

## SEARCH STRATEGIES IN THE COMPLEX PLANE

Until recently little attention has been paid to the computation of complex roots in comparison with the real case. Henrici [21] justifies the need for further investigation in the polynomial case, stating that:

> "Even if the given polynomial has real coefficients, it can be proved that in a certain statistical sense, if the degree is high enough, most of its zeros will be complex. In most applications of polynomials (e.g. in the theory of control systems or in differential equations) real and complex zeros are equally relevant."

Such comments may well extend to the general function. Henrici's work is a development of complex analysis which gives particular attention to numerical aspects of the subject. A considerable amount of theory is available to assist in the solution of polynomial equations; in particular the establishment of bounds and methods for determining the numbers of zeros in specific regions [22]. It is thus desirable in such cases to employ a program specifically designed for polynomials. To date few algorithms have been published for the solution of

non-polynomial equations and those available show a diversity of methods. Brief descriptions of some approaches to the problem are given below, together with comments on the practical performance of algorithms where available. Few comparative studies would appear to have been carried out, however.

## 6.1 SEARCH METHODS

### Graphical Methods

A preliminary sketch of a polynomial function may give clues to the location of complex roots. For the general function Larkin [29] describes how automatic plotting of the real and imaginary parts can give useful information on the location of roots and poles and their orders. Such a diagram may be expensive to produce if the region of interest is uncertain or when the function is complicated. In the latter case, not only will each function evaluation be lengthy, but a fine mesh may be required to give a clear picture of the behaviour of the function. For a transcendental function $f(z)$, the graphs are likely to be complicated and Larkin suggests that plots of $\ln|f(z)|$ = constant and $\arg(f(z))$ = constant are likely to be more helpful. This representation also provides an analogy with electric field theory. The field lines thus generated also indicate convergence regions for the iterative process $z_1 = z_0 - \lambda f(z_0)/f'(z_0)$ with $\lambda$ small, although a warning is given that starting within

such a region does not guarantee that a subsequent iterate will not, in practice, jump outside the region.

Comparison with a Polynomial

Ahlfors [2] suggests that practical use may be made of Rouche's theorem in the following manner:
Suppose that the function $f(z)$ possesses a Taylor expansion of the form $f(z) = P_{n-1}(z) + z^n f_n(z)$ where $P_{n-1}(z)$ is a polynomial of degree $(n-1)$ and $R^n |f_n(z)| < |P_{n-1}(z)|$ on the circle $|z| = R$, then $f(z)$ has the same number of zeros in the region $|z| \leq R$ as $P_{n-1}(z)$. Routines to determine the number of zeros of a polynomial in a given circle are available, for example Lehmer [30]. The solutions of the polynomial equation $P_{n-1}(z) = 0$ might also provide suitable first estimates for the zeros of $f(z)$.

If the conditions for the application of Rouche's theorem are not fulfilled, the use of a truncated series approximation is considerably more risky. This is illustrated by Delves and Lyness [14] with the example $f(z) = e^z$ whose nth. degree Taylor expansion must have n zeros in the complex plane whilst the original function has none. They also warn that obtaining the coefficients of the approximating polynomial is likely to involve heavy computation and the resulting equation may be ill-conditioned irrespective of the condition of $f(z)$.

## Numerical Use of the Principle of the Argument

Henrici [21] considers the principle of the argument to be "a powerful instrument for finding first approximations, however crude, to the zeros of analytic functions". For such a function the number of roots within a simple closed curve C is given by

$$\frac{1}{2\pi i}\int_C \frac{f'(z)}{f(z)}\,dz \quad = \quad \frac{1}{2\pi}\,\Delta_c \arg f(z) \quad = \quad n(f(z),0)$$

where $n(f(z),0)$ is the winding number of $f(z)$ about the origin, provided that no root of $f(z)$ lies on C. In the case of a meromorphic function, we have the more general formula $N - P = n(f(z),0)$ where N is the number of roots and P is the number of poles within C.

The curve usually chosen for the numerical calculation of the integral is either a circle or a rectangle. Henrici points out that although the winding number is an integer, it is quite possible to select the wrong value unless careful attention is given to the effect of rounding errors. The algorithm given by Henrici depends upon a subdivision of the contour C which is such that each subarc subtends an angle not greater than $\pi/2$ at the origin; numerical quadrature is avoided. Both this method and the direct evaluation of the integral can present the following problems:

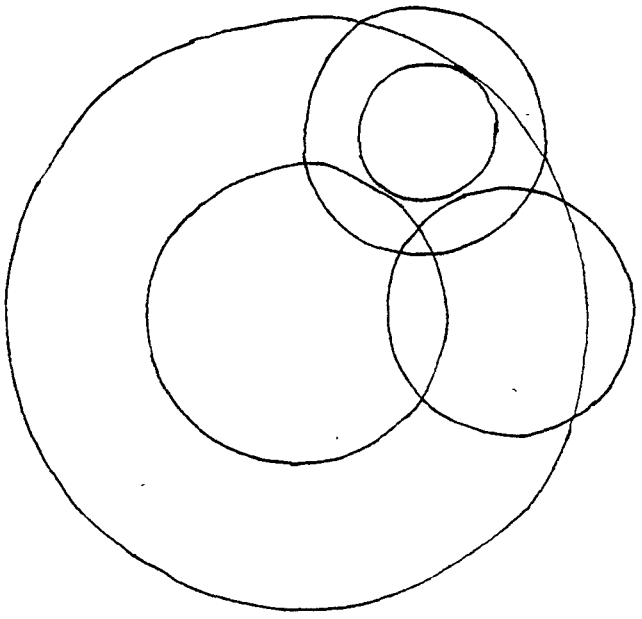
1. It is difficult to ensure that no zero lies on the boundary C.

2. A very small subdivision of the curve is required if a root is near to the boundary.

For these reasons it may be more practicable to use an algorithm such as that of Wehl [45] for polynomials which systematically eliminates regions which contain no root.

A frequently discussed implementation of the principle of the argument is Lehmer's method for polynomials [30]. The first step is to find, by starting with the unit circle and successively doubling or halving the radius, an annulus $R < |z| < 2R$, containing at least one root of the equation. This annulus is covered by eight smaller circles, at least one of which must contain a root. When such a circle has been found, an annulus is again obtained and the process may be repeated as illustrated in Fig 6.1.

Fig 6.1



This method of generating successive regions may be used for non-polynomial equations; alternatively the efficiency can be improved by varying the radii of the covering circles [16]. Lehmer's method for determining whether a region contains a root is, however, restricted to polynomials. The principal criticism of Lehmer as the sole method has been that of inefficiency [1], [14], without compensating improvement in limiting accuracy. Susceptibility to machine underflow or overflow has also been mentioned. A more practical proposition is to use Lehmer's method for the initial search, switching to an alternative method, such as Newton, as soon as it will give convergence [22].

The method of Delves and Lyness [14] adopts a similar technique but evaluates the actual numbers of roots in each region and can be used to solve $f(z) = 0$ where $f(z)$ is any analytic function of z. The algorithm consists of the following steps:

1. The number of roots in the initial region is evaluated by contour integration; the region is subdivided and the evaluation is repeated until the number of roots, N, in each subregion is acceptably small (N = 5 is suggested).

2. Numerical estimation of

$$\frac{1}{2\pi i} \int_C z^n \frac{f'(z)}{f(z)} \, dz$$

gives, by Cauchy's theorem, the sum of the nth. powers of the roots. These sums determine a polynomial equation whose zeros are the same as those of the given function.

3. The polynomial equation is solved using a subroutine specifically designed for such functions.

4. If necessary the solutions may be refined by the use of an iterative method with the original function.

Details are given of alternative methods of subdivision based on rectangles or circles and suitable methods of numberical quadrature are described in each case. The algorithm is claimed to be more efficient than that of Lehmer for solving high order polynomials and also reduces

the build-up of rounding errors associated with explicit deflation.

A major problem associated with the direct use of the principle of the argument is the requirement for derivative values. Delves and Lyness [14] have developed two methods for use in cases where an analytic expression for $f'(z)$ is unavailable. These algorithms are based on:

1. evaluation of $\ln(f(z))$.
2. obtaining coefficients of the truncated Taylor series for $f(z)$ using numerical integration.

Spira [41] presents an alternative method of calculating $\Delta_c \arg f(z)$ based on function evaluations at discrete points but the selection of covering discs is based on a knowledge of the least upper bound of $|f'(z)|$ in the region.

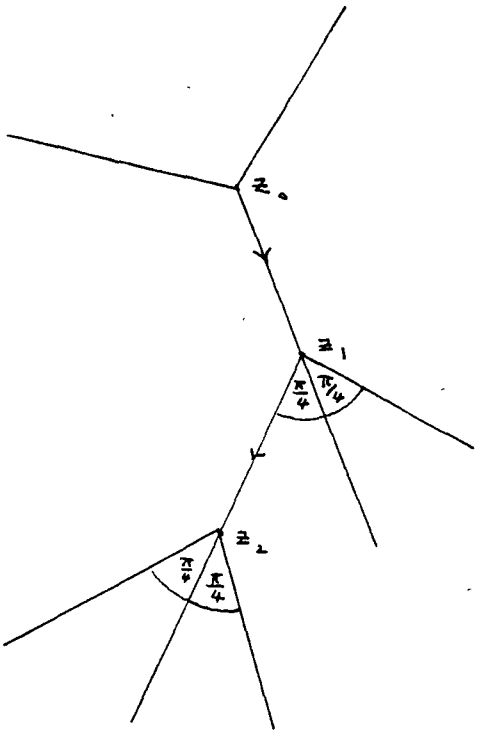
Descent Methods

These methods are based on the idea of minimization of the magnitude of the function value. One such example is the method of steepest descent which has the disadvantage of requiring derivative values; in addition, the rate of convergence can be slow [21].

A direct search method for equation-solving was proposed by Ward [44]. This seeks to minimize the function $|Re(f(z))| + |Im(f(z))|$ by examining a set of five points $(x_0, y_0)$, $(x_0 \pm \lambda, y_0)$ and $(x_0, y_0 \pm \lambda)$, choosing that which gives a minimum function value and taking this as the centre point for the next iteration. When the chosen point is the centre of the five, the step length $\lambda$ is reduced before resuming the search. More generally, any number of points placed equidistantly on the circumference of a circle may be used. The most serious shortcoming of the algorithm is the possibility of locating a local minimum which is not a root of the equation.

Bach [4] presents a more sophisticated "walk pattern" commencing with the vertices of an equilateral triangle and subsequent searches in forward branching directions as shown in Fig 6.2. If the process converges, the algorithm is repeated with a smaller step length; otherwise the original configuration may be rotated. In addition to improving the success rate, Bach claims that his algorithm will also give improved efficiency as compared with Ward's selection of points.

Fig 6.2




When several roots of an equation are required, descent methods will generally require separate estimates for each. Hence this approach is likely to be inappropriate for root-tracking problems.

## Interpolation methods

The iterative methods described in Chapter 3 may also be used for the computation of complex roots provided that their formulation is not dependent upon the retention of an interval containing a root. In practice a method is usually chosen which allows iterates to move automatically into the complex plane. Thus Laguerre's method has found

favour for polynomial equations whilst that of Muller is the usual choice in the general case. The former method has been found to have good global convergence properties. Less is known in the case of Muller, but the results of Chapter 5 would seem to indicate that, given suitable bounds, it may be possible to dispense with other preliminary search strategies. If a complex initial estimate is provided, rational interpolation may be used although there has been less experience with this method.

Bahar and Fitzwater [5] have developed a method of solution for an equation expressed in the form $F(\nu) = 1$, which is designed to enable the loci of the roots to be traced as the parameters of F are varied. An outline of the procedure is as follows:

The first two estimates are obtained by fixing $Re(\nu)$ and varying $Im(\nu)$ by bisection until $|F(\nu)| = 1$. Subsequent iterations use the starting point $(2\nu_{p+1} - \nu_p)$ and a search direction $i(\nu_{p+1} - \nu_p)$ for the bisection i.e. approximately perpendicular to the locus $|F(\nu)| = 1$. This process is repeated until $Im(F(\nu))$ changes sign. The estimated root is then refined by linear interpolation parallel to $|F(\nu)| = 1$ and bisection perpendicular to this direction.

The authors state that, for the functions considered, "$F'(\nu)$ cannot be written in a closed analytical form and numerical differentiation is subject to significant errors in critical regions." The method described does not require estimates of the derivative and can give satisfactory results even when $F(\nu)$ has isolated poles in the region of interest.

## 6.2 NUMERICAL EXAMPLES

The descent and interpolation methods were tested on a small selection of functions using FORTRAN routines by Bach [3] and the National Physical Laboratory [20] respectively. Table 6.1 shows the functions chosen, together with the roots sought. Separate estimates of each root were supplied in the case of the downhill algorithm; for the interpolation program an attempt was made to obtain all the required roots from a single user-supplied starting value. Muller's method was chosen in preference to rational interpolation as it gave more satisfactory results in most cases.

The results obtained are given in Tables 6.2 and 6.3. Both routines performed successfully with polynomial functions (numbers 1 to 3) although the superior efficiency of routine RTFSIC in terms of the numbers of function evaluations is apparent. The inclusion of a poor starting value for function number 1 demonstrates that success can be achieved even when the initial estimate is

some distance from a root. Better starting values are required for transcendental functions in both cases; for roots successfully computed, the observations regarding efficiency are similar to the polynomial examples.

The tables for this section are given on pages 131-135.

TABLE 6.1 FUNCTIONS TESTED (COMPLEX ROOTS)

| FUNCTION | ROOTS SOUGHT |
|---|---|
| 1. $z^3 - 1$ | 1, $-0.5 \pm 0.8660i$ |
| 2. $z^3 - (3+4i)z^2 + (-1+11i)z + (6-6i)$ | 2, 3i, 1+i |
| 3. $(z+1)^2 (z+i)^2$ | -1,-1,-i,-i |
| 4. $z - e^{-z}$ | Real root 0.5671 + 5 complex roots |
| 5. $4i(z-i) - \exp(-2\ \mathrm{Re}(z))$ | 0.2699+0.7301i<br>-0.7440+1.7440i<br>-1.3089+2.3089i |
| 6. $z^2 + \ln \lvert 2 + \mathrm{Re}(z) \rvert$ | -1.980, -2.017<br>$\pm$0.8326i |
| 7. $z - \cos(z)$ | Real root 0.7391 + 6 complex roots |
| 8. The acoustic waveguide function as given by Rodman [39]. | A complex root near 0.1 + 0.1i |

TABLE 6.2 RESULTS OBTAINED USING ROUTINE RTFS1C

TOLERANCES

(Relative error in root < $\varepsilon$ or magnitude of final function value < $\eta$)

Functions 7 and 8 $\varepsilon = 10^{-6}$ $\eta = 10^{-8}$

Remaining Functions $\varepsilon = 10^{-4}$ $\eta = 10^{-6}$

| FUNCTION | ROOT | STARTING POINT | TOTAL NO. FUNCTION EVALUATIONS |
|---|---|---|---|
| 1 | ALL | (0,0) | 16 |
| 1 | ALL | (100,100) | 32 |
| 2 | ALL | (0,0) | 19 |
| 3 | ALL | (0,0) | 31 |
| 4 | (0.5671,0.0)<br>(-2.4016,-10.776)<br>(-2.4016,10.776)<br>(-1.5339,4.3752)<br>(-1.5339,-4.3752) | (0.0,0.0) | 49 |

(cont.)

TABLE 6.2 (cont.)

| FUNCTION | ROOTS | STARTING POINT | TOTAL NO. FUNCTION EVALUATIONS |
|---|---|---|---|
| 5 | (0.2699,0.7301) | (0,0) | 140 |
| | (-0.7440,1.7440) | | |
| | fails to find | | |
| | third root. | | |
| 6 | (0.0,0.8326) | (0,0) | 13 |
| | (0.0,-0.8326) | | |
| | (-2.0171,0.0) | (-2.01,0) | 8 |
| | (-1.9802,0.0) | (-1.98,0) | 6 |
| 7 | (0.7391,0.0) | (0,0) | 53 |
| | (-2.4869,1.8094) | | |
| | (-2.4869,-1.8094) | | |
| | (-9.1100,2.9502) | | |
| | (-9.1100,-2.9502) | | |
| | (-15.488,3.4566) | | |
| | (-15.488,-3.4566) | | |
| 8 | (0.07200,0.005304) | (0.1,0.1) | 12 |

TABLE 6.3 RESULTS OBTAINED USING ROUTINE CRF (BACH)

INITIAL STEP LENGTHS

| | |
|---|---|
| Function 1 (starting point (100,100)) | 10.0 |
| Function 5 (real roots) | 0.01 |
| Function 7 | 0.01 |
| Function 8 | 0.1 |
| Remaining Functions | 1.0 |

TOLERANCES

(Final step length < z or magnitude of final function value < $\eta$)

| | | |
|---|---|---|
| Function 1 (starting point (100,100)) | $z = 10^{-1}$ | $\eta = 10^{-2}$ |
| Functions 7 and 8 | $z = 10^{-6}$ | $\eta = 10^{-8}$ |
| Remaining Functions | $z = 10^{-4}$ | $\eta = 10^{-6}$ |

| FUNCTION | ROOT | STARTING POINT | NO. FUNCTION EVALUATIONS |
|---|---|---|---|
| 1 | (1.0,0.0) | (0.0,0.0) | 12 |
| | (-0.5,0.8660) | (-1.0,-1.0) | 69 |
| | (-0.5,-0.8660) | (-1.0,1.0) | 69 |
| | (-0.5,-0.8660) | (100,100) | 87 |
| 2 | (1.0,1.0) | (0.0,0.0) | 96 |
| | (0.0,3.0) | (0.0,1.0) | 75 |
| | (2.0,0.0) | (4.0,0.0) | 6 |
| 3 | (-1.0,0.0) | (0.0,0.0) | 3 |
| | (-1.0,0.0) | (2.0,0.0) | 63 |
| | (0.0,-1.0) | (1.0,1.0) | 63 |

(cont.)

TABLE 6.3 (cont.)

| FUNCTION | ROOT | STARTING POINT | NO. FUNCTION EVALUATIONS |
|---|---|---|---|
| 4 | (0.5671,0.0) | (0.0,0.0) | 237 |
| | (-2.4016,10.776) | (0.0,10.0) | 153 |
| | (-2.4016,-10.776) | (-2.0,-10.0) | 81 |
| | (-1.5339,4.3752) | (0.0,5.0) | 114 |
| | (-1.5339,-4.3752) | (-1.5,-4.0) | 72 |
| 5 | (0.2699,0.7301) | (0.0,0.0) | 138 |
| | (-0.7440,1.7440) | (-1.0,1.0) | 300 |
| | (-1.3089,2.3089) | (-1.5,2.5) | 324 |
| 6 | (0.0,0.8326) | (0.0,1.0) | 102 |
| | (0.0,-0.8326) | (0.0,-1.0) | 102 |
| | (-1.980,0.0) | (-1.98,0.0) | 258 |
| | (-2.017,0.0) | (-2.01,0.0) | 36 |
| 7 | (0.7391,0.0) | (0.0,0.0) | 216 |
| | (-2.4869,1.8094) | (-2.0,2.0) | 63 |
| | (-2.4869,-1.8094) | (-2.5,-1.8) | 144 |
| | (-9.1100,-2.9501) | (-10.0,0.0) | 90 |
| | (-9.1100,2.9501) | (-9.1,3.0) | 147 |
| | (-15.488,-3.4566) | (-15.0,0.0) | 117 |
| | (-15.488,3.4566) | (-15.5,3.5) | 99 |
| 8 | (0.07200,0.5304) | (0.1,0.1) | 198 |

## 6.3 CONCLUDING REMARKS

It is not always possible to make a clear distinction between global and local methods for root evaluation since the region of convergence of an iterative procedure is heavily dependent upon the nature of the function under consideration. Delves and Lyness [14] consider that "A feature of almost any global method for locating zeros is that it is uncommon to find the zeros to high accuracy". According to this criterion, the method of Lehmer and direct search descent methods are best suited to finding crude estimates of the roots only, since each of these methods requires a large number of iterations in comparison with, say, interpolation methods. The results obtained above indicate, however, that it may be possible to dispense with a preliminary global search and to use an interpolation algorithm with root suppression to conduct the search and to carry out the iterative improvements.

In practice it is likely that further information will be available to the user from consideration of the source of the equation and the nature of the required roots. Many theoretical results for the eigenvalue problem, in particular the calculation of bounds using a matrix norm and the singular value decomposition are valid in the complex case. Wilkinson [47] has observed that:

> "favourable distributions are not uncommon for matrices which arise in connexion with damped mechanical and electrical oscillations, for which the eigenvalues are, in general, complex conjugate pairs."

If the problem can be reformulated as a real root-finding exercise, for example the waveguide problem for dielectric tubes referred to in Chapter 5, considerable economies can be made and the chances of success improved.

On the question of attainable accuracy Wilkinson points out that "in our experience polynomials with complex zeros which have arisen in practice have had quite well-conditioned zeros." [46] There appears to be insufficient evidence to extend this comment to the non-polynomial case. The two-dimensional nature of the problem, in addition to complicating the initial search, will make accuracy checks on the computed results difficult. It may, however, be the case that high accuracy is not required, as in a problem of automatic control mentioned by Henrici [21] in which only the number of zeros in a given region is required.

The development of programs for complex root-finding tends to reflect the extent to which complex roots have been sought in practical situations.

---

# CHAPTER 7

# IMPLICATIONS FOR SOFTWARE DEVELOPMENT

Interpolation methods are well-established for the computation of real roots and it is likely that they will continue to form the basis of algorithms for some time. For this reason, the inclusion of more sophisticated features of implementation, such as those described in Chapter 2, is desirable, both to extend the scope and to improve the reliability of the program.

The algorithms described, when convergent, can generally be relied upon to produce results of the maximum attainable accuracy consistent with machine precision and the condition of the problem, provided that termination criteria are selected with care. Most published numerical tests have been carried out using very good initial estimates of the roots. Although this is probably necessary for some functions, there is potential for the use of such methods as search procedures over a wider region, particularly in the polynomial case. When complex roots are required, iterative methods using an

interpolating function can also be strongly recommended from the point of view of efficiency.

Determinant evaluation combined with equation-solving has given very satisfactory results for generalized eigenvalue problems and is preferable to matrix iterative methods for the selection of specific roots and for tracking problems. The use of an adjustable bound has proved a particularly useful feature in this context.

Muller's method has been widely used when derivative values are not readily available; there is now, however, increasing interest in the use of rational functions for interpolation. In a recent paper [7] Barzilai and Ben-Tal show that the asymptotic rate of convergence of an algorithm to a simple root is independent of the nature of the interpolating function and depends only upon the number of interpolating points used and the orders of derivatives matched at these points. Thus the choice between rational interpolation and Muller's method will be determined principally by the nature of the function whose zeros are required. Very satisfactory results are obtained for polynomials, both real and complex by the Muller method, but Barzilai and Ben-Tal favour rational interpolation when the function values change rapidly. The examples of Chapters 4 and 6 tend to confirm this view. It was also observed that although the rational interpolation method can be economical in terms of the number of function evaluations required, it is more prone

to failure when a large number of roots is sought and a suppression technique is employed. Robinson [38] warns that success with quadratic interpolation for minimization problems cannot be guaranteed because of the possibility of repeated function values. Practical experience with the routine by Gonnet [18] shows that this can also be a cause of failure when such an interpolating function is used for equation-solving.

Whilst published routines have tended to favour three point interpolation, the rate of convergence in practice is not perhaps as good as might be expected in comparison with lower order methods. Barzilai and Ben-Tal suggest that maximum efficiency is obtained using two point interpolation. They also point out, however, that attempts to improve reliability by retaining an interval for the root are likely to incur the penalty of a slower convergence rate; a fact which is well-known for the secant and regula falsi methods.

The major outstanding difficulties concern the automatic search for several roots. In particular, the following problems were noted:

1. The difficulty of distinguishing between genuine multiple roots and the failure of a routine to suppress a previous root.

2. Locating a second root which is some distance from the first without causing a complete breakdown of the procedure by, for example, the occurrence of overflow.

3. The acceptance of points which are not roots of the given equation. This is particularly associated with the use of the automatic stopping criterion.

4. Failure to detect some of the roots when several are required.

The above points suggest that the greatest current need is for further study of the global convergence properties of iterative methods. Another area for further investigation is the detection of multiple roots and the development of an effective technique for the acceleration of convergence in such cases. It is also worthwhile to incorporate facilities for handling function values in a scaled form. The development of routines for the computation of standard functions in this form will be a necessary adjunct to the equation-solving program. Reverse communication is another valuable feature, allowing flexibility to the user, particularly when solving non-polynomial equations which still require some experimentation with input parameters in many cases.

---

## APPENDIX A

## ORDER OF CONVERGENCE OF THE SECANT METHOD

The iterative process may be written

$$x_{i+1} = \frac{x_{i-1} f_i - x_i f_{i-1}}{f_i - f_{i-1}}$$

$$\Rightarrow \alpha + \varepsilon_{i+1} = \frac{(\alpha + \varepsilon_{i-1}) f(\alpha + \varepsilon_i) - (\alpha + \varepsilon_i) f(\alpha + \varepsilon_{i-1})}{f(\alpha + \varepsilon_i) - f(\alpha + \varepsilon_{i-1})}$$

where $\alpha$ is the exact root and $\varepsilon_i$ is the error in $x_i$.

Using Taylor's series to second order terms $\alpha + \varepsilon_{i+1} \doteq$

$$\frac{\left[(\alpha + \varepsilon_{i-1})\left(f(\alpha) + \varepsilon_i f'(\alpha) + \frac{\varepsilon_i^2}{2} f''(\alpha)\right) - (\alpha + \varepsilon_i)\left(f(\alpha) + \varepsilon_{i-1} f'(\alpha) + \frac{\varepsilon_{i-1}^2}{2} f''(\alpha)\right)\right]}{\left[f(\alpha) + \varepsilon_i f'(\alpha) + \frac{\varepsilon_i^2}{2} f''(\alpha)\right] - \left[f(\alpha) + \varepsilon_{i-1} f'(\alpha) + \frac{\varepsilon_{i-1}^2}{2} f''(\alpha)\right]}$$

Putting $f(\alpha) = 0$ gives $\alpha + \varepsilon_{i+1} \doteq$

$$\frac{(\varepsilon_i - \varepsilon_{i-1}) \alpha f'(\alpha) + (\varepsilon_i - \varepsilon_{i-1}) \varepsilon_i \varepsilon_{i-1} \frac{f''(\alpha)}{2} + (\varepsilon_i^2 - \varepsilon_{i-1}^2) \alpha \frac{f''(\alpha)}{2}}{(\varepsilon_i - \varepsilon_{i-1}) f'(\alpha) + (\varepsilon_i^2 - \varepsilon_{i-1}^2) \frac{f''(\alpha)}{2}}$$

$$\Rightarrow \alpha + \varepsilon_{i+1} \doteq \frac{\alpha f'(\alpha) + \varepsilon_i \varepsilon_{i-1} \frac{f''(\alpha)}{2} + (\varepsilon_i + \varepsilon_{i-1}) \alpha \frac{f''(\alpha)}{2}}{f'(\alpha) + (\varepsilon_i + \varepsilon_{i-1}) \frac{f''(\alpha)}{2}}$$

whence $\varepsilon_{i+1} \doteq \dfrac{\varepsilon_i \varepsilon_{i-1} \frac{f''(\alpha)}{2}}{f'(\alpha) + (\varepsilon_i + \varepsilon_{i-1}) \frac{f''(\alpha)}{2}} \doteq \dfrac{\varepsilon_{i-1} \varepsilon_i f''(\alpha)}{2f'(\alpha)}$

$\Rightarrow \varepsilon_{i+1} \doteq k \varepsilon_{i-1} \varepsilon_i$ where k is constant.

Assume a solution of the form $\left| \dfrac{\varepsilon_i}{\varepsilon_{i-1}^p} \right| = c \quad (i = 1, 2, \ldots)$

where c is constant, then $|\varepsilon_i| = c |\varepsilon_{i-1}|^p$

$\Rightarrow |\varepsilon_{i+1}| = |k| |\varepsilon_{i-1}| |\varepsilon_i| = k \left( \dfrac{|\varepsilon_i|}{c} \right)^{1/p} |\varepsilon_i|$

$\Rightarrow c |\varepsilon_i|^p = k c^{-1/p} |\varepsilon_i|^{1+1/p}$

If this is to be valid for all positive integers i, we require

$$p = 1 + \frac{1}{p}$$

For convergence we must have $|\varepsilon_i| < |\varepsilon_{i-1}|$

so we select the root greater than unity to give

$$p = (1 + \sqrt{5})/2$$

---

## APPENDIX B

## CALLING PROGRAM FOR A REVERSE COMMUNICATION ROUTINE

```
C                PROGRAM GONNET.FOR
C
C  MAIN PROGRAM CALLING FUNCTION ROOT1 OF GONNET AND
C  INCORPORATING FUNCTION DEFLATION.
C  MAXIMUM NUMBER OF ROOTS TO BE FOUND IS TWENTY.
C  "STEPPING-OFF" POINT FOR SECOND AND SUBSEQUENT ROOTS
C  IS PREVIOUS ROOT FOUND.
C
      EXTERNAL F
C
      DIMENSION W(9),ROOTS(20)
      DATA IN,NOUT/5,5/
C
      WRITE(NOUT,1)
    1 FORMAT(//1X,34H RESULTS FOR GONNET WITH DEFLATION/
     *          1X,20HFUNCTION EXP(-X) - X/)
      WRITE(NOUT,2)
    2 FORMAT(1X,28H NUMBER OF ROOTS REQUIRED = )
      READ(IN,3)NROOTS
```

```
    3 FORMAT(I2)
      WRITE(NOUT,4)
    4 FORMAT(1X,7H EPS = )
      READ(IN,6)EPS
      WRITE(NOUT,5)
    5 FORMAT(1X,7H ETA = )
      READ(IN,6)ETA
    6 FORMAT(E)
      WRITE(NOUT,7)
    7 FORMAT(1X,20H INITIAL ESTIMATE = )
      READ(IN,8)X
    8 FORMAT(F)
      FX=F(X)
C
C  NEXT ROOT
      DO 17 I=1,NROOTS
      WRITE(NOUT,9)
    9 FORMAT(/12X,1HX,20X,2HFX)
      ITS=0
      IF(I.EQ.1)ITS=1
      W(1)=0
      XERR=1.0E32
C
C NEXT ITERATION
   10 IERR=0
      WRITE(NOUT,11)X,FX
   11 FORMAT(2(5X,1P,E15.8))
      IF(I.EQ.1)GO TO 14
```

```
C
C  FUNCTION DEFLATION PROCESS
      DO 13 J=1,I-1
      D=X-ROOTS(J)
      IF(D.NE.0.0)GO TO 12
C  PERTURB IF COMPUTATIONALLY EQUAL TO A PREVIOUS ROOT
      X=1.01*X
      FX=F(X)
      ITS=ITS+1
      GO TO 10
   12 FX=FX/D
   13 CONTINUE
C
C  CALL TO ROOT-FINDER
   14 X=ROOT1(X,FX,XERR,ESTD,IERR,W)
      GO TO (18,20,22)IERR
      FX=F(X)
      ITS=ITS+1
C  CONVERGENCE TEST
      IF(XERR.GT.EPS*ABS(X).AND.ABS(FX).GT.ETA)GO TO 10
C
C  ROOT FOUND
   15 WRITE(NOUT,16)X,FX,ITS
   16 FORMAT(/1X,8H ROOT = ,1P,E15.8/
     *        1X,18H FUNCTION VALUE = ,1P,E15.8/
     *        1X,34H NUMBER OF FUNCTION EVALUATIONS = I2/)
      ROOTS(I)=X
C
```

```
   17 CONTINUE
      STOP
C
C  FAILURE TERMINATION
   18 WRITE(NOUT,19)
   19 FORMAT(/1X,24H MORE THAN 80 ITERATIONS/)
      STOP
   20 WRITE(NOUT,21)
   21 FORMAT(/1X,25H REPEATED ARGUMENT VALUES/)
      STOP
   22 WRITE(NOUT,23)
   23 FORMAT(/1X,27H UNABLE TO APPLY ANY METHOD/)
      STOP
      END
C
C  FUNCTION EVALUATION ROUTINE
      FUNCTION F(X)
      F=EXP(-X)-X
      RETURN
      END
```

---

## APPENDIX C

## INPUT PARAMETERS FOR ROUTINE RTFS1C

A list of the parameters to be supplied by the user is given, together with a summary of the purpose of each.

METHOD Offers choice of three-point rational interpolation or quadratic interpolation.

X Vector containing up to three approximations to the next root sought.

A Function values are supplied in the form $ab^c$. Vector A contains the values of a corresponding to the estimates in X.

IBASE The base, b, used for function evaluation.

IC Vector containing the values of c in the function values corresponding to the estimates in X.

NMAX The maximum number of roots required.

N The number of roots found so far.

ROOTS Vector containing the known roots and, if required, approximations to later roots.

LROOTS The length of the vector ROOTS.

STEP When one estimate only is provided, this parameter may be used to construct the two further estimates required for three point interpolation.

TOL Vector giving tolerances for each of the four convergence tests: automatic stopping, relative error in the root, magnitudes of function value and suppressed function value.

BOUND Maximum magnitude for each iterate.

NEXTX Determines how an approximation to the next root is to be found following the acceptance of a root. The choice is between a user-supplied estimate, the last iterate, the root just found or its conjugate.

MAXNFV The maximum total number of function evaluations.

NFV Indicates the number of approximations to the next root which are being supplied.

INFORM Is set to zero before the first call and subsequently takes the value returned by the routine. This parameter indicates the progress of iterations.

IFAIL Failure parameter giving options of hard failure (with message) or soft failure (with or without message).

---

# APPENDIX D

## OUTPUT DATA FOR REAL ROOT TESTS

### RESULTS FOR BUS AND DEKKER METHOD USING NAG ROUTINE C05AZF

### FUNCTION $x^3$

NUMBER OF ROOTS REQUIRED = 1

MAXIMUM NUMBER OF ITERATIONS PER ROOT = 99

ABSOLUTE TOLERANCE = 0.5E-04

INITIAL INTERVAL = (-0.5, 1.0)

| x | f(x) |
|---|---|
| -5.00000000E-01 | -1.25000000E-01 |
| 1.00000000E+00 | 1.00000000E+00 |
| 3.33333340E-01 | -3.70370380E-02 |
| -2.63157900E-01 | -1.82242310E-02 |
| -1.67810830E-01 | -4.72563290E-03 |
| 4.16094580E-01 | 7.20404120E-02 |
| -1.31866260E-01 | -2.29298430E-03 |
| -9.79853550E-02 | -9.40770110E-04 |

| x | f(x) |
|---|---|
| -6.24008580E-02 | -2.42980650E-04 |
| 1.76846860E-01 | 5.53085250E-03 |
| -5.23325790E-02 | -1.43323170E-04 |
| -3.78528050E-02 | -5.42368180E-05 |
| -2.40977780E-02 | -1.39936500E-05 |
| 7.63745420E-02 | 4.45498100E-04 |
| -2.10379310E-02 | -9.31127350E-06 |
| -1.49531850E-02 | -3.34349820E-06 |
| -9.51793390E-03 | -8.62239790E-07 |
| 3.34283040E-02 | 3.73545100E-05 |
| -8.54898830E-03 | -6.24804520E-07 |
| -5.99923370E-03 | -2.15917250E-07 |
| -3.81826460E-03 | -5.56670290E-08 |
| 1.48050200E-02 | 3.24509180E-06 |
| -3.50418440E-03 | -4.30289600E-08 |
| -2.43483250E-03 | -1.44346840E-08 |
| -1.54958700E-03 | -3.72089900E-09 |
| 6.62771650E-03 | 2.91133220E-07 |
| -1.44639390E-03 | -3.02593580E-09 |
| -9.97081200E-04 | -9.91269110E-10 |
| -6.34545300E-04 | -2.55498230E-10 |
| 2.99658560E-03 | 2.69079160E-08 |
| -6.00390990E-04 | -2.16422550E-10 |
| -4.11225720E-04 | -6.95409800E-11 |

| x | f(x) |
|---|---|
| -2.61699490E-04 | -1.79229150E-11 |
| 1.36744310E-03 | 2.55698250E-09 |
| -2.36699490E-04 | -1.32614800E-11 |
| -1.65576120E-04 | -4.53934440E-12 |
| -1.05380390E-04 | -1.17025210E-12 |
| 6.31031330E-04 | 2.51277010E-10 |
| -8.03803920E-05 | -5.19338320E-13 |
| -5.53803920E-05 | -1.69850990E-13 |
| -3.03803920E-05 | -2.80401370E-14 |
| 3.00325460E-04 | 2.70879710E-11 |
| -5.38039240E-06 | -1.55754950E-16 |
| 1.96196080E-05 | 7.55215600E-15 |

ROOT = -5.38039240E-06

FUNCTION VALUE = -1.55754950E-16

NUMBER OF ITERATIONS = 44

---

## RESULTS USING ROUTINE RTFS1C

## FUNCTION $x^3$

TOLERANCE FOR FUNCTION VALUE = 1.0E-12

FIRST ESTIMATE = 1.0

METHOD 1 - RATIONAL INTERPOLATION

| x | f(x) |
|---|---|
| 1.20000000E+00 | 1.72800000E+00 |
| 1.10000000E+00 | 1.33100000E+00 |
| 5.46202690E-01 | 1.62952680E+00 |
| 4.05109520E-01 | 6.64840340E-02 |
| 2.78531050E-01 | 2.16083130E-02 |
| 1.86557710E-01 | 6.49291340E-03 |
| 1.28224540E-01 | 2.10820800E-03 |
| 8.74674260E-02 | 6.69173950E-04 |
| 5.96148870E-02 | 2.11867420E-04 |
| 4.06999000E-02 | 6.74186450E-05 |
| 2.77686220E-02 | 2.14122830E-05 |
| 1.89461350E-02 | 6.80082990E-06 |
| 1.29280080E-02 | 2.16070170E-06 |
| 8.82106230E-03 | 6.86376910E-07 |
| 8.73910860E-03 | 6.67423360E-07 |
| 4.88067840E-03 | 1.16262740E-07 |
| 3.39173420E-03 | 3.90180380E-08 |
| 2.36583750E-03 | 1.32420350E-08 |

| x | f(x) |
|---|---|
| 2.34385730E-03 | 1.28763710E-08 |
| 1.30260760E-03 | 2.21024710E-09 |
| 9.07216760E-04 | 7.46677720E-10 |
| 6.32605250E-04 | 2.53161920E-10 |
| 6.26727920E-04 | 2.46171130E-10 |
| 3.48332020E-04 | 4.22649350E-11 |
| 2.42592040E-04 | 1.42767600E-11 |
| 1.69161060E-04 | 4.84062250E-12 |
| 1.67589440E-04 | 4.70695410E-12 |
| 9.31451930E-05 | 8.08130210E-13 |

SOLUTION = 9.31451930E-05

NUMBER OF ITERATIONS = 29

---

METHOD 2 - QUADRATIC INTERPOLATION

| x | | f(x) | |
|---|---|---|---|
| 1.2000000E+0 | 0.0000000E+0 | 1.7280000E+00 | 0.0000000E+00 |
| 1.1000000E+0 | 0.0000000E+0 | 1.3310000E+00 | 0.0000000E+00 |
| 5.4848492E-1 | 3.1490379E-1 | 1.8333270E-03 | 2.5297567E-01 |
| 3.7284821E-1 | 3.3624260E-1 | -7.4629842E-02 | 1.0221381E-01 |
| 2.0861734E-1 | 3.1309670E-1 | -5.2272713E-02 | 1.0186300E-02 |
| 7.6628333E-2 | 2.6262930E-1 | -1.5406168E-02 | -1.3488232E-02 |
| 3.9331354E-3 | 2.0397476E-1 | -4.9086143E-04 | -8.4770473E-03 |
| -3.7881312E-2 | 1.4637415E-1 | 2.3805064E-03 | -2.5059859E-03 |
| -5.6644735E-2 | 9.6487597E-2 | 1.4003112E-03 | 3.0492085E-05 |
| -5.9718549E-2 | 5.7548959E-2 | 3.8036791E-04 | 4.2511608E-04 |
| -5.4138352E-2 | 2.9060510E-2 | -2.1515734E-05 | 2.3098369E-04 |
| -4.4445008E-2 | 9.8436694E-3 | -7.4874962E-05 | 5.7380509E-05 |
| -3.3674451E-2 | -1.8952223E-3 | -3.7822909E-05 | -6.4405597E-06 |
| -2.3635800E-2 | -8.1107657E-3 | -8.5395563E-06 | -1.3059700E-05 |
| -1.5228538E-2 | -1.0529142E-2 | 1.5332115E-06 | -6.1580985E-06 |
| -8.7476367E-3 | -1.0565733E-2 | 2.2602404E-06 | -1.2460036E-06 |
| -4.1179948E-3 | -9.2911888E-3 | 9.9663995E-07 | 3.2939637E-07 |
| -4.0797358E-3 | -9.2048673E-3 | 9.6911882E-07 | 3.2030044E-07 |
| 1.3856479E-4 | -6.3656681E-3 | -1.6841987E-08 | 2.5758122E-07 |
| 1.3144002E-3 | -4.5882456E-3 | -8.0741430E-08 | 7.2811122E-08 |
| 1.8066090E-3 | -3.1324375E-3 | -4.7283762E-08 | 6.4705442E-11 |
| 1.8233937E-3 | -3.1033351E-3 | -4.6619255E-08 | -1.0663201E-09 |
| 1.8667228E-3 | -1.2694307E-3 | -2.5195322E-09 | -1.1224951E-08 |
| 1.5972343E-3 | -5.7186099E-4 | 2.5077898E-09 | -4.1897096E-09 |

| x | | f(x) | |
|---|---|---|---|
| 1.2796307E-3 | -1.2663151E-4 | 2.0337785E-09 | -6.2002945E-10 |
| 1.2677420E-3 | -1.2545502E-4 | 1.9776180E-09 | -6.0290804E-10 |
| 7.5315850E-4 | 2.7366420E-4 | 2.5801061E-10 | 4.4521090E-10 |
| 4.9634485E-4 | 3.4070899E-4 | -5.0572389E-11 | 2.1225899E-10 |
| 3.0233297E-4 | 3.4143468E-4 | -7.8100975E-11 | 5.3823091E-11 |
| 2.9952409E-4 | 3.3826252E-4 | -7.5944304E-11 | 5.2336826E-11 |
| 7.7845092E-5 | 2.7338248E-4 | -1.6982224E-11 | -1.5462077E-11 |
| 5.2177811E-6 | 2.1346972E-4 | -7.1316992E-13 | -9.7102350E-12 |
| -3.5310603E-5 | 1.5785640E-4 | 2.5956500E-12 | -3.3431028E-12 |
| -3.5638663E-5 | 1.5638980E-4 | 2.5696614E-12 | -3.2290472E-12 |
| -6.2536358E-5 | 7.9118136E-5 | 9.2980573E-13 | 4.3299056E-13 |
| -6.0376278E-5 | 4.5775595E-5 | 1.5944891E-13 | 4.0467822E-13 |

SOLUTION = (-6.03762780E-05, 4.57755950E-05)

NUMBER OF ITERATIONS = 37

---

RESULT USING ROUTINE RTFS1C

FUNCTION $(X - 1)EXP[-1/(X - 1)^2]$

METHOD 1 - RATIONAL INTERPOLATION

RELATIVE TOLERANCE FOR ROOT = 1.0E-4

FIRST ESTIMATE = 1.5

SOLUTION = 1.1441417

NUMBER OF ITERATIONS = 34

RESULTS USING ROUTINE RTFS1C

FUNCTION EXP(-X) - X

RELATIVE TOLERANCE FOR ROOT = 1.0E-4

FIRST ESTIMATE = 0.0

NUMBER OF ROOTS REQUESTED = 2

BOUND = 100.0

SOLUTION = 5.67143290E-01

NUMBER OF ITERATIONS = 5

SOLUTION = -1.77339410E+01

NUMBER OF ITERATIONS = 22

BOUND = 25.0

SOLUTION = 5.67143290E-01

NUMBER OF ITERATIONS = 5

TERMINATES BY EXCEEDING BOUND

RESULTS FOR ROUTINE RTFS1C

BRENT'S FUNCTION

RELATIVE TOLERANCE FOR ROOT = 1.0E-6

FIRST ESTIMATE = 3.0

NUMBER OF ROOTS REQUESTED = 19

| SOLUTION | NUMBER OF ITERATIONS |
|---|---|
| 3.02291530E+00 | 6 |
| 4.19061160E+01 | 15 |
| 5.59535960E+01 | 14 |
| 9.00088680E+01 | 8 |
| 1.10026530E+02 | 7 |
| 1.32040550E+02 | 7 |
| 1.56052110E+02 | 8 |
| 1.82062060E+02 | 9 |
| 2.10071100E+02 | 10 |
| 2.40080050E+02 | 8 |
| 2.72090270E+02 | 10 |
| 3.06105120E+02 | 9 |
| 2.89013770E+02 | 4 |
| 3.42136940E+02 | 8 |
| Floating underflow occurs | |
| 5.45443620E+02 | exceeds 100 |

HARD FAILURE OCCURS WITH IRRECOVERABLE OVERFLOW

RESULTS USING ROUTINE RTFS1C WITH RATIONAL INTERPOLATION

FUNCTION: PRODUCT R=1 TO R=20 OF (X - R)

AUTOMATIC STOPPING WITH TOLERANCE = 0.05

FIRST ESTIMATE = 0.0

NUMBER OF ROOTS REQUESTED = 20

| SOLUTION | NUMBER OF ITERATIONS |
|---|---|
| 1.00000000E+00 | 11 |
| 2.00000000E+00 | 11 |
| 3.00000000E+00 | 10 |
| 4.00000000E+00 | 10 |
| 5.00000000E+00 | 9 |
| 6.00000010E+00 | 5 |
| 7.00000000E+00 | 9 |
| 8.00000000E+00 | 12 |
| 1.20000000E+01 | 9 |
| 1.41833500E+01 | 6 |
| 1.70082570E+01 | 6 |
| 1.90000000E+01 | 9 |
| 1.95331750E+01 | 9 |
| 2.00000000E+01 | 9 |
| 1.80000000E+01 | 14 |
| 1.78666890E+01 | 5 |
| 1.60000000E+01 | 15 |
| 1.59242180E+01 | 5 |
| 1.71162410E+01 | 11 |
| 1.40000000E+01 | 17 |

RESULTS USING ROUTINE RTFS1C WITH RATIONAL INTERPOLATION

BRENT'S FUNCTION

AUTOMATIC STOPPING WITH TOLERANCE = 0.05 (A)

OR RELATIVE TOLERANCE FOR ROOT = 1.0E-06 (B)

FIRST ESTIMATE = 3.0

NUMBER OF ROOTS REQUESTED = 19

| SOLUTION | NUMBER OF ITERATIONS | STOPPING CRITERION |
|---|---|---|
| 3.02291530E+00 | 6 | B |
| 3.25241210E+00 | 7 | A |
| 5.67085050E+00 | 6 | A |
| 6.68375360E+00 | 6 | B |
| 1.28949690E+01 | 10 | A |
| 1.87007450E+01 | 7 | A |
| 1.96760000E+01 | 8 | B |
| 2.80314690E+01 | 25 | A |
| 2.98282270E+01 | 9 | A |
| 3.92464580E+01 | 6 | A |
| 4.19061160E+01 | 10 | A |
| 5.84661070E+01 | 6 | A |
| 7.19856650E+01 | 9 | B |
| 8.85848240E+01 | 6 | A |
| 1.08563600E+02 | 6 | A |
| 1.32040550E+02 | 8 | A |
| 1.56052110E+02 | 7 | B |
| Floating underflow occurs | | |
| 1.86999610E+02 | 6 | A |
| 2.07160740E+02 | 43 | A |

RESULTS FOR THE FUNCTION (4x - 7)/(x - 2)

RELATIVE TOLERANCE FOR ROOT = 1.0E-4

TOLERANCE FOR FUNCTION VALUE = 1.0E-6

ROUTINE BY GONNET

INITIAL ESTIMATE = 1.6

FAILS - UNABLE TO APPLY ANY METHOD

INITIAL ESTIMATE = 1.7

ROOT = 1.75000000E+00

NUMBER OF FUNCTION EVALUATIONS = 10

ROUTINE BY BARRODALE AND WILSON

INITIAL ESTIMATE = 1.0

FAILS - MAXIMUM NUMBER OF ITERATIONS EXCEEDED

INITIAL ESTIMATE = 1.5

FLOATING DIVIDE CHECK INDICATED

FALSE ROOT CLAIMED AT x = 1.5

INITIAL ESTIMATE = 1.6

ROOT = 1.75000000E+00

NUMBER OF FUNCTION EVALUATIONS = 9

ROUTINE RTFS1C

FIRST ESTIMATE = 1.0

TERMINATES BY EXCEEDING BOUND

FIRST ESTIMATE = 1.5

ROOT = 1.75000000E+00

NUMBER OF FUNCTION EVALUATIONS = 6

---

RESULTS USING ROUTINE RTFS1C WITH RATIONAL INTERPOLATION

FUNCTION $x^3 - 2x - 5$

RELATIVE TOLERANCE FOR ROOT = 1.0E-4

TOLERANCE FOR FUNCTION VALUE = 1.0E-6

FIRST ESTIMATE = 0.0

| x | f(x) |
|---|---|
| 2.00000000E-01 | -5.39200000E+00 |
| 1.00000000E-01 | -5.19900000E+00 |
| -1.78690590E+00 | -7.13183720E+00 |
| -3.68973140E-01 | -4.31228610E+00 |
| -7.18846600E-01 | -3.93376390E+00 |
| -1.25653490E+00 | -4.47084800E+00 |
| -7.87057390E-01 | -3.91343530E+00 |
| -1.13718940E+00 | -4.19623430E+00 |
| -1.57084700E+00 | -5.73446580E+00 |
| -1.96332090E+00 | -8.64123150E+00 |
| -6.12624560E+01 | -2.29805900E+05 |
| -7.81385420E-01 | -3.91431430E+00 |
| 2.33308670E-01 | -5.45391770E+00 |
| -3.21638720E+00 | -3.18412240E+01 |
| -1.71305430E+00 | -6.60094370E+00 |
| -4.10551350E+00 | -6.59883910E+00 |
| -1.00318460E+00 | -4.00321510E+00 |
| 1.16612770E+00 | -5.74649210E+00 |
| -2.43999470E+00 | -1.46467000E+01 |

| x | f(x) |
|---|---|
| -1.65887770E+00 | -6.24726940E+00 |
| -2.59816010E+00 | -1.73423920E+01 |
| -4.08890130E-01 | -4.25058260E+00 |
| -5.29274090E+00 | -1.42680630E+02 |
| 1.06417480E+00 | -5.92320570E+00 |
| -2.17093550E+00 | -1.08896640E+01 |
| -5.01862390E+01 | -1.26306630E+05 |
| 5.54213240E+00 | 1.54143610E+02 |
| -1.72784480E+00 | -6.70270050E+00 |
| -1.46265040E+00 | -5.20381470E+00 |
| -6.45013390E-01 | -3.97832610E+00 |
| -2.60233960E+00 | -1.74188110E+01 |
| -5.69349440E+00 | -1.78172640E+02 |
| 3.82437830E-01 | -5.70894080E+00 |
| 4.86842140E+00 | 1.00652180E+02 |
| 6.07223150E-01 | -5.99055100E+00 |
| 3.85838420E+01 | 5.73580930E+04 |
| 8.74495030E-01 | -6.08022740E+00 |
| -3.37979140E+01 | -3.85447280E+04 |
| 8.79259590E-01 | -6.07876590E+00 |
| 4.77454920E+01 | 1.08741660E+05 |
| 8.83516310E-01 | -6.07735880E+00 |
| 1.38158090E+01 | 2.60448290E+03 |
| 9.24066540E-01 | -6.05907360E+00 |
| 7.49925010E+00 | 4.01749960E+02 |
| 1.09051690E+00 | -5.88416160E+00 |

| x | f(x) |
|---|---|
| 4.06190380E+00 | 5.38937960E+01 |
| 1.53028370E+00 | -4.47699790E+00 |
| 2.39886460E+00 | 4.00665990E+00 |
| 2.04488530E+00 | -5.38968090E-01 |
| 2.09675470E+00 | 2.46212480E-02 |
| 2.09454450E+00 | -7.76052480E-05 |

SOLUTION = 2.09455150E+00

NUMBER OF ITERATIONS = 52

---

RESULTS FOR THE FUNCTION $x - 4\sqrt{(x - 1)}$

REALATIVE TOLERANCE FOR ROOT = 1.0E-4

TOLERANCE FOR FUNCTION VALUE = 1.0E-6

NUMBER OF ROOTS REQUESTED = 2

ROUTINE GONNET

INITIAL ESTIMATE = 1.0

ROOT = 1.07179680E+00

FUNCTION VALUE = -2.98023220E-08

NUMBER OF FUNCTION EVALUATIONS = 9

ATTEMPT TO TAKE SQUARE ROOT OF NEGATIVE ARGUMENT.

FALSE ROOT GIVEN AT x = 9.44271910E-01

INITIAL ESTIMATE = 15.0

ROOT = 1.49282030E+01

FUNCTION VALUE = 0.00000000E+00

NUMBER OF FUNCTION EVALUATIONS = 4

ATTEMPT TO TAKE SQUARE ROOT OF NEGATIVE ARGUMENT.

FAILS - UNABLE TO APPLY ANY METHOD.

ROUTINE BY BARRODALE AND WILSON WITH REALRT = .TRUE.

INITIAL ESTIMATE = 1.0

ROOT = 1.07179680E+00

FUNCTION VALUE = -2.23517420E-07

NUMBER OF FUNCTION EVALUATIONS = 9

FLOATING UNDERFLOW OCCURS

TERMINATES BY EXCEEDING MAXIMUM NUMBER OF ITERATIONS.

FIRST ESTIMATE = 15.0

ROOT = 1.49282030E+01

FUNCTION VALUE = 0.00000000E+00

NUMBER OF FUNCTION EVALUATIONS = 5

FLOATING UNDERFLOW OCCURS.

TERMINATES BY EXCEEDING MAXIMUM NUMBER OF ITERATIONS.

ROUTINE BY BARRODALE AND WILSON WITH REALRT = .FALSE.

```
FIRST ESTIMATE = 1.0

ROOT = (1.07189010E+00, 6.31861800E-06)
FUNCTION VALUE = (-6.03094700E-04, -4.08135640E-05)
NUMBER OF FUNCTION EVALUATIONS = 40

ROOT = (1.49272000E+01, 2.85409660E-04)
FUNCTION VALUE = (-4.65393070E-04, 1.32453580E-04)
NUMBER OF FUNCTION EVALUATIONS = 65

FIRST ESTIMATE = 15.0

ROOT = (1.49282030E+01, 0.00000000E+00)
FUNCTION VALUE = (0.00000000E+00, 0.00000000E+00)
NUMBER OF FUNCTION EVALUATIONS = 5

FALSE ROOT GIVEN AT x = (1.15009130, 1.20104330E-05)
```

ROUTINE RTFS1C WITH RATIONAL INTERPOLATION

```
FIRST ESTIMATE = 1.0

ROOT = (1.07179690E+00, 0.00000000E+00)
FUNCTION VALUE = (8.04662700E-07, 0.00000000E+00)
NUMBER OF FUNCTION EVALUATIONS = 5

ROOT = (1.49282030E+01, 4.58385330E-10)
FUNCTION VALUE = (0.00000000E+00, 2.12737370E-10)
NUMBER OF FUNCTION EVALUATIONS = 56
```

---

## RESULTS FOR THE FUNCTION $x^{20} - 1$

RELATIVE TOLERANCE FOR ROOT = 1.0E-4

NUMBER OF ROOTS REQUESTED = 2

### ROUTINE BY GONNET

INITIAL ESTIMATE = 0.5

FAILS - UNABLE TO APPLY ANY METHOD

INITIAL ESTIMATE = 0.6

OVERFLOW OCCURS

ROOT = 9.99975090E-01

FUNCTION VALUE = -4.98056410E-04

NUMBER OF FUNCTION EVALUATIONS = 51

FAILS WITH REPEATED ARGUMENT VALUES

### ROUTINE BY BARRODALE AND WILSON

INITIAL ESTIMATE = 0.5

ROOT = 1.00000000E+00

FUNCTION VALUE = 0.00000000E+00

NUMBER OF FUNCTION EVALUATIONS = 5

FLOATING DIVIDE CHECK.

FAILS BY EXCEEDING MAXIMUM NUMBER OF ITERATIONS.

ROUTINE RTFS1C WITH RATIONAL INTERPOLATION

FIRST ESTIMATE = 0.5

ROOT = 9.99999890E-01

FUNCTION VALUE = 2.23517420E-06

NUMBER OF FUNCTION EVALUATIONS = 15

FALSE ROOT GIVEN AT x = -3.89568380E+00

---

RESULTS FOR THE FUNCTION $(x^2 - x - 1)/(x^2 - x + 1)$

RELATIVE TOLERANCE FOR ROOT = 1.0E-4

TOLERANCE FOR FUNCTION VALUE = 1.0E-6

INITIAL ESTIMATE = 0.0

ROUTINE BY GONNET

ROOT = -6.18034000E-01

FUNCTION VALUE = 7.45058050E-09

NUMBER OF FUNCTION EVALUATIONS = 6

FLOATING OVERFLOW OCCURS.

TERMINATES BY EXCEEDING MAXIMUM NUMBER OF ITERATIONS.

ROUTINE BY BARRODALE AND WILSON

ROOT = -6.18033980E-01

FUNCTION VALUE = -7.45058070E-09

NUMBER OF FUNCTION EVALUATIONS = 7

ROOT = 1.61803400E+00

FUNCTION VALUE = 7.45058050E-09

NUMBER OF FUNCTION EVALUATIONS = 10

ROUTINE RTFS1C WITH RATIONAL INTERPOLATION

ROOT = -6.18033990E-01

FUNCTION VALUE = 1.86264510E-09

NUMBER OF FUNCTION EVALUATIONS = 7

TERMINATES BY EXCEEDING BOUND

## APPENDIX E

## ATTAINABLE ACCURACY FOR THE ROOTS OF AN EQUATION

Suppose that the equation $f(x) = 0$ has a root of multiplicity m at $x = \alpha$.

Then $f^{(r)}(\alpha) = 0 \qquad (r = 0, 1, \ldots m-1)$

Let $\bar{f}(x)$ and $\bar{\alpha}$ denote the computed values of the function and root respectively.

Thus we may write $\bar{f}(x) = f(x) + \eta(x)$

and $\bar{\alpha} = \alpha + \varepsilon$

where $\eta(x)$ and $\varepsilon$ are perturbations due to rounding errors.

Hence $0 = f(\alpha) = f(\bar{\alpha} - \varepsilon)$

$$= f(\bar{\alpha}) - \varepsilon f'(\bar{\alpha}) + \ldots + (-1)^r \frac{\varepsilon^r}{r!} f^{(r)}(\bar{\alpha}) + \ldots$$

$$= \bar{f}(\bar{\alpha}) - \eta(\bar{\alpha}) - \varepsilon f'(\bar{\alpha}) + \ldots + (-1)^r \frac{\varepsilon^r}{r!} f^{(r)}(\bar{\alpha}) + \ldots$$

No further improvement in accuracy can be achieved when the computed function value $\bar{f}(\bar{\alpha})$ is zero.

Then $0 = -\eta(\bar{\alpha}) - \varepsilon f'(\bar{\alpha}) + \ldots + (-1)^r \frac{\varepsilon^r}{r!} f^{(r)}(\bar{\alpha}) + \ldots$

Retaining the first non-zero derivative term, we have

$$0 = -\eta(\bar{\alpha}) + (-1)^m \frac{\varepsilon^m}{m!} f^{(m)}(\bar{\alpha})$$

$$\Rightarrow \quad \varepsilon^m = \frac{(-1)^{m-1} m!\, \eta(\bar{\alpha})}{f^{(m)}(\bar{\alpha})}$$

$$\Rightarrow \quad |\varepsilon| = \left| \left[ \frac{(-1)^{m-1} m!\, \eta(\bar{\alpha})}{f^{(m)}(\bar{\alpha})} \right]^{1/m} \right|$$

Attainable accuracy is hence dependent upon the accuracy to which we can evaluate the function in the region of the root and also upon the multiplicity of the root. The occurrence of m! and the exponent 1/m will usually mean that multiple roots cannot be computed to an accuracy approaching machine precision. Multiple precision arithmetic becomes necessary in such circumstances unless a low degree of accuracy is sufficient.

EXAMPLES

1. The function x - exp(-x) has a simple zero in the interval [0,1].

   We can expect standard functions to be computed to approximately machine precision.

   Hence relative error in computed f(x) $\doteq$ u

   where u is the unit rounding error (maximum e such that 1.0 + e is represented as 1.0)

   The maximum value of exp(-x) on the interval [0,1] is 1.0 so that the absolute error in f(x) $\leq$ u.

   Also $f'(x) > 1.0, \quad x \in [0,1]$

   Hence the absolute error in the root satisfies

$$|\varepsilon| \doteq \left| \frac{\eta(\bar{\alpha})}{f'(\bar{\alpha})} \right| < u$$

The actual root is 0.567 to three significant figures so that its relative error will not exceed about 1.8u. It should therefore be possible to calculate the root to near machine precision.

2. The function $f(x) = x^3 - 3x^2 + 3x - 1$ which has a triple zero at $x = 1$.

If the nested multiplication method is used for the evaluation of the function, we have

$f(x) = ((x - 3.0)x + 3.0)x - 1.0$

Let $x = 1.0 + \varepsilon$, then

$$f(x) = ((\varepsilon - 2.0)(\varepsilon + 1.0) + 3.0)(1.0 + \varepsilon) - 1.0$$
$$= (\varepsilon^2 - \varepsilon + 1.0)(1.0 + \varepsilon) - 1.0$$
$$= (\varepsilon^3 + 1.0) - 1.0$$

This will be computed as zero if $\varepsilon^3 < u$.

For example, if u = 0.75E-8 (The approximate value for single precision on the DEC10) this gives $\varepsilon < 0.002$ approximately, so x will be accepted as a root if it lies in the approximate interval $0.998 < x < 1.002$

Dahlquist [13] points out that a root may not remain ill-conditioned if the function can be given "... in such a form that its value can be computed with less absolute error as x approaches $\alpha$". This remark is applicable to the following two examples in which polynomial functions are presented in factored form:

3. $f(x) = (x - 1.0)^3$

In this case the relative error in f(x) will be approximately 3u.

If $x = 1.0 + e$ then $f(x) = e^3$.

On the DEC10 the smallest representable number is of order 1.0E-39 so that the computed value of f(x) will remain non-zero whilst $e > u$.

Hence the absolute error in $f(x) \simeq 3ue$ .

Also $f'''(x) = 3!$ for all values of x.

It follows that $|\varepsilon| \simeq \left| \left[ \frac{(-1)^2\, 3!\, 3ue}{3!} \right]^{1/3} \right| \simeq (3u)^{1/3} |e|$

which decreases in magnitude as x approaches the root.

Convergence will thus continue until the root is computed to machine precision.

4. $f(x) = \prod_{r=1}^{20} (x - r)$

The relative error in f(x) will be approximately 20u.

Let $x = k + e$ where k is a root.

Then $\eta(x) \simeq 20ue \prod_{\substack{r=1 \\ r \neq k}}^{20} (x - r)$

When e is small we also have that $f'(x) \simeq \prod_{\substack{r=1 \\ r \neq k}}^{20} (x - r)$

so that $|\varepsilon| \simeq \left| \frac{\eta(x)}{f'(x)} \right| \simeq 20u|e|$

which again decreases as $e \rightarrow u$.

Hence the root may be calculated to machine precision.

---

# APPENDIX F

## NUMERICAL RESULTS FOR EIGENVALUE PROBLEMS

### THE STANDARD PROBLEM $Ax = \lambda x$

MAIN PROGRAM USED:

```
C
C              PROGRAM EIGEN1.FOR
C
C  SOLUTION OF THE STANDARD COMPLEX EIGENVALUE PROBLEM
C  USING DETERMINANT EVALUATION BY NAG ROUTINE F03AHF
C  AND EQUATION-SOLVING BY LASLIB ROUTINE RTFS1C.
C
      REAL X(8),A(8),ROOTS(40),STEP(2),WORK(15),TOL(4)
      COMPLEX ARRAY(20,20)
      INTEGER IC(4),IWORK(6)
C
```

```
C  SET PARAMETERS FOR RTFS1C:
      METHOD=2
      IBASE=2
      NEXTX=-3
      TOL(1)=0.0
      TOL(2)=1.0E-6
      TOL(3)=0.0
      TOL(4)=0.0
      STEP(1)=0.1
      STEP(2)=0.0
C
      WRITE(21,1)METHOD,IBASE,NEXTX,X(1),
     *           X(1),TOL(1),TOL(2),STEP(1),
     *           STEP(2)
    1 FORMAT(1H3,7X,26HRESULTS OF PROGRAM EIGEN1.//
     *       8X,8HMETHOD =,I2,14X,7HIBASE =,I2//
     *       8X,7HNEXTX =,I3//
     *       8X,18HINITIAL ESTIMATE =,F5.1,1H,F5.1//
     *       8X,8HTOL(1) =,E8.1,8X,8HTOL(2) =,E8.1//
     *       8X,6HSTEP =,F5.1,1H,F5.1//)
C
      READ(20,2)NMATS
C
C  NEXT MATRIX:
      DO 20 ITIMES=1,NMATS
      X(1)=0.0
      X(2)=1.0
```

```
      IWORK(1)=0
      IWORK(2)=0
      N=0
      NFV=0
      IFAIL=1
C
      READ(20,2)NMAX
    2 FORMAT(//8X,I3//)
      DO 4 I=1,NMAX
      READ(20,3)(ARRAY(I,J),J=1,NMAX)
    3 FORMAT(8X,8F5.1)
    4 CONTINUE
C
C  COMPUTE BOUND USING COLUMN NORM:
      BOUND=0.0
      DO 6 J=1,NMAX
      COLSUM=0.0
      DO 5 I=1,NMAX
      COLSUM=COLSUM+CABS(ARRAY(I,J))
    5 CONTINUE
      IF(COLSUM.GT.BOUND)BOUND=COLSUM
    6 CONTINUE
C
      LROOTS=2*NMAX
      MAXNFV=100*NMAX
```

```
C
      WRITE(21,8)
    8 FORMAT(1H1,8X,6HMATRIX,/)
      DO 9 I=1,NMAX
      WRITE(21,3)(ARRAY(I,J),J=1,NMAX)
    9 CONTINUE
      WRITE(21,10)NMAX,BOUND,MAXNFV
   10 FORMAT(/8X,27HNUMBER OF ROOTS REQUESTED =,I3//
     *         8X,7HBOUND =,F5.1//
     *         8X,8HMAXNFV =,I4//)
C
C  FUNCTION EVALUATION AND CALL TO ROOT-FINDER:
   11 CALL DETERM(NMAX,ARRAY,X(1),A(1),IC(1))
      CALL RTFS1C(METHOD,X,A,IBASE,IC,NMAX,N,ROOTS,
     *            LROOTS,STEP,TOL,BOUND,NEXTX,MAXNFV,
     *            NFV,WORK,IWORK,INFORM,IFAIL)
C
C  TESTS FOR CONVERGENCE OR FAILURE:
   12 IF(INFORM.EQ.1.OR.INFORM.EQ.2)GO TO 11
      IF(INFORM.GT.6)GO TO 14
      WRITE(21,13)ROOTS(2*N-1),ROOTS(2*N),
     *            IWORK(1)
   13 FORMAT(8X,12HEIGENVALUE =,F10.6,1H,F10.6,
     *       2X19HNUMBER ITERATIONS =,I3/)
      IF(INFORM.LT.0)GO TO 20
      GO TO 11
C
```

```
C  FAILURE DETECTED:
   14 IF(INFORM.EQ.8)GO TO 16
      IF(INFORM.GT.8)GO TO 18
      WRITE(21,15)
   15 FORMAT(/8X,29HCURRENT ITERATE EXCEEDS BOUND/)
      GO TO 20
   16 WRITE(21,17)
   17 FORMAT(/8X,28HMAXIMUM FUNCTION EVALUATIONS/)
      GO TO 20
   18 WRITE(21,19)INFORM
   19 FORMAT(/8X,31HFAILURE OF RTFS1C WITH INFORM =,I3/)
C
   20 CONTINUE
      STOP
      END
C
C
```

```
C  SUBROUTINE FOR DETERMINANT EVALUATION.
C
      SUBROUTINE DETERM(NMAX,ARRAY,X,DET,ID)
      REAL RINT(20),DET(2)
      COMPLEX X,ARRAY(20,20),XARRAY(20,20)
      IA=20
      IFAIL=1
C
      DO 2 I=1,NMAX
      DO 1 J=1,NMAX
      XARRAY(I,J)=ARRAY(I,J)
    1 CONTINUE
      XARRAY(I,I)=ARRAY(I,I)-X
    2 CONTINUE
C
      CALL F03AHF(NMAX,XARRAY,IA,DET(1),DET(2),ID,RINT,
     *            IFAIL)
      IF(IFAIL.EQ.0)RETURN
      DET(1)=0.0
      DET(2)=0.0
      ID=0
      RETURN
      END
```

DATA FOR PROGRAM EIGEN1:

| | MATRIX | EIGENVALUES |
|---|---|---|
| 1. | (8 x 8) | |
| | $\begin{bmatrix} -2 & 1 & 0 & \dots & 0 & 0 \\ 1 & -2 & 1 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 & -2 \end{bmatrix}$ | -0.120615, -0.467911<br>-1.000000, -1.652704<br>-2.347296, -3.000000<br>-3.532089, -3.879385 |
| 2. | $\begin{bmatrix} 5 & 4 & 1 & 1 \\ 4 & 5 & 1 & 1 \\ 1 & 1 & 4 & 2 \\ 1 & 1 & 2 & 4 \end{bmatrix}$ | 1, 2, 5, 10 |
| 3. | $\begin{bmatrix} 6 & 4 & 4 & 1 \\ 4 & 6 & 1 & 4 \\ 4 & 1 & 6 & 4 \\ 1 & 4 & 4 & 6 \end{bmatrix}$ | -1, 5, 5, 15 |
| 4. | $\begin{bmatrix} 1 & 2 & 3 & 0 & 1 & 2 \\ 2 & 4 & 5 & -1 & 0 & 3 \\ 3 & 5 & 6 & -2 & -3 & 0 \\ 0 & -1 & -2 & 1 & 2 & 3 \\ 1 & 0 & -3 & 2 & 4 & 5 \\ 2 & 3 & 0 & 3 & 5 & 6 \end{bmatrix}$ | -1.696322849<br>-1.696322851<br>0.2849864395<br>0.2849864365<br>12.41133642<br>12.41133643 |

| MATRIX | EIGENVALUES |
|---|---|
| 5. (n = 6) $\left[\begin{array}{c\|c} I_{n-1} & \begin{matrix}1\\2\\\\n-1\end{matrix} \\ \hline 1\ \ 2\ \ \ldots\ \ n-1 & n \end{array}\right]$ | 1, 1, 1, 1<br>-4.326238<br>11.326238 |
| 6. $\begin{bmatrix} 6 & -3 & 4 & 1 \\ 4 & 2 & 4 & 0 \\ 4 & -2 & 3 & 1 \\ 4 & 2 & 3 & 1 \end{bmatrix}$ | 5.236068 twice<br>0.763932 twice |
| 7. $\begin{bmatrix} 5 & 7 & 6 & 5 \\ 7 & 10 & 8 & 7 \\ 6 & 8 & 10 & 9 \\ 5 & 7 & 9 & 10 \end{bmatrix}$ | 0.01015<br>0.84311<br>3.85806<br>30.28868 |
| 8. $\begin{bmatrix} 8 & -1 & -5 \\ -4 & 4 & -2 \\ 18 & -5 & -7 \end{bmatrix}$ | 1, $2 \pm 4i$ |
| 9. $\begin{bmatrix} -1 & -1 & -1 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$ | $0.309017 \pm 0.951057i$<br>$-0.809017 \pm 0.587785i$ |

| | MATRIX | EIGENVALUES |
|---|---|---|
| 10. | $\begin{bmatrix} 1+2i & 3+4i & 21+22i \\ 43+44i & 13+14i & 15+16i \\ 5+6i & 7+8i & 25+26i \end{bmatrix}$ | -7.47753 + 6.88032i<br>6.70088 - 7.87599i<br>39.7767 + 42.99567i |
| 11. | $\begin{bmatrix} 7 & 3 & 1+2i & -1+2i \\ 3 & 7 & 1-2i & -1-2i \\ 1-2i & 1+2i & 7 & -3 \\ -1-2i & -1+2i & -3 & 7 \end{bmatrix}$ | 0, 8, 8, 12 |

---

RESULTS USING MULLER'S METHOD:

| | EIGENVALUE | NO. ITERATIONS |
|---|---|---|
| 1. | -0.120615 | 14 |
| | -1.000000 | 14 |
| | -1.652704 | 8 |
| | -2.347296 | 7 |
| | -3.000000 | 7 |
| | -3.532089 | 6 |
| | -3.879385 | 5 |
| | -0.467911 | 6 |
| 2. | 1.000000 | 9 |
| | 2.000000 | 7 |
| | 5.000000 | 5 |
| | 10.000000 | 5 |
| 3. | -1.000000 | 7 |
| | 5.000000 | 16 |
| | 5.000000 | 4 |
| | CURRENT ITERATE EXCEEDS BOUND | |
| 4. | 0.284986 | 17 |
| | 0.284986 | 5 |
| | -1.696323 - 0.000001i | 15 |
| | -1.696323 + 0.000001i | 5 |
| | 12.411336 + 0.000001i | 8 |
| | 12.411337 - 0.000001i | 4 |

| | EIGENVALUE | NO. ITERATIONS |
|---|---|---|
| 5. | 0.999983 - 0.000116i | 191 |
| | 1.000038 + 0.000018i | 89 |
| | 1.000003 + 0.000009i | 30 |
| | 0.999997 - 0.000000i | 17 |
| | -4.326238 - 0.000000i | 8 |
| | 11.326238 + 0.000000i | 6 |
| 6. | 0.763979 + 0.000004i | 109 |
| | 0.763863 + 0.000001i | 36 |
| | 5.235777 + 0.000000i | 11 |
| | 5.236055 - 0.000002i | 20 |
| 7. | 0.010150 | 9 |
| | 0.843107 | 9 |
| | 3.858057 | 6 |
| | 30.288685 | 6 |
| 8. | 1.000000 - 0.000000i | 7 |
| | 2.000000 + 4.000000i | 6 |
| | 2.000000 - 4.000000i | 6 |
| 9. | 0.309017 + 0.951057i | 7 |
| | -0.809017 + 0.587785i | 9 |
| | -0.809017 - 0.587785i | 6 |
| | 0.309017 - 0.951057i | 6 |

| | EIGENVALUE | NO. ITERATIONS |
|---|---|---|
| 10. | -7.477530 + 6.880321i | 7 |
| | 6.700876 - 7.875989i | 6 |
| | 39.776655 + 42.995667i | 6 |

11. MAXIMUM FUNCTION EVALUATIONS REACHED

---

## GENERALIZED PROBLEMS

### THE EQUATIONS CONSIDERED:

1. An equation of the form $A\underline{x} = \lambda B\underline{x}$ given by Peters and Wilkinson [36].

   A and B are of order 20 and band symmetric of width 7.

   $a_{ii} = 51 - i, \quad a_{ij} = 1, \quad 0 < |i - j| \leqslant 3$

   $b_{ii} = 41 - i, \quad b_{ij} = 1, \quad 0 < |i - j| \leqslant 3$

   Eigenvalues in the range $-10 < \lambda < 10$ are sought.

2. The problem $[(e^{\lambda} - 1)B_1 + \lambda^2 B_2 - B_0]\underline{x} = \underline{0}$ quoted by Ruhe [40], where

   $B_0 = b_0 I,$

   $B_1 = (b_{jk}^{(1)}), \quad b_{jk}^{(1)} = [n + 1 - \max(j,k)].j.k$

   $B_2 = (b_{jk}^{(2)}), \quad b_{jk}^{(2)} = n\delta_{jk} + 1/(j + k)$

   in the case $n = 8$ and $b_0 = 100$.

3. Quadratic equation $(B_0 + \lambda B_1 + \lambda^2 B_2)\underline{x} = \underline{0}$ as quoted by Ruhe [40], having

$$B_0 = \begin{bmatrix} 121.0 & 18.9 & 15.9 \\ 0.0 & 2.7 & 0.145 \\ 11.9 & 3.64 & 15.5 \end{bmatrix}$$

$$B_1 = \begin{bmatrix} 7.66 & 2.45 & 2.1 \\ 0.23 & 1.04 & 0.223 \\ 0.6 & 0.756 & 0.658 \end{bmatrix}$$

$$B_2 = \begin{bmatrix} 17.6 & 1.28 & 2.89 \\ 1.28 & 0.824 & 0.413 \\ 2.89 & 0.413 & 0.725 \end{bmatrix}$$

4. Problem of the form $(B_0 + \lambda B_1 + \lambda^2 B_2)\underline{x} = \underline{0}$, again discussed by Ruhe [40].

Here we define

$$B_0 = \begin{bmatrix} -1 + 2\alpha^2 & \alpha(1 - \alpha^2 - 2\beta^2) & 2\alpha^2\beta^2 & -\alpha\beta^2(\alpha^2 + \beta^2) \\ 2\alpha & -(\alpha^2 + 2\beta^2) & 2\alpha\beta^2 & -\beta^2(\alpha^2 + \beta^2) \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$B_1 = \begin{bmatrix} 3\alpha & -(1 + \alpha^2 + 2\beta^2) & \alpha(1 + 2\beta^2) & -\beta^2(\alpha^2 + \beta^2) \\ 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \end{bmatrix}$$

$$B_2 = I$$

where $\alpha$ is a non-negative parameter and $\beta = 1 + \alpha$.
The case $\alpha = 0$ gives triple eigenvalues at $\pm i$ and a double eigenvalue at 0.

---

RESULTS USING ROUTINE RTFS1C AND NAG DETERMINANT ROUTINES:

1. METHOD - RATIONAL INTERPOLATION

BASE FOR FUNCTION EVALUATION = 2

INITIAL ESTIMATE = 1.0

TOLERANCE FOR RELATIVE ERROR IN ROOT = 1.0E-6

EACH SEARCH COMMENCED FROM PREVIOUS ROOT WITH STEP LENGTH = 0.01

NUMBER OF ROOTS REQUESTED = 20

MAXIMUM NUMBER OF FUNCTION EVALUATIONS = 400

BOUND = 10.0

| EIGENVALUE | NO. ITERATIONS |
|---|---|
| 1.236230 | 27 |
| 1.261924 | 7 |
| 1.285635 | 11 |
| 1.312505 | 11 |
| 1.345003 | 22 |
| 1.357572 | 9 |
| 1.371315 | 6 |
| 1.386684 | 8 |
| 1.403472 | 8 |
| 1.422235 | 6 |
| 1.447517 | 6 |
| 1.470427 | 6 |
| 1.495213 | 7 |
| 1.333394 | 17 |

| EIGENVALUE | NO. ITERATIONS |
|---|---|
| 1.322600 | 10 |
| 1.277398 | 15 |
| 1.303011 | 6 |
| 1.294097 | 7 |
| 1.269440 | 10 |
| 1.254381 | 5 |

The results obtained agree, to the six decimal places quoted, with those given by Peters and Wilkinson, but were computed in the order given above.

---

2. METHOD - RATIONAL INTERPOLATION

BASE FOR FUNCTION EVALUATION = 2

INITIAL ESTIMATE = 0.0

TOLERANCE FOR RELATIVE ERROR IN ROOT = 1.0E-6

EACH SEARCH COMMENCED FROM ROOT JUST FOUND WITH STEP LENGTH = 0.1

NUMBER OF ROOTS REQUESTED = 16

MAXIMUM NUMBER OF FUNCTION EVALUATIONS = 320

BOUND = 100

| EIGENVALUE | NO. ITERATIONS |
|---|---|
| 0.217461 | 6 |
| 0.884962 | 10 |
| 1.394724 | 9 |
| 1.726304 | 12 |
| 2.007944 | 7 |
| 2.335425 | 7 |
| 2.731077 | 6 |
| 3.182596 | 8 |
| -3.491853 | 40 |
| -3.801275 | 8 |
| -3.968169 | 9 |
| -4.521556 | 9 |
| -3.702762 | 14 |

| EIGENVALUE | NO. ITERATIONS |
|---|---|
| -3.627468 | 9 |
| -3.571756 | 6 |
| -7.642558 | 7 |

The roots agree, to six decimal places, with the values given by Ruhe and were obtained in the order given above.

---

3. METHOD - QUADRATIC INTERPOLATION

BASE FOR FUNCTION EVALUATION = 2

INITIAL ESTIMATE = -1 + i

TOLERANCE FOR RELATIVE ERROR IN ROOT = 1.0E-6

EACH SEARCH COMMENCED FROM COMPLEX CONJUGATE OF ROOT JUST FOUND USING STEP = 0.1 + 0.1i

NUMBER OF ROOTS REQUESTED = 6

MAXIMUM NUMBER OF FUNCTION EVALUATIONS = 120

BOUND = 10.0

| EIGENVALUE | NO. ITERATIONS |
|---|---|
| -0.917998 + 1.760584i | 7 |
| -0.917998 - 1.760584i | 4 |
| 0.094722 + 2.522877i | 8 |
| 0.094722 - 2.522877i | 4 |
| -0.884830 + 8.441512i | 6 |
| -0.884830 - 8.441512i | 4 |

These results agree, to six decimal places, with those given by Ruhe and were obtained in the order shown above.

---

4. The actual roots are as follows:

$0, \quad -\alpha, \quad \pm i, \quad \pm(1+\alpha)i \quad$ and $\quad -\alpha \pm (1+\alpha)i.$

For each value of $\alpha$ the following input parameters were set:

METHOD - QUADRATIC INTERPOLATION

BASE FOR FUNCTION EVALUATION = 2

EACH SEARCH WAS COMMENCED FROM THE COMPLEX CONJUGATE OF THE ROOT JUST FOUND USING STEP 0.1 + 0.1i

NUMBER OF ROOTS REQUESTED = 8

BOUND = 10.0

$\alpha = 0.5$

INITIAL ESTIMATE = -1 - 2i

RELATIVE TOLERANCE FOR ROOT = 1.0E-6

MAXIMUM NUMBER OF FUNCTION EVALUATIONS = 320

| EIGENVALUE | NO. ITERATIONS |
|---|---|
| 0.000000 - 1.500000i | 15 |
| -0.000000 + 1.500000i | 4 |
| -0.000000 - 1.000000i | 11 |
| -0.000000 + 1.000000i | 4 |
| 0.000000 + 0.000000i | 13 |
| -0.500000 - 0.000000i | 260 |
| -0.500000 - 1.500000i | 6 |
| -0.500000 + 1.500000i | 4 |

$\alpha = 0.1$

INITIAL ESTIMATE = -1 - i

RELATIVE TOLERANCE FOR ROOT = 1.0E-6

MAXIMUM NUMBER OF FUNCTION EVALUATIONS = 800

| EIGENVALUE | NO. ITERATIONS |
|---|---|
| -0.000000 - 1.100000i | 20 |
| -0.100000 + 1.100000i | 11 |
| -0.100000 - 1.100000i | 4 |
| -0.000000 + 1.100000i | 9 |
| 0.000000 - 1.000000i | 8 |
| 0.000000 + 1.000000i | 9 |
| -0.100000 - 0.000000i | 8 |

MAXIMUM FUNCTION EVALUATIONS REACHED

$\alpha = 0.0$

INITIAL ESTIMATE = -0.01 - 1.01i

RELATIVE TOLERANCE FOR ROOT = 1.0E-6 OR

AUTOMATIC STOPPING WITH TOLERANCE = 0.01

MAXIMUM NUMBER OF FUNCTION EVALUATIONS = 800

| EIGENVALUE | NO. ITERATIONS |
|---|---|
| 0.004300 - 1.001796i | 15 |
| -0.000251 + 0.998630i | 24 |
| 0.000241 - 0.999292i | 14 |
| -0.000288 + 1.000035i | 14 |
| -0.000279 - 1.000037i | 5 |
| -0.000016 + 1.000204i | 10 |
| 0.000000 - 0.000001i | 21 |
| 0.000000 + 0.000000i | 5 |

The results reflect the increasingly ill-conditioned nature of the problem as $\alpha \rightarrow 0$.

## APPENDIX G

## APPLICATION OF THE SINGULAR VALUE DECOMPOSITION

### The Singular Value Decomposition

The singular values of an m x n matrix A are defined as

$\sigma_i(A) \geq 0$ such that $\sigma_i^2(A) = \lambda_i(A^H A)$ [i = 1,2,...,n] (1)

It can be shown that A may be factorized in the form $A = U \Sigma V^H$ where U and V are m x m and n x n unitary matrices respectively and $\Sigma$ is an m x n matrix with

$$\left.\begin{aligned} \Sigma_{ii} &= \sigma_i \\ \Sigma_{ij} &= 0 \end{aligned}\right\} \quad \begin{aligned} &[i = 1,2,\ldots,k \text{ and } k = \min(m,n)] \\ &\text{otherwise.} \end{aligned}$$

If A is of rank r we have, further, that

$\sigma_{r+1}, \ldots, \sigma_k = 0$.

Also $A^H A = V \Sigma^2 V^H$ in accordance with (1) above.

In the real case the factorization may be carried out using equivalence transformations so as to produce the singular values in the order $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_k$.

An example of such a procedure is provided by NAG routine F02WAF [33] which performs the following steps:

1. Reduction to upper triangular form by means of Householder transformations

2. Further reduction to bidiagonal form by a sequence of Given's plane rotations

3. Iterative use of the QR algorithm to obtain an approximation, to the desired accuracy, to diagonal form

Acceptance of Roots

In generalized eigenvalue problems we seek values of a variable parameter $\beta$ such that $A(\beta)$ is singular. There will then exist at least one zero singular value.
Thus, in practical computation, $\sigma_k$ will be accepted as a root if the ratio $\sigma_k / \sigma_1$ is sufficiently small.
For the dielectric tube problem considered at the National Physical Laboratory [15] it was found that, for valid roots, $\sigma_k / \sigma_1 <$ macheps, where macheps, the unit rounding error, was of order 1.0E-16.
The false roots claimed occasionally as a result of the automatic stopping criterion were detected by much larger values of this ratio (typically of order 1.0E-3).

## Calculation of the Eigenvectors

In the real case, if $\beta$ is an eigenvalue of A and $\underline{x}$ is the corresponding eigenvector, then $A(\beta)\underline{x} = \underline{0}$ and $A = Q\Sigma P^T$ where P, Q are orthogonal.

Thus $Q\Sigma P^T\underline{x} = \underline{0} \Rightarrow \Sigma P^T\underline{x} = \underline{0}$

Putting $P^T\underline{x} = \underline{y}$ we have that $\Sigma\underline{y} = \underline{0}$

But $\Sigma$ may be partitioned as $\left[\begin{array}{c|c} D & 0 \\ \hline 0 & 0 \end{array}\right]$

where D is r x r diagonal.

Similarly, $\underline{y}$ may be written $\begin{bmatrix} \underline{y}_1 \\ \underline{y}_2 \end{bmatrix}$

where $\underline{y}_1$ is of dimension r.

Hence $\Sigma\underline{y}_1 = \underline{0} \Rightarrow \underline{y}_1 = \underline{0}$ and an arbitrary choice may be made for $\underline{y}_2$.

In particular, if we set $\underline{y}_2$ equal to the elementary unit vector $\underline{e}_i$, we have that $\underline{x} = P\underline{y}$

$\Rightarrow \underline{x} = \underline{p}_i$ (the ith. column of P) [i = r+1, ..., k]

A set of eigenvectors may thus be read directly from the details of the singular value decomposition.

---

## REFERENCES


[1] Acton, F,S. 1970 Numerical Methods that Work, New York (Harper & Row)

[2] Ahlfors, L.V. 1953 Complex Analysis, New York (McGraw-Hill)

[3] Bach,H. 1969 "Complex Root Finding", Collected Algorithms from CACM, 365

[4] Bach,H. 1969 "On the Downhill Method", Comm.ACM 12, 12, 675-684

[5] Bahar,E. & Fitzwater,M. 1981 "Numerical Technique to Trace the Loci of the Complex Roots of Characteristic Equations", SIAM J.Sci.Stat.Comput., 2, 4, 389-403

[6] Barrodale,I & Wilson,K.B. 1978 "A Fortran Program for solving a Nonlinear Equation by Muller's Method", J. of Comp. & App.Math., 4, 2, 159-166

[7] Barzilai,J. & Ben-Tal,A. 1982 "Nonpolynomial and Inverse Interpolation for Line Search: Synthesis and Convergence Rates", SIAM J. Numer.Anal., 19, 6, 1263-1277

[8] Brent,R.P. 1971 "An Algorithm with Guaranteed Convergence for finding a Zero of a function", Computer J., 14, 4, 422-425

[9] Brent,R., Winograd,S. & Wolfe,P. 1973 "Optimal Iterative Processes for Root-Finding", Numer.Math.,20, 327-341

[10] Bus,J.C.P. & Dekker,T.J. 1975 "Two Efficient Algorithms with Guaranteed Convergence for finding a Zero of a Function", ACM Trans.Math.Software 1, 4, 330-345

[11] Cox,M.G. & Lehrian,D.E. 1973 "Real Procedure Zero", NPL Algorithms Library, National Physical Laboratory C5/01/0/Algol60/1/73

[12] Crawford,C.R. 1973 "Reduction of a Band-Symmetric Generalized Eigenvalue Problem", Comm.ACM 16, 41-44

[13] Dahlquist,G. & Bjorck,A. 1974 Numerical Methods, Englewood Cliffs N.J. (Prentice-Hall)

[14] Delves,L.M. & Lyness,J.N. 1967 "A Numerical Method for Locating the Zeros of an Analytic Function", Math.Comp. 21, 561-577

[15] Ferriss,D.H., Hammarling,S.J., Martin,D.W. & Wareham,A.G.P. 1983 "Numerical Solution of Equations describing Electromagnetic Propagation in Dielectric Tube Waveguides", NPL Report, National Physical Laboratory DITC 16/83

[16] Friedli,A. 1973 "Optimal Covering Algorithms in Methods of Search for Solving Polynomial Equations", J.ACM 20, 290-300

[17] Forsythe,G.E., Malcolm,M.A. & Moler,C.B. 1977 Computer Methods for Mathematical Computations, Englewood-Cliffs N.J. (Prentice-Hall)

[18] Gonnet,G.H. 1977 "On the Stucture of Zero Finders", BIT 17, 170-183

[19] Gregory,R.T. & Karney,D.L. 1969 A Collection of Matrices for Testing Computational Algorithms, New York (Wiley)

[20] Hammarling,S.J., Kenward,P.D. & Symm,H.J. 1981 "Subroutine RTFS1C/Z", Linear Algebra Subroutine Library, National Physical Laboratory

[21] Henrici,P. 1974 Applied and Computational Complex Analysis Vol.1, New York (Wiley)

[22] Householder,A.S. 1970 The Numerical Treatment of a Single Non-Linear Equation, New York (McGraw-Hill)

[23] Ince,E.L. 1956 Ordinary Differential Equations, New York (Dover)

[24] Jarratt,P. & Nudds,D. 1965 "The use of Rational Functions in the Iterative Solution of Equations on a Digital Computer", Computer J. 8, 62-65

Kronsjo,L.I. 1979 Algorithms: Their Complexity and Efficiency, Chichester (Wiley)

Kublanovskaya,V.N. 1970 "On an Approach to the Solution of the Generalized Latent Value Problem for $\lambda$-matrices", SIAM J.Numer.Anal. 7, 4, 532-537

Kung,H.T. & Traub,J.F. 1974 "Optimal Order of One-Point and Multi-Point Iteration", J.ACM 21, 4

Lancaster,P. 1977 "A Review of Numerical Methods for Eigenvalue Problems Nonlinear in the Parameter", ISNM 38, Basel und Stuttgart (Birkhauser Verlag) 43-67

Larkin,F.M. 1964 "A Combined Graphical and Iterative Approach to the Problem of finding Zeros of Functions in the Complex Plane", Computer J. 7, 212-219

Lehmer,D.H. 1961 "A Machine Method for Solving Polynomial Equations", J.ACM 8, 151-162

Milne-Thompson,L.M. 1933 The Calculus of Finite Differences, London (Macmillan)

Muller,D.E. 1956 "A Method of Solving Algebraic Equations using an Automatic Computer", Math.Tables Aids Comput. 10, 208-215

NAG Fortran Library Mark 9 Vols.1 & 4, Oxford (Numerical Algorithms Group)

[34] Nerinckx,D. & Haegemans,A. 1976 "A Comparison of Non-Linear Equation Solvers", J.Comp.& App.Math. 2, 2, 145-148

[35] Ostrowski,A.M. 1960 Solution of Equations and Systems of Equations, New York (Academic Press)

[36] Peters,G. & Wilkinson,J.H. 1969 "Eigenvalues of $A\underline{x} = \lambda B\underline{x}$ with Band Symmetric A and B", Computer J. 12, 4, 398-404

[37] Peters,G. & Wilkinson,J.H. 1970 $A\underline{x} = \lambda B\underline{x}$ and the Generalized Eigenproblem", SIAM J.Numer.Anal. 7, 4, 479-492

[38] Robinson,S.M. 1979 "Quadratic Interpolation is Risky", SIAM J.Numer.Anal. 16, 3, 377-379

[39] Rodman,R.D. "Muller's Method for finding Roots of an Arbitrary Function", Collected Algorithms from CACM 196

[40] Ruhe,A. 1973 "Algorithms for the Nonlinear Eigenvalue Problem", SIAM J.Numer. Anal. 10, 4, 674-689

[41] Spira,R. 1967 "Zeros of Approximate Functional Approximations", Math.Comp. 21, 41-48

[42] Terray,J. & Lancaster,P. 1975 "A Boundary Value Problem from the Study of Heat Transfer", ISNM 27 Basel und Stuttgart (Birkhauser Verlag) 303-308

[43] Traub,J.F. 1964 Iterative Methods for the Solution of Equations, Englewood Cliffs N.J. (Prentice-Hall)

[44] Ward,J.A. 1957 "The Downhill Method of Solving f(z) = 0", J.ACM 4, 2, 148-150

[45] Wehl,H. 1924 "Randbemerkungen zu Hauptproblemen der Mathematik,II. Fundamentalsatz der Algebra und Grundlagen der Mathematik." Math.Z. 20, 131-150

[46] Wilkinson,J.H. 1963 Rounding Errors in Algebraic Processes, London (H.M.S.O.)

[47] Wilkinson,J.H. 1965 The Algebraic Eigenvalue Problem, Oxford (Clarendon Press)

[48] Wilkinson,J.H. Notes on Differential Equations and Eigenvalue Problems (unpublished)

[49] Wittrick,W.H. & Williams,F.W. 1971 "A General Algorithm for Computing Natural Frequencies of Elastic Structures", Quart.J.Mech.& App.Math. 24, 3, 264-284